Journal of
*Sensor and Actuator Networks*

MDPI

*Article*

# Intelligent IoT Platform for Agroecology: Testbed

**Naila Bouchemal [1,*], Nicola Chollet [1] and Amar Ramdane-Cherif [2]**

[1] ECE—Ecole d'ingénieurs—Campus de Paris 1, 10 Rue Sextius Michel, 75015 Paris, France; nicolas.chollet@ece.fr
[2] Laboratoire d'Ingénierie des Systèmes de Versailles (LISV), 10 Av. de l'Europe, 78140 Vélizy-Villacoublay, France; rca@lisv.uvsq.fr
* Correspondence: naila.bouchemal@ece.fr

**Abstract:** Smart farming is set to play a crucial role in the sustainable transformation of agriculture. The emergence of precision agriculture, facilitated by Internet of Things (IoT) platforms, makes effective communication among the various sensors and devices on farms essential. The development of smart sensors that utilize artificial intelligence (AI) algorithms for advanced edge computations only intensifies this need. Moreover, once data are collected, farmers frequently find it challenging to apply them effectively, especially in alignment with agroecological principles. In this context, this paper introduces an energy-efficient platform for embedded AI sensors that leverages the LoRaWAN network, along with a knowledge-based system to aid farmers in decision-making rooted in sensor data and agroecological practices. This paper focuses on the deployment of an end-to-end IoT platform that integrates a wireless sensor network (WSN), embedded AI, and a knowledge base.

**Keywords:** agriculture; AI; IoT; LoRaWAN; smart farming; TinyML



**Citation:** Bouchemal, N.; Chollet, N.; Ramdane-Cherif, A. Intelligent IoT Platform for Agroecology: Testbed. *J. Sens. Actuator Netw.* **2024**, *13*, 83. https://doi.org/10.3390/jsan13060083

Academic Editor: Jordi Mongay Batalla

Received: 30 September 2024
Revised: 19 November 2024
Accepted: 21 November 2024
Published: 2 December 2024



## 1. Introduction

Agriculture is the cornerstone of human civilization, marking the transition from a nomadic hunter–gatherer lifestyle to a settled, city-based existence. Historically, agriculture was labor-intensive, with low productivity and high sensitivity to climatic events, requiring many small farms, and at least a third of the population engaged in primary agricultural production.

During the late 19th and early 20th centuries, agriculture underwent significant transformation, known as Agriculture 2.0 or the Green Revolution. This modern agriculture is characterized by the extensive use of heavy machinery, chemical fertilizers, and chemical protections (insecticides and pesticides). Today, it dominates global agriculture, enabling farmers to produce massive quantities of food with minimal human effort and high adaptability to climate challenges. Modern agriculture has significantly contributed to the world's food supply, supporting the growing global population.

However, modern agricultural practices have also introduced numerous challenges, posing serious environmental, health, and economic risks, thereby threatening global food security. Food security [1] means ensuring that all individuals have physical, social, and economic access to sufficient, safe, and nutritious food to meet their dietary needs and preferences for an active and healthy life. This concept encompasses the availability, accessibility, affordability, and quality of food.

In fact, large-scale farming practices have led to soil degradation, water pollution, and deforestation. Moreover, modern agriculture heavily relies on synthetic fertilizers, pesticides, and herbicides, which negatively affect the environment. The extensive use of these chemicals has caused soil and water contamination and has led to the development of pesticide-resistant insects and weeds [2].

In addition to its environmental harms, modern agriculture is ill-equipped to address climate change. Numerous studies have shown that the Earth has been warming since the

mid-19th century due to human activity and the production of greenhouse gases. Even a modest rise in temperature will disrupt rainfall patterns and increase the frequency and severity of extreme weather events, such as floods, heatwaves, fires, and monsoons. This will lead to significant and uneven shifts in climate patterns worldwide. Some regions will experience a drastic reduction in their food production capabilities, threatening food security, especially in developing countries where smallholder farmers are more vulnerable to adverse weather conditions [3].

Addressing these issues requires adopting more efficient farming practices that produce more food with fewer resources.

This leads to the central question: How can we enhance farmers' ability to use smart-farming tools to implement sustainable agroecological practices?

Fortunately, the digital revolution has introduced numerous tools to make agriculture more sustainable. The field dedicated to integrating recent technologies into farming is known as Smart Farming or Precision Agriculture. This approach uses advanced technology to improve agricultural production, increase yields, reduce waste and resource usage (both mechanical and chemical), and enhance the overall reliability of food production.

In this context, we view Smart Farming as a solution for addressing this question.

Smart Farming is already being widely implemented and is expected to become even more prevalent in the future. This progression has led to Agriculture 3.0, which incorporates digital tools for automation and monitoring. It is now evolving into Agriculture 4.0, with the integration of advanced information and communication technology (ICT) tools, such as artificial intelligence (AI) and the Internet of Things (IoT). Concretely, smart farming involves integrating various technologies such as sensing technologies, data analytics, artificial intelligence, and robotics. A general view of the smart farming processes is proposed in Figure 1.

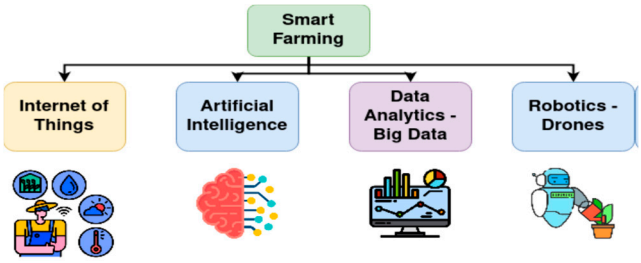


**Figure 1.** Smart Farming technologies.

## 2. Related Works

All technologies of an end-to-end IoT platform are generally used to assist the farming process. IoT platforms are robust tools for collecting and interpreting data to make informed decisions based on specific criteria. Besides that, embedded AI is a burgeoning field of artificial intelligence (AI). It aims to focus on optimizing machine learning algorithms to run on hardware with limited computational power and memory, such as microcontroller units (MCUs) or embedded central processing units (CPUs). Thus, it reduces latency, enhances privacy, and minimizes energy consumption.

This paper centers on deploying an end-to-end IoT platform (Figure 2) integrated with a small-scale wireless sensor network (WSN), embedded AI, and knowledge base. The aim is to help farmers in adhering to agroecological principles, which rely heavily on observing and interpreting the behaviors of the farm's biotope and biocenosis.

One area that could significantly advance agriculture is the application of AI-powered computer vision for assessing crop growth and finding issues such as pest infestations or weed proliferation. These AI algorithms typically depend on cloud computing due to their substantial processing requirements. This reliance needs specialized network architectures and raises concerns about privacy, security, and latency, while also increasing energy demands due to the need for high data transfer rates. However, battery life is a

critical factor in agriculture since sensors are often deployed over vast distances without reliable access to electricity.

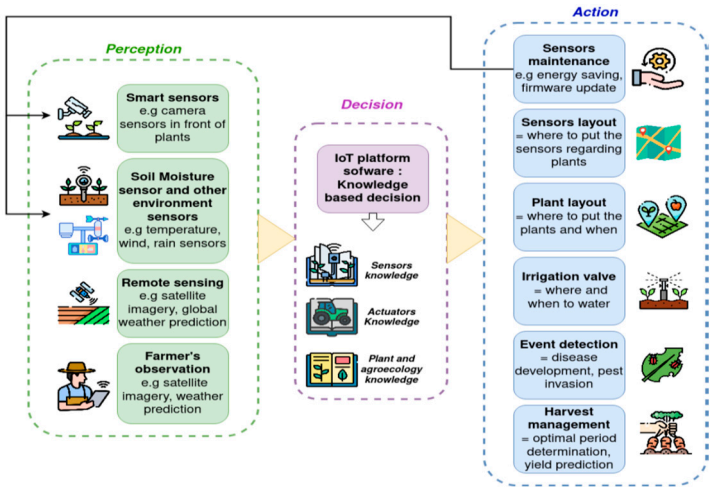


**Figure 2.** End-to-end IoT platform.

Meanwhile, recent spectacular progress in computational technologies has led to an unprecedented boom in the field of artificial intelligence (AI). AI is now used in a plethora of research areas and has shown its capability to bring novel approaches and solutions to various research problems. However, the extensive computation needed to train AI algorithms comes with a cost. Driven by the need to reduce energy consumption, the carbon footprint and the cost of computers running machine learning algorithms, Tiny Machine Learning (TinyML) is nowadays considered as a promising AI alternative focusing on technologies and applications (Figure 3) for extremely low-profile devices. This is the case for agriculture [4].

