



Article

Lightweight Digit Recognition in Smart Metering System Using Narrowband Internet of Things and Federated Learning

Vladimir Nikić , Dušan Bortnik , Milan Lukić , Dejan Vukobratović  and Ivan Mezei *

Faculty of Technical Sciences, University of Novi Sad, 21000 Novi Sad, Serbia; niki.vladimir@uns.ac.rs (V.N.); bortnik@uns.ac.rs (D.B.); milan_lukic@uns.ac.rs (M.L.); dejanv@uns.ac.rs (D.V.)

* Correspondence: imezei@uns.ac.rs

Abstract: Replacing mechanical utility meters with digital ones is crucial due to the numerous benefits they offer, including increased time resolution in measuring consumption, remote monitoring capabilities for operational efficiency, real-time data for informed decision-making, support for time-of-use billing, and integration with smart grids, leading to enhanced customer service, reduced energy waste, and progress towards environmental sustainability goals. However, the cost associated with replacing mechanical meters with their digital counterparts is a key factor contributing to the relatively slow roll-out of such devices. In this paper, we present a low-cost and power-efficient solution for retrofitting the existing metering infrastructure, based on state-of-the-art communication and artificial intelligence technologies. The edge device we developed contains a camera for capturing images of a dial meter, a 32-bit microcontroller capable of running the digit recognition algorithm, and an NB-IoT module with (E)GPRS fallback, which enables nearly ubiquitous connectivity even in difficult radio conditions. Our digit recognition methodology, based on the on-device training and inference, augmented with federated learning, achieves a high level of accuracy (97.01%) while minimizing the energy consumption and associated communication overhead (87 μ Wh per day on average).

Keywords: NB-IoT; machine learning; smart metering; lightweight digit recognition; federated learning



Citation: Nikić, V.; Bortnik, D.; Lukić, M.; Vukobratović, D.; Mezei, I. Lightweight Digit Recognition in Smart Metering System Using Narrowband Internet of Things and Federated Learning. *Future Internet* **2024**, *16*, 402. <https://doi.org/10.3390/fi16110402>

Academic Editor: Paolo Bellavista

Received: 4 October 2024

Revised: 28 October 2024

Accepted: 29 October 2024

Published: 31 October 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Measuring household consumption (e.g., water, gas, etc.) is currently based on two types of devices. The first type consists of various traditional mechanical meters (TMs). TMs are dial meters and do not provide any smart metering or telemetry functions. Employing this type of metering device requires a lot of expensive manual labor for manual readings, often disturbing privacy. On the other hand, smart meters (SM) utilize recent technologies to measure consumption more efficiently. They usually feature wireless readings and low-power operation necessary for battery-powered devices.

To ensure stable wireless connectivity for metering devices placed in challenging environments such as basements, shafts, holes, and closed objects, many power utilities turn their attention to Narrowband IoT (NB-IoT) technology as a suitable low-cost solution. NB-IoT is an addition to the 4G LTE standard that allows up to 50,000 low-power devices (e.g., SMs) to connect to a single base station. NB-IoT will enable massive cellular IoT and is a game-changer for the mobile industry and operators, who aim to use NB-IoT to provide seamless connectivity for billions of new devices. The key features in favor of the utilization of NB-IoT in the SM scenario are as follows [1]:

- Ultra low-power operation—When a small payload is sent a couple of times daily, power saving mode (PSM) allows for a battery lifetime of up to 10 years.
- Deep indoor penetration—Coverage extension modes allow for up to 20 dB gain compared to the GSM employing repeated transmissions.

- High-Density Connectivity Solution—The ability to connect a large number of devices without network congestion.
- Efficient Low-Data Transfer—Technology is optimized for small, infrequent data transmissions.

The primary advantages of utilizing NB-IoT cellular technology stem from its reliance on existing base station infrastructure, significantly reducing the labor demands associated with system setup and data acquisition. Additionally, NB-IoT’s PSM is pivotal for extending battery lifespan, allowing devices deployed in remote or hard-to-reach locations to function for prolonged periods. This capability minimizes the need for maintenance interventions, particularly regarding battery replacement, thereby enhancing operational efficiency and reducing long-term upkeep costs.

Additionally, state-of-the-art modules provide several fallback options in case the NB-IoT technology is not supported in the corresponding region, including GSM and long-term evolution machine type communication (LTE-M). Consequently, these modules will consistently remain operational whenever a mobile network signal is accessible, which is the most widely available connectivity option.

This paper presents an innovative new solution that “smartifies” the TMs in a retrofitting manner [2], as an alternative to the existing energy-inefficient (e.g. GSM/GPRS) products or non-telemetry-based products. Our energy-efficient device using wireless NB-IoT technology transforms the TM into an SM by applying an innovative, low-cost upgrade of the existing TM (no need to replace TM). It consists of an embedded low-cost camera and a smart number recognition system. The SM architecture is depicted in Figure 1.