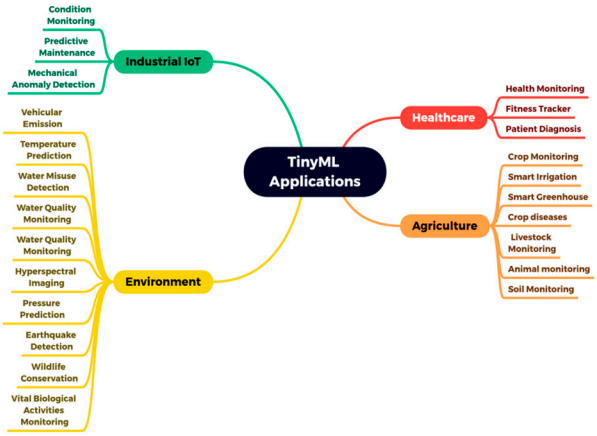


**Figure 3.** TinyML applications.

Research on Tiny ML is relatively nascent but holds significant promise for a variety of applications. Within the agricultural sector, the potential use cases are extensive, including livestock management and insect detection [5]. However, implementations in smart farming remain limited, although we predict growth in this area in the future. Most current research is focused on industrial applications; a comprehensive survey of recent advancements in TinyML can be found in the work of the authors in [6].

In addition to agricultural applications, TinyML has been employed in diverse scenarios, such as adaptive traffic control [7] and wildlife conservation [8]. Furthermore, TinyML presents numerous advantages over Fog, Edge, and Cloud computing, particularly in terms of privacy, security, latency, and energy efficiency, as discussed by the authors in [9].

The authors in [10] designed an embedded ML pipeline that helps farmers and scientists to check the health of the crop and its growth. This pipeline allows users to create an embedded ML that can be used for different plants in labs, greenhouses, farms, or gardens. The first step of the pipeline is data collection, where the authors proposed best practices to collect data plants. The next step is training a convolution neural network (CNN) for two cases: (a) estimation of the leaf area index (LAI) and (b) prediction of the plant growth stage. After the training phase, the ML model is compressed and converted to TensorFlow Lite (TFLite) format to be deployed on an MCU device. For testing, the authors have chosen Sony Spresense setup as target device. The authors in [11] proposed a TinyML solution to detect drought stress in soybeans. The system is composed of a Raspberry Pi zero W and Sony IMX219 camera module. The Raspberry device runs the CNN model on the captured image to detect crop drought stress and then sends prediction to a web platform. The CNN model was converted to TFLite format in order to be deployed on a limited-resources device. The authors in [10] proposed a low-power and real-time image detector for grape leaf esca diseases based on a compressed CNN model. Many compressing techniques such as CP decomposition, tucker decomposition, and tSVD are analyzed to choose the method with the best compression factor and accuracy. CP decompositions were chosen and applied on the CNN model. After the training and validation of the model, it was compressed with post-training quantization using TFLite to generate a model with 8bits. The compressed model is deployed on OpenMV Cam STM32H7. The device is mounted on an agricultural vehicle moving at a constant speed through the cultivation field. The authors in [12] proposed a system for agri-environmental management employing moisture sensors and real-time video analysis of soil photographs. The VGG-19 model is used to distinguish picture types and calculate the quantity of water needed for irrigation based on the kind of crop planted. The model is trained on a Kaggle soil structure dataset and evaluated on a proprietary dataset. The authors in [13] proposed a low-cost device based on the MCU ESP32-CAM that uses a camera to gather data from numerical water meters to check central pivot irrigation systems. The device runs a TinyML model to process the images to read the water meter; it is then transmitted to a server using LoRaWAN. The TinyML model scored an accuracy of 88%.

## 3. Our Proposal

To evaluate the feasibility and efficiency of our proposed architecture, we conducted a study using an agricultural use case focused on monitoring strawberries in two operational modes: normal behavior and degraded condition mode.

In the normal behavior mode (Figure 4), an intelligent, battery-powered camera sensor with embedded TinyML algorithms is deployed on the sensors field. Its primary function is to detect and count strawberry fruits. Once the count is decided, the sensor transmits this data via a LoRaWAN network to an application server hosted in the cloud. This server houses a decision platform that processes the fruit count alongside other relevant factors.

Decisions based on this data may include initiating fertilizer applications if the fruit count is below expectations, implementing measures against potential diseases or pests, detecting theft by humans or wildlife, or issuing a harvest notification when the fruits reach maturity. The decision directives are then communicated to the executor, who could be the farmer or an autonomous robot, to improve their actions. For instance, a human or robotic operator will only move for harvest when the fruits are ripe.

After training the TinyML model, if the confidence score is lower than a threshold, then it switches to degraded mode (Figure 5). In the degraded mode, the images captured by the camera of the sensor must be transferred to the server; the TinyML model is then re-trained on the server side, and then re-loaded onto the sensor according to the firmware update over the air procedure (FUOTA).

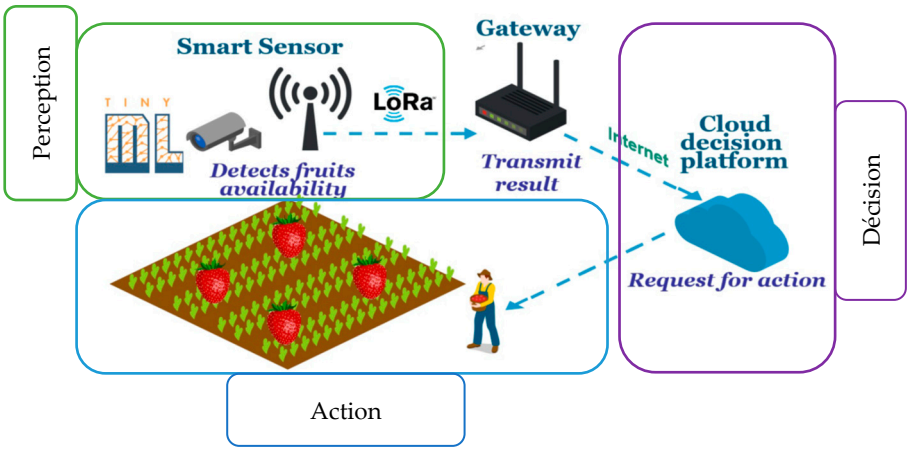


**Figure 4.** Normal mode.

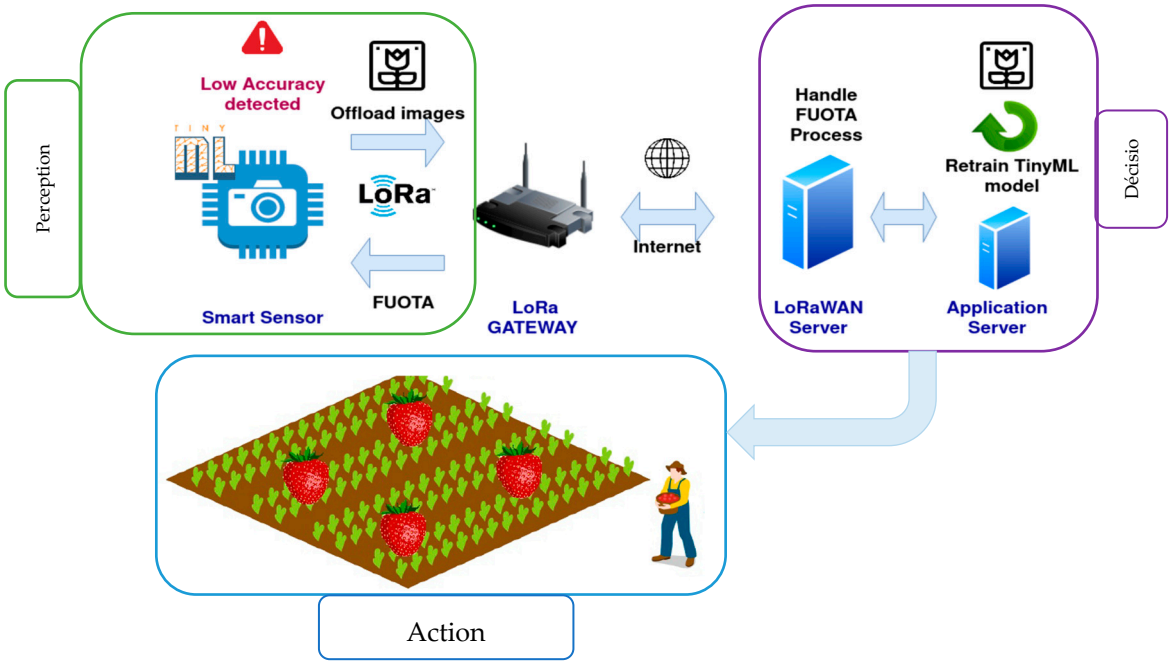


**Figure 5.** System in degraded condition mode.

The entire process is depicted in Figure 6, with steps for normal behavior mode highlighted in blue and those for degraded conditions mode highlighted in red.

### *3.1. Normal Behavior Mode*

#### 3.1.1. Data Collection

To start the training of our model, the first step involves gathering data. While datasets like the comprehensive fruit dataset from Kaggle [14] can be used, the preferred approach in embedded machine learning is to collect data directly from the devices. This is particularly beneficial because embedded camera modules often have varying quality levels. In our specific scenario, initial datasets were obtained using the camera device. We employed the Edge Impulse tool [15] to capture images directly from devices such as the Arduino Portenta [16] and the ESP32 cam [17], which share the same camera module as the STM32 [18]. Once collected, the data must undergo labeling. See Figure 7.

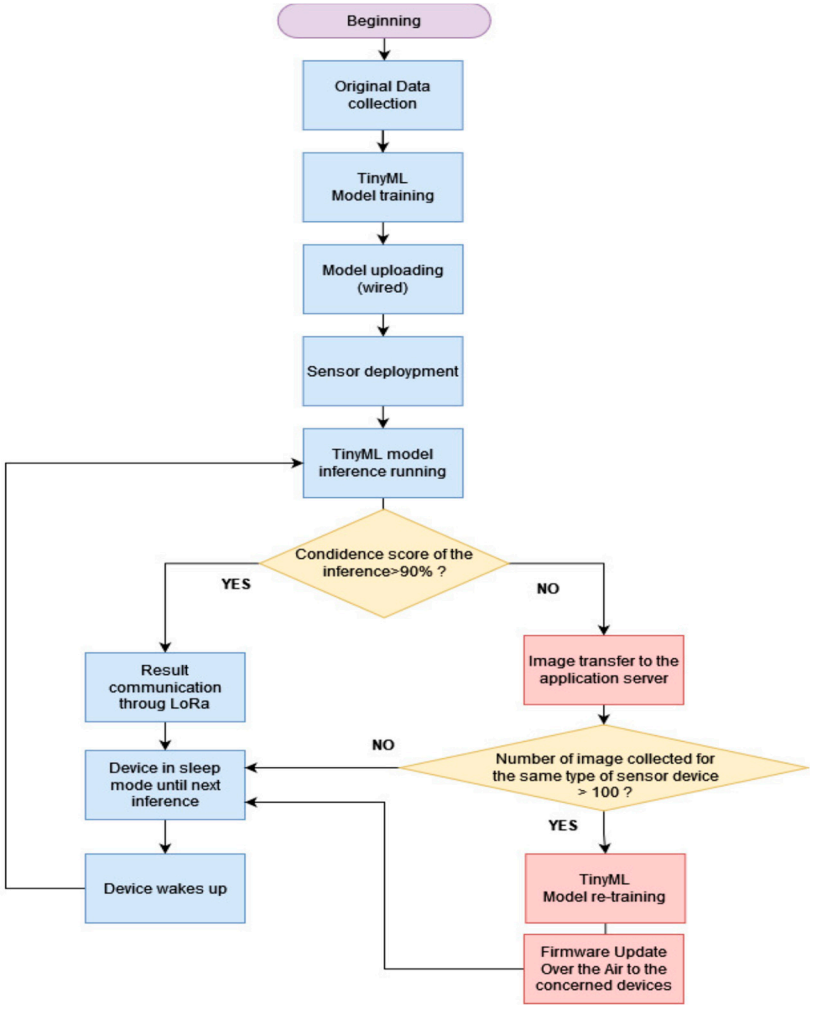


**Figure 6.** End-to-end process platform.

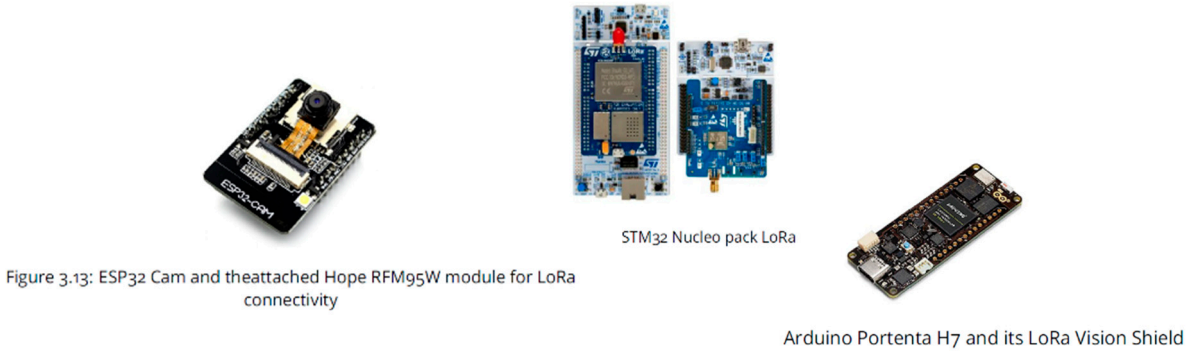


**Figure 7.** Hardware components.

### 3.1.2. TinyML Model Training

Our model was trained to count the number of fruits in an image using the FOMO (Faster Object, More Object) algorithm developed by Edge Impulse. FOMO is an innovative machine-learning algorithm designed for object detection on highly constrained devices. It enables devices to count objects, find their locations within an image, and track multiple objects in real time, consuming up to 30 times less processing power and memory compared to other similar algorithms like MobileNet SSD or YOLOv5 [19], which are also suitable for constrained devices.

The FOMO algorithm runs based on heatmap determination, where the original image is divided into smaller sections. Each section undergoes binary classification to find the presence or absence of the target object being counted. (Figure 8).

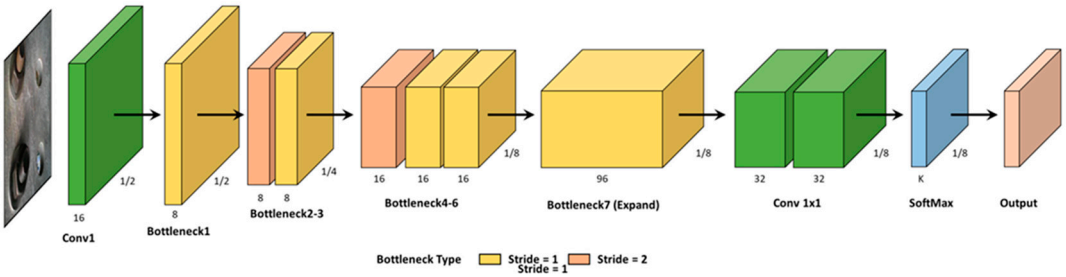


**Figure 8.** FOMO algorithm.

Following training, the model undergoes testing across multiple datasets to assess its accuracy. Since accuracy in machine learning refers to the proportion of correct predictions made by a model compared to the total number of predictions on a given data set, we set a minimum accuracy threshold of 90%. The precision in our application is not critical.

### 3.1.3. Model Uploading

After the first model is trained, it is uploaded to the device via a wired connection. The overall process of data collection, model training, and deploying according to Edge Impulse is depicted in Figure 9.

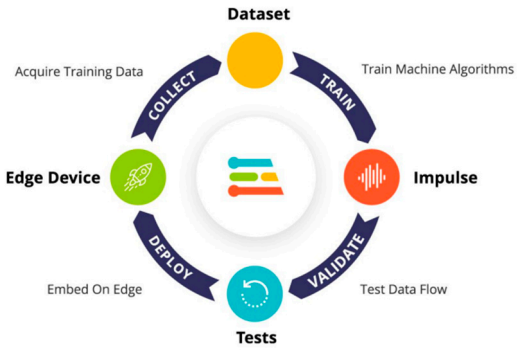


**Figure 9.** TinyML edge impulse model creation process.

### 3.1.4. On-Device Inferring

As soon as it is deployed, the device will begin making predictions. For our application, which involves counting strawberry fruits, we set the number of inferences performed to one per day, although the frequency can be adjusted according to the application's needs. If the confidence score is higher than 90%, then the device sends the results to the server according to step 5. Otherwise, the system passes to degraded mode. Every time the device runs the TinyML algorithms, it assigns a confidence score to the prediction. In machine learning, and specifically in the context of convolutional neural networks (CNNs) used for classification tasks, a confidence score is a value that stands for the model's certainty about its prediction. After processing input data, the model generates a prediction, in our case, of the estimated number of fruits. The confidence score (Cs), usually a value between zero and one, shows how certain the model is that its prediction is correct, with a score closer to one indicating higher confidence. For example, in our specific case, the prediction might output three strawberries with a confidence score of 0.95, meaning that the device is highly certain.

However, if it counts four strawberries with a confidence score of 0.45, this indicates that this result should not be considered. In the case of successive inference processes that have low confidence scores, this might show that the model needs to be re-adapted.

In situations where multiple predictions in a row (number to be decided by the application, in our case we set it to four) have low confidence scores, the system enters degraded condition mode.

#### 3.1.5. Result Transmission

Once the result is obtained, it is transferred to the application server via the LoRaWAN network. The message consists of a single packet, with only the number of fruits included in the payload as an integer. After transmission, the device waits for an acknowledgment of reception from the gateway. If it does not receive it, it retries sending the message.

#### 3.1.6. Result Interpretation

The application server can interpret the data. Even if it only received a number, extra information such as the localization, the number of recent passages, or the historical data values of the sensors can be retrieved from the knowledge implemented base.

#### 3.1.7. Sleep Procedure

Once the device finishes transmission, it goes back in sleep mode. Sleep mode in microcontrollers refers to a low-power state that conserves energy when the device is inactive or when full power is not necessary. In sleep mode, certain functions or sections of the microcontroller are turned off or used in a reduced-power state to minimize energy consumption, thereby extending the battery life of battery-operated devices. The exit from sleep mode can occur due to a scheduled procedure that is induced at the end of a timer, or due to an external signal. In our case, it is triggered by a timer.

### *3.2. Degraded Condition Mode*

In degraded condition mode, the phases follow the device inference.

#### 3.2.1. Image Data Transmission (Heavy Data Offload)

To address the issue of local drift, the system needs to train another model; therefore, the application server needs new data. We propose the following algorithm for transmitting images over LoraWAN (Algorithm 1).

**Algorithm 1** Image transmission algorithm

```
if Cs ≤ 0.9 then
    Compress image to Webp format
    Convert image in Webp format to hexadecimal string Determine data rate for transmission
    Split the file into several packet N according to data rate Group packet together by a number of 10
    Add an application specific CRC code for each group. Establish connection with gateway
    while PacketSent ≤ N do
        Send packet by group
        Server check the integrity of the received data
        if CRC does not match then
            Server asks for group Re-transmission
            Device re-transmit the group
        else
            Servers sends acknowledgment of reception
        end if
    end while
    Final error verification check
    Server acknowledge the reception
end if
```

The procedure begins by compressing the image to the WebP format, which is identified as more efficient than JPEG or JPEG2000 in the existing literature [20]. The file is then prepared for transfer by converting it to a hexadecimal string. Once the data rate is selected, we decide the largest payload size per packet and then split our file accordingly. We group the packets in sets of ten to minimize network congestion, as each packet typically receives an acknowledgement in LoRaWAN. To prevent any loss of information during this grouping, we add an extra Cyclic Redundancy Code to each group, allowing the application server to verify the successful receipt of each one. The transmission process then starts, and each group of packets is acknowledged by the server. If a group is not received, it is re-sent. Once all the groups have been retrieved, the server conducts a final verification of the overall received file and acknowledges the device.

#### 3.2.2. Server Model Re-Training

Once a certain amount of new image data has been collected, the server can re-train a new model. In our application, we obtained a model with sufficient accuracy using 100 images.

Depending on the specific task a smart sensor must perform, this number may be lower or higher. Images can be collected from one device or multiple devices monitoring the same phenomenon under the same conditions. Linking devices with similar characteristics can be challenging due to variations in hardware, applications, and the environments in which they are deployed, among other factors.

The model re-training is quite straightforward and occurs in the same manner as in the normal behavior mode.

#### 3.2.3. Firmware Update over the Air

After obtaining the new model, the last step in the process is to update the concerned devices. Once more, the server performs device grouping through semantic capabilities. When the server is prepared to update, it starts the FUOTA procedure described in Figure 10.

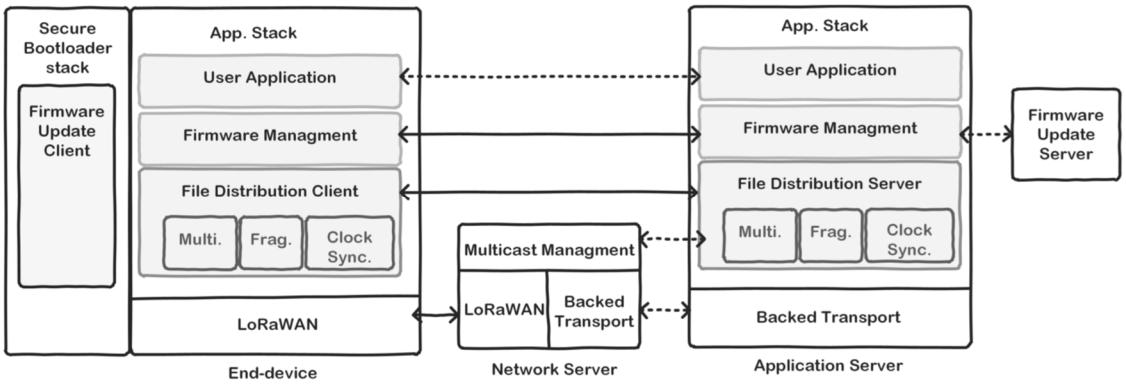


**Figure 10.** FUOTA procedure.

## 4. Experimentation and Results

### *4.1. Normal Mode*

#### 4.1.1. TinyML Model Creation Process with Edge Impulse

For our experiment, we trained our model following the previously described phases to identify strawberries, using a collection of 100 pictures, each containing 0 to 10 instances of the fruit. We trained two distinct models: one using images from the Arduino Portenta Shield and its HM-01B0 monochrome camera, and another using images from the ESP32 and its integrated OV2640 camera.

Since the STM32 also uses the OV2640 camera, we will employ the same TinyML model for it. The Arduino's camera, being black and white, produces images of lower

resolution but of smaller size, which can be helpful for image offloading. The type of images does not appear to influence the model size, as both library files retrieved from Edge Impulse after training are approximately the same size (242 kB for Arduino and 253 kB for ESP). Regarding model performance, the model for the Arduino achieved a mean accuracy of 90.2%, while the model for the ESP32/STM32 achieved a mean accuracy of 92.3%.

This corresponds to steps 1, 2, and 3 of the end-to-end process.

#### 4.1.2. Sensor Deployment (Hardware Deployment Results)

After training both TinyML models (one on Arduino Portenta et one on the ESP32/STM32), they are ported to the various devices and tested multiple times to verify the system's robust behavior and obtain key metrics. We constructed a small wooden structure to hold the sensors and evaluated our architecture with varying numbers of strawberries, see Figure 11.


**Figure 11.** Test bench featuring the ESP32-Cam on the left, the Arduino Portenta in the middle, and the STM32 on the right.

The technical characteristics of each device, with respect to RAM and flash memory capacity, are presented in Table 1.

**Table 1.** Hardware characteristics.

| | ESP32 | Arduino Portenta | STM32 |
|---|---|---|---|
| RAM | 512 kb | 1 Mb | 100 kb |
| Operating frequency | 8 MHz | 48 MHz | 32 MHz |
| Flash memory | 448 Kb | 2 Mb | 1 Mb |

We can observe from this hardware deployment testing that the models are performing as expected, confirming the efficiency of TinyML in agricultural scenarios. However, we also highlight the importance of hardware selection when developing an embedded AI application (Table 2). In this specific scenario, some hardware, like the Portenta, appears somewhat oversized as most of its resources are unused, while, conversely, the STM32 is underperforming due to RAM capacity. On the other hand, the ESP32 seems to be the right size.

**Table 2.** Hardware deployment results.

| | ESP32 | Arduino Portenta | STM32 |
|---|---|---|---|
| Firmware memory usage | 18.42% | 3.88% | 9.32% |
| RAM usage | 88.87% | 24.39% | 100% |
| Inference time | 312 ms | 148 ms | 676 ms |
| Accuracy respected? | Yes | Yes | Yes |

#### 4.1.3. LoraWAN Transmission (Transmission)

As soon as the TinyML model finished running, the number of fruits inferred is transmitted to the server through LoRaWAN; the message will be transferred in the form

of a single packet. In our experimental setup, the Gateway and the node were in the same room, approximately 10 m apart, with a clear line of sight. We collected the average time of the transmission process over 10 transmissions for the typical data rates and devices. Additionally, we calculated the maximum number of transmissions per day according to the duty cycle. The data rate (DR) in LoRaWAN is a crucial factor. It is the number of bits transmitted per unit of time. In LoRaWAN modulation, the data rate is influenced by several factors, including the spreading factor (SF), bandwidth (BW), and coding rate (CR). It is expressed by the following formula:

$$DR = \frac{BW \times SF \times CR}{2^{SF}}$$

So, we consider data rate DR0 to DR6 (Table 3). We have seen that the transmission time appears to be more dependent on the data rate than on the device (Table 4). This can be explained using the same transmission chip hardware, the SX1276, in each microcontroller. Finally, we also conclude that, overall, the transmission duration is quite short, minimizing power consumption and adhering to duty cycle regulations (a maximum of 864 s a day for transmission). This allows devices to communicate their results between 947 and 65,954 times a day.

**Table 3.** Bit rate for each data rate (DR0—DR6) configured with the spreading factor and the bandwidth.

| Data Rate | Configuration (SF + BW) | Bit Rate (bit/s) |
|---|---|---|
| 0 | LoRaWAN: SF12/125 kHz | 250 |
| 1 | LoRaWAN: SF11/125 kHz | 440 |
| 2 | LoRaWAN: SF10/125 kHz | 980 |
| 3 | LoRaWAN: SF9/125 kHz | 1760 |
| 4 | LoRaWAN: SF8/125 kHz | 3125 |
| 5 | LoRaWAN: SF7/125 kHz | 5470 |
| 6 | LoRaWAN: SF7/250 kHz | 11,000 |

**Table 4.** Transmission time in milliseconds of inference result for various data rates and maximal number of inferences per day.

| | DR0 | DR1 | DR2 | DR3 | DR4 | DR5 | DR6 |
|---|---|---|---|---|---|---|---|
| ESP32 | 912.4 | 458.1 | 241.6 | 117.2 | 61.2 | 28.8 | 15.2 |
| Max inference per day | 947 | 1886 | 3576 | 7372 | 14,118 | 30,000 | 56,842 |
| Arduino Portenta | 854.7 | 421.3 | 222.3 | 115.7 | 54.5 | 29.2 | 13.1 |
| Max inference per day | 1011 | 2051 | 3887 | 7468 | 15,853 | 29,589 | 65,954 |
| STM32 | 887.5 | 451.9 | 252.2 | 122.8 | 57.6 | 29.6 | 14.7 |
| Max Inference per day | 974 | 1912 | 3426 | 7036 | 15,000 | 29,189 | 58,776 |

Energy consumption: the WSN network used here is LoraWAN. It has three modes, sleep, run, and transmission mode. The average energy consumption of each device in various operational states is provided in the manufacturers' datasheets and is presented in Table 5. We chose to use the theoretical values for energy consumption, as experimental readings yielded similar average results.

**Table 5.** Power consumption for each device across various states.

| Mode | ESP32 | Arduino Portenta | STM32 |
|---|---|---|---|
| Sleep | 10 µA | 2.95 µA | 0.29 µA |
| Run | 190 mA | 121 mA | 105 mA |
| Transmission LoRa | 211.5 mA | 142.5 mA | 126.5 mA |

We can calculate the daily average energy consumption for each device by multiplying the time spent in each mode (specifically, 'run' during the inference process, 'transmission' during the transmission process, and 'sleep' during the sleep process) by the respective energy consumption of that mode and adding them together. The formula is as follows:

$$Edaily = Erun \times Trun + Etransmission \times Ttransmission + Esleep \times Tsleep$$

Using this daily energy consumption data, we can calculate the average battery life of a device using a 2000 mA battery, a standard battery size for IoT devices. These results are presented in Figure 12.

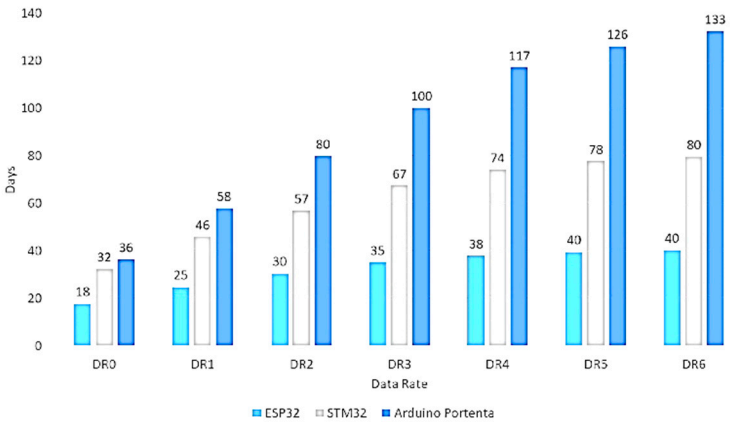


**Figure 12.** Battery lifetime (in days) for various data rates; based on experimental results and theoretical values.

With our experimental system, we illustrate that a 2000 mA battery can sustain operation for between 18 and 40 days for the ESP32, 32 and 80 days for the STM32, and 36 and 133 days for the Arduino Portenta. Table 6 illustrates the total energy requirement in mA for one year of device operation.

**Table 6.** Total energy requirement in mA for one year of device operation.

| | DR0 | DR1 | DR2 | DR3 | DR4 | DR5 | DR6 |
|---|---|---|---|---|---|---|---|
| ESP32 | 41,203 | 29,512 | 23,941 | 20,740 | 19,299 | 18,465 | 18,115 |
| Arduino | 20,098 | 12,584 | 9134 | 7286 | 6225 | 5786 | 5507 |
| STM32 | 22,605 | 15,901 | 12,828 | 10,836 | 9833 | 9402 | 9172 |

#### 4.1.4. Result Interpretation (Knowledge Base)

Once the transmission phase is completed, this enables our system to make informed decisions based on the sensor network. These decisions may pertain to farm management actions or sensor network maintenance operations. An effective tool for managing a wide range of data and agents, as is the case in agricultural IoT, is the creation of an expert system capable of making decisions with the help of a knowledge base. We will show how the ontology is employed by a reasoning algorithm to help informed decision-making within

our system. Our approach integrates elements from the Semantic Sensor Network [21] to describe the WSN and draws on the Plant Ontology available on Planteome [22] for plant descriptions. Additionally, we reference the work of authors in [23] to model interactions between pests and diseases. Furthermore, we will include a rule layer to define the interactions among these classes, generating valuable insights for managing farms and sensor networks.

For the knowledge base of our IoT platform related to agroecology, we have chosen to propose three main classes: WSN (wireless sensor network), Crops, and Farm. The overall model is presented in Figure 13. We used Protegé software release 5.6.5 to construct our ontology. We evaluated our ontology with O'FAIRe [24], an Ontology FAIRness evaluator, and obtained a fair score based on various criteria. The results are presented in Figure 14. The assessment of fairness for the ontology is based on various rules outlined by the authors [25].


**Figure 13.** Knowledge base for result interpretation.

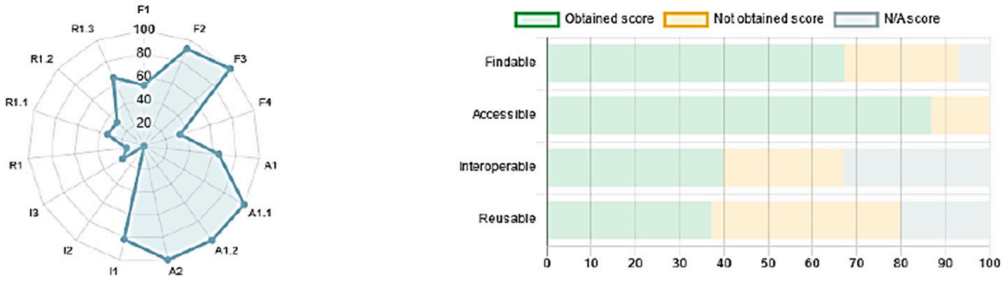


**Figure 14.** Ontology fairness evaluator results.

### *4.2. Degraded Mode*

#### 4.2.1. Image Data Transmission (Heavy Data Offload)

The architecture we propose that supports model update for local application is, to our knowledge, novel. Consequently, we found no literature addressing this specific use case of heavy data offloading for model re-training. However, studies exploring the transfer of substantial data, such as images or audio files, using the LoRaWAN network are available. In this section, we will examine them to better define our strategy for implementing this functionality in our system. Firstly, we will conduct a theoretical calculation to understand the number of pictures that can be sent daily based on the picture size. Various data rate values exist, figured out by network availability, which in turn decide the speed (S) of transmission in bits per second (Bps). A lower data rate increases the probability of message transmission at the expense of bit rate.

We are also aware of the duty cycle (DC)—the time a device can send data daily as per regulations. Using this information, along with the size of a media (MS), we can decide the maximum number (N) of times a media can be transmitted in a day using the following formula:

$$N = \frac{S \times DC}{MS}$$

We implemented algorithm1 into only the Arduino Portenta and ESP32 microcontrollers. We are concentrating on these two microcontrollers at this time because the main complexity of our algorithm arises from the transmission side and the size of the image taken, while the STM32 utilize the same transmission chip and camera in their hardware configuration.

The images taken from the Portenta and ESP32 can vary in size with each measurement. The average size of images taken by the Arduino Portenta and compressed with WebP over ten images is 4.7 kb. For the ESP32, the average size of images, which are in color this time, is higher and reaches an average of 9.8 kb. The average transmission time for each image at different data rates is presented in Table 7. We kept the same setup as previously, with only one node at a time, separated by 10 m from the gateway, and with a clear line of sight.

**Table 7.** Average image transmission time (s) and maximum daily transfers at various data rates, according to duty cycle.

| Device | DR0 | DR1 | DR2 | DR3 | DR4 | DR5 | DR6 |
|---|---|---|---|---|---|---|---|
| ESP32 | 117.60 | 66.82 | 30.00 | 16.70 | 9.41 | 5.37 | 2.67 |
| Max image transfer per day | 7 | 13 | 29 | 52 | 92 | 161 | 323 |
| Arduino Portenta | 56.40 | 32.05 | 14.39 | 8.01 | 4.51 | 2.58 | 1.28 |
| Max image transfer per day | 15 | 27 | 60 | 108 | 191 | 335 | 674 |

From these results, we highlight the efficiency of our algorithm for sending images of varied sizes, as it outperformed the other available methods. We also obtained a result like the theoretical one regarding the duty cycle limitation in Table 8.

**Table 8.** Number of image offload operations possible with a 2000 mA battery, according to data rates and quantity of images sent in a single operation.

| Device | Image Quantity | DR0 | DR1 | DR2 | DR3 | DR4 | DR5 | DR6 |
|---|---|---|---|---|---|---|---|---|
| ESP32 | 10 | 29 | 51 | 113 | 204 | 362 | 633 | 1274 |
| | 50 | 6 | 10 | 23 | 41 | 72 | 127 | 255 |
| | 100 | 3 | 5 | 11 | 20 | 36 | 63 | 127 |
| Arduino | 10 | 90 | 158 | 351 | 631 | 1120 | 1960 | 3942 |
| | 50 | 18 | 32 | 70 | 126 | 224 | 392 | 788 |
| | 100 | 9 | 16 | 35 | 63 | 112 | 196 | 394 |

#### 4.2.2. Server Model Re-Training

Knowing that we only needed 100 pictures to train our TinyML model, we observe that the Arduino Portenta can re-train a new model in one day when communicating at data rates between DR3 and DR6. For the ESP32, re-training is possible at data rates between DR5 and DR6.

However, devices, especially in agricultural scenarios such as ours, are usually not deployed in isolation, and multiple others are seeing the same phenomenon. Therefore, data could be collected from multiple nodes, making it possible to transfer enough images for model re-training, even at the lowest data rate. For example, ten devices equipped with the Arduino Portenta could each transfer ten images at the lowest data rate, allowing a model to be re-trained even in degraded network situations. A comparison between Arduino protenta and STM32 is represented in Figures 15 and 16. Separating the data offload could also be beneficial for battery lifetime. Instead of having one device transfer 100 images, it would be more optimized to share this task among ten devices to maintain a balance in the battery level, avoiding one device running out of power more quickly than the others.

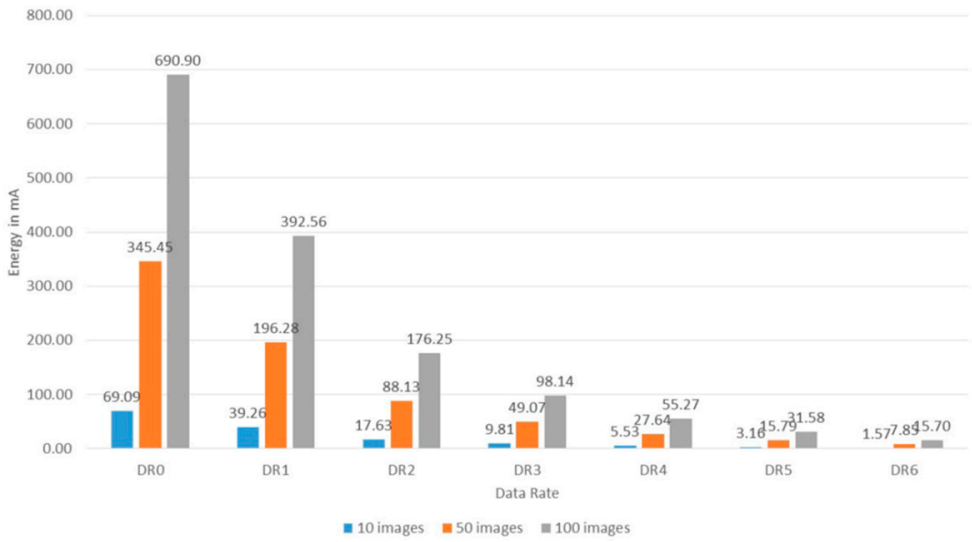


**Figure 15.** Energy consumption of the system for transferring various numbers of images using the ESP 32.

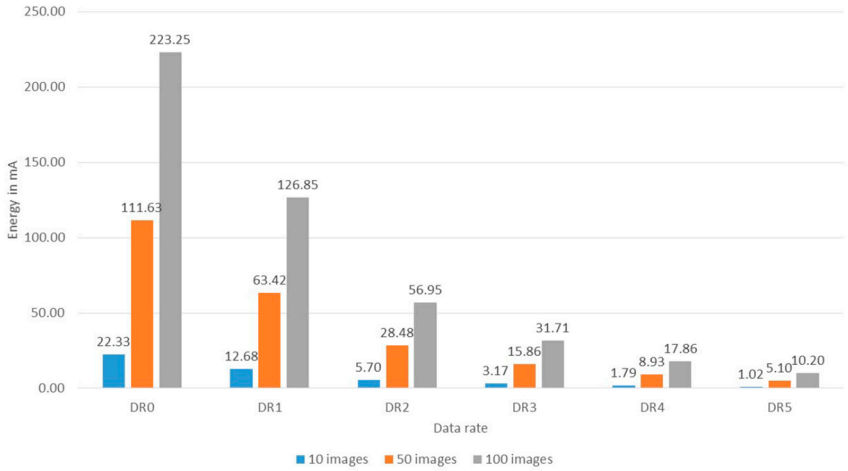


**Figure 16.** Energy consumption of the system for transferring various numbers of images using the Arduino protenta.

#### 4.2.3. FUOTA

In our final experimentation, we focused on the firare update over the air procedure (FUOTA). We implemented the protocol described below using the open-source LoRaWAN server, ChirpStack [26]. We adapted the FUOTA server proposed by the authors in [26] to run on our Laird Gateway. The Arduino Portenta H7 operates Arm Mbed OS in charge of the reception of the update. First, we trained a new example model with new data and compiled it. The size of the updated model is 83.6 kb. The FUOTA process is long as the firmware size is large. We compiled the average time measured for 10 FUOTA process for a node device (reception and update) with the same firmware according to different data rates (DRs) in Table 9.

**Table 9.** FUOTA process duration in seconds (s).

| | DR0 | DR1 | DR2 | DR3 | DR4 | DR5 | DR6 |
|---|---|---|---|---|---|---|---|
| Arduino Portenta | 552 | 475 | 412 | 302 | 245 | 176 | 153 |
| Duty Cycle limitation | 1 | 1 | 2 | 2 | 3 | 4 | 5 |

From these first results, we conclude that updating the TinyML model is possible with LoRaWAN. As the FUOTA process is punctual, we can also assert that the duty cycle is respected, especially since it is the gateway that uses its duty cycle for transmission, with the node device only sending acknowledgment messages. Therefore, it is also of utmost importance for the gateway to correctly group the devices that need to receive the same update, as performed in the multicast addressing procedure of the FUOTA recommendation. If multiple devices need different updates, the system should carefully schedule those FUOTA processes to avoid overflowing its duty cycle, especially with lower data rates.

To illustrate energy consumption, we analyzed two scenarios, both utilizing a DR6 data rate. In each scenario, the device predominantly remains in standby mode, waking up once daily to execute a single TinyML inference and send a brief telemetry message to the server. Additionally, the device activates once a week for firmware updates, which are 100 kb in size. The first scenario employs LoRaWAN for all communications, while the second scenario uses WiFi for comparison. For our experiment, we implemented the basic energy evaluation model from the INET framework of Omnet++ [27]. The simulator assesses the battery's expected lifetime and operates until both scenarios deplete a 2000 mA battery. Results are illustrated in Figure 17, indicating that leveraging LoRaWAN in this application could lead to significant energy savings compared to WiFi usage.

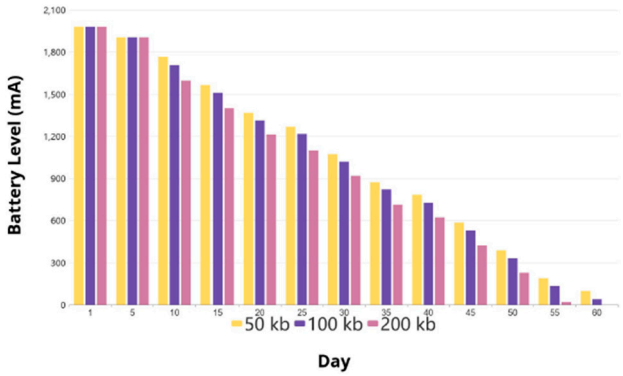


**Figure 17.** Evolution of battery level over time for various size of firmware.

Additionally, we suggest assessing the battery life for different sizes of firmware updates, using the same parameters as before (Arduino Portenta only and DR6). The results are illustrated in Figure 18. As anticipated, larger update sizes adversely affect energy consumption. The overall finding regarding the FUOTA process indicates that it is suitable for a TinyML model and is energy efficient. However, the frequency of the FUOTA process may vary depending on the specific application. Therefore, careful consideration of battery sizing is necessary to ensure optimal device performance.

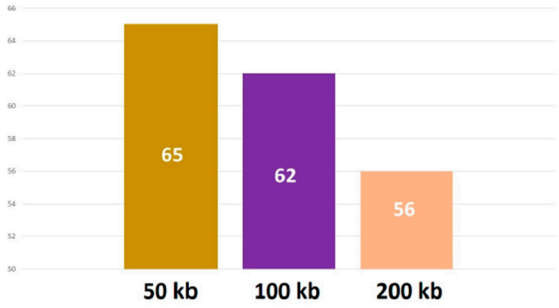


**Figure 18.** Battery lifetime of the system in days for various sizes of firmware.

## 5. Conclusions

In this paper, we proposed an intelligent IoT platform for smart agriculture that integrates various technologies. Initially, we utilized TinyML and LoRaWAN to develop an energy-efficient model for fruit detection, proving the potential of these technologies in agriculture. Our experiments indicated that this model achieved an accuracy of 90% and was three times more energy-efficient than a cloud-based alternative for the same application, paving the way for a new array of computer vision applications in smart farming using battery-powered sensors. However, despite these promising results, the TinyML paradigm has limitations on device learning capabilities.

To achieve this, we proposed an end-to-end platform that integrates several novel contributions to support the TinyML. First, we demonstrated the energy efficiency of embedded AI systems in processing large data sets, such as images. Next, we developed a protocol that facilitates remote updates for intelligent sensors over LoRaWAN, overcoming the challenges posed by large firmware sizes. Additionally, we introduced an energy-efficient offloading strategy for managing agroecological IoT platforms.

Our findings underscore the potential of embedded AI in agriculture, tackling networking challenges within LoRaWAN networks, and highlighting the effectiveness of knowledge-based systems in complex areas like agroecology and sensor management.

Moreover, we proposed a method for handling large sensor data by initiating the re-training of AI models in the cloud, called firmware updates over the air (FUOTA), by updating the TinyML model once the sensors are deployed to enhance accuracy post-acquisition whenever a sensor's confidence score drops below a specified threshold. This

approach allows models to utilize local data for re-training and subsequent remote updates. We also created an ontology-based knowledge base to represent IoT devices, their interactions with plants, and agroecological principles, thereby establishing a comprehensive framework for agricultural IoT.

Overall, our results highlight the future potential of embedded AI in agriculture and provide solutions to address networking challenges within the LoRaWAN network. We tested the proposal under various hardware and network conditions, demonstrating the effectiveness of knowledge-based systems for complex application areas like agroecology and sensor management.

**Author Contributions:** Conceptualization, N.B. and N.C.; Methodology, A.R.-C. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** https://github.com/NicolasChollet51/Portenta_Fragola (accessed on 10 November 2024); https://github.com/NicolasChollet51/LoRaWAN-FUOTA-TinyML (accessed on 10 November 2024).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Foley, J.A.; Ramankutty, N.; Brauman, K.A.; Cassidy, E.S.; Gerber, J.S.; Johnston, M.; Mueller, N.D.; O'Connell, C.; Ray, D.K.; West, P.C.; et al. Solutions for a cultivated planet. *Nature* **2011**, *478*, 337–342. [CrossRef] [PubMed]
2. Zhang, L.; Yan, C.; Guo, Q.; Zhang, J.; Ruiz-Menjivar, J. The impact of agricultural chemical inputs on environment: Global evidence from informetrics analysis and visualization. *Int. J. Low-Carbon Technol.* **2018**, *13*, 338–352. [CrossRef]
3. Hardy, J.T. *Climate Change: Causes, Effects, and Solutions*; John Wiley & Sons: Hoboken, NJ, USA, 2003.
4. Abadade, Y.; Temouden, A.; Bamoumen, H.; Benamar, N.; Chtouki, Y.; Hafid, A.S. A comprehensive survey on tinyml. *IEEE Access* **2023**, *11*, 96892–96922. [CrossRef]
5. Zapico, J.L.; Ahlgren, F.; Zennaro, M. Insect biodiversity in agriculture using IoT: Opportunities and needs for further research. In Proceedings of the 2021 IEEE Globecom Workshops (GC Wkshps), Madrid, Spain, 7–11 December 2021.
6. Ray, P.P. A review on TinyML: State-of the-art and prospects. *J. King Saud Univ. Inf. Sci.* **2021**, *34*, 1595–1623. [CrossRef]
7. Roshan, A.N.; Gokulapriyan, B.; Siddarth, C.; Kokil, P. Adaptive Traffic Control With TinyML. In Proceedings of the 2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, India, 25–27 March 2021.
8. Olsson, S.; Efforts, T. TinyML and IoT for Conservation Efforts. Hackster.io. 2022. Available online: https://www.hackster.io/saraolsson4s/tinyml-andiot-for-conservation-efforts-dd34db (accessed on 10 November 2024).
9. Aslanpour, M.S.; Gill, S.S.; Toosi, A.N. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet Things* **2020**, *12*, 100273. [CrossRef]
10. Sheth, D.; Sudharsan, B.; Breslin, J.G.; Ali, M.I. Embedded ml pipeline for precision agriculture. In Proceedings of the 2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Milano, Italy, 4–6 May 2022; pp. 527–528.
11. Mohammed, S.W.; Soora, N.R.; Polala, N.; Saman, S. Smart water resource management by analyzing the soil structure and moisture using deep learning. In *IOT with Smart Systems*; Choudrie, J., Mahalle, P., Perumal, T., Joshi, A., Eds.; Springer Nature: Singapore, 2023; pp. 709–719.
12. Ramos-Giraldo, P.; Reberg-Horton, S.C.; Mirsky, S.; Lobaton, E.; Locke, A.M.; Henriquez, E.; Zuniga, A.; Minin, A. Low-cost smart camera system for water stress detection in crops. In Proceedings of the 2020 IEEE SENSORS, Rotterdam, The Netherlands, 25–28 October 2020; pp. 1–4.
13. Matilla, D.M.; Murciego, A.L.; Jiménez-Bravo, D.M.; Mendes, A.S.; Leithardt, V.R. Low-cost edge computing devices and novel user interfaces for monitoring pivot irrigation systems based on internet of things and lorawan technologies. *Biosyst. Eng.* **2022**, *223*, 14–29. [CrossRef]
14. Mummigatti, K.V.; Chandramouli, S.M.; Ramachandra, D.H. Deep Neural Network System Using Ontology to Recommend Organic Fertilizers for a Sustainable Agriculture. *Ingénierie Des Systèmes D'information* **2023**, *28*, 461. [CrossRef]
15. Impulse, E. FOMO: Object Detection for Constrained Devices. 2021. Available online: https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices (accessed on 10 November 2024).
16. Available online: https://docs.arduino.cc/resources/datasheets/ASX00026-datasheet.pdf (accessed on 10 November 2024).
17. Available online: https://www.espressif.com/en/news/ESP32_CAM (accessed on 10 November 2024).
18. Available online: https://www.st.com/en/evaluation-tools/p-nucleolrwan2.html#overview (accessed on 10 November 2024).

19. Aadithya, V.; Balakumar, S.; Bavishprasath, M.; Raghul, M.; Malathi, P. Comparative Study Between MobilNet FaceMask Detector and YOLOv3 Face-Mask Detector. In *Sustainable Communication Networks and Application: Proceedings of ICSCN 2021*; Springer: Singapore, 2022; pp. 801–809.
20. Wei, C.C.; Su, P.Y.; Chen, S.T. Comparison of the LoRa image transmission efficiency based on different encoding methods. *Int. J. Inf. Electron. Eng.* **2020**, *10*, 1–4. [CrossRef]
21. Available online: https://www.w3.org/TR/vocab-ssn/ (accessed on 10 November 2024).
22. Cooper, L.; Meier, A.; Laporte, M.A.; Elser, J.L.; Mungall, C.; Sinn, B.T.; Cavaliere, D.; Carbon, S.; Dunn, N.A.; Smith, B.; et al. The Planteome database: An integrated resource for reference ontologies, plant genomics and phenomics. *Nucleic Acids Res.* **2018**, *46*, D1168–D1180. [CrossRef] [PubMed]
23. Rodríguez-García, M.Á.; García-Sánchez, F. CropPestO: An Ontology Model for Identifying and Managing Plant Pests and Diseases. In *International Conference on Technologies and Innovation*; Springer: Cham, Switzerland, 2020; pp. 18–29.
24. Agroportal. Agroportal/Fairness: This Project Is a Fairness Assessment Tool for Ontologies, Vocabularies and Semantic Resources. Available online: https://github.com/agroportal/fairness#-ofaire-ontologyfairness-evaluator (accessed on 10 November 2024).
25. Hu, S.; Wang, H.; She, C.; Wang, J. AgOnt: Ontology for agriculture internet of things. In Proceedings of the Computer and Computing Technologies in Agriculture IV: 4th IFIP TC 12 Conference, CCTA 2010, Nanchang, China, 22–25 October 2010; Selected Papers, Part I 4. Springer: Berlin/Heidelberg, Germany, 2011; pp. 131–137.
26. Available online: https://github.com/chirpstack/chirpstack-fuota-server (accessed on 10 November 2024).
27. Available online: https://omnetpp.org/ (accessed on 10 November 2024).