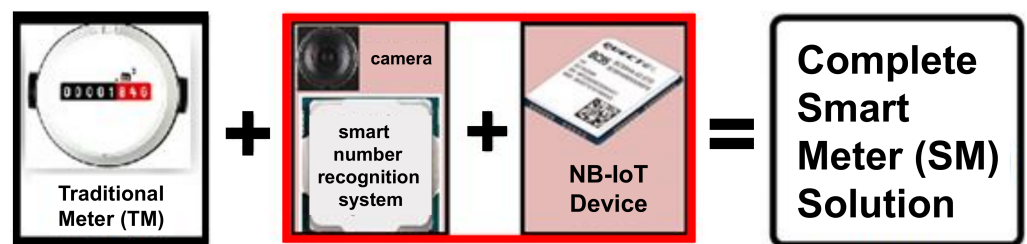


Figure 1. Proposed SM architecture used for old TM retrofitting.

The system architecture (Figure 2), which is described in more detail in Section 3, has been designed to provide end-to-end connectivity between SMs and cloud infrastructure using NB-IoT provided by a mobile network operator (MNO) and cloud services. The end-users can retrieve the data using RESTful-based interfaces and other application-layer protocols such as MQTT or CoAP.

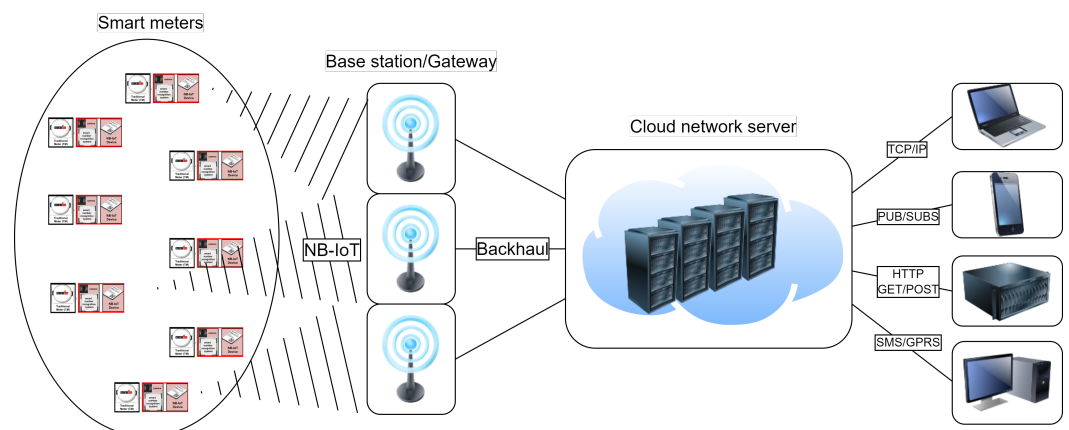


Figure 2. The system architecture consists of a collection of deployed SMs that communicate via MNO with the cloud.

The “smartification” of the TMs is based on the addition of a machine learning (ML) algorithm. Based on deployment, the ML models can be divided into three groups: cloud-based, mobile-based, and tiny. While cloud and mobile solutions feature high frequency (GHz), high memory capacity (GBs), and high storage (G/T/PBs), they also have high power consumption (W/kW) and a significant impact on the environment [3]. On the other side, there are tiny, resource-restricted devices with small ML models (e.g., TinyML [4,5]) featuring much lower operating frequencies (MHz), memory capacity (KBs) [6], storage (MBs), and power consumption (mW), being more environmental-friendly. Additionally, such a small power consumption enables battery-powered tiny devices to foster the long-term viability of such a solution.

In our case, the appropriate ML algorithm would enable retrofitting devices to detect numbers displayed on the dial metering devices. Numbers and digit recognition are well-known problems in computer science usually solved by neural networks. Digit recognition tasks feature lower complexity than number recognition. Recurrent neural networks (RNNs) are often used for image-based sequence recognition tasks. However, since the RNNs are computationally more expensive in both training and inference, compared to convolutional neural networks (CNNs), their application in resource-constrained systems is limited [7]. As there are many different types of TMs, with various colors, sizes, and fonts, it would be worth having a specifically trained model for each distinct device. In order not to overwhelm the server, it would be beneficial to have CNN model training on the device itself. As devices used in embedded environments are resource-constrained, CNN models, trained and executed on them, should be minimal.

Finally, to spread the “knowledge” across all devices and increase their accuracy in digit detection, we utilize federated learning (FL). FL is a novel ML concept particularly suitable for applications in massive IoT employing low-capability nodes [8]. The power of FL emerges from the massive engagement of nodes. Nodes are required to share a local model update on the cloud. Their models are fused at the central cloud location and distributed back to all nodes. Depending on the methods used, the performance of individual nodes could be enhanced (Figure 3). Recently, there has been a lot of research conducted on the intersection of the IoT and FL, as reported in several systematic reviews [9–12]. In metering scenarios, FL is primarily used for consumption forecasting [13]. In our study, we employed an averaging methodology grounded in the principle of weight averaging across models. Furthermore, we adhered to the recommendations outlined in [14,15], which suggest that the batch normalization layer in CNN models should remain unchanged during the averaging process of the other layers for each device.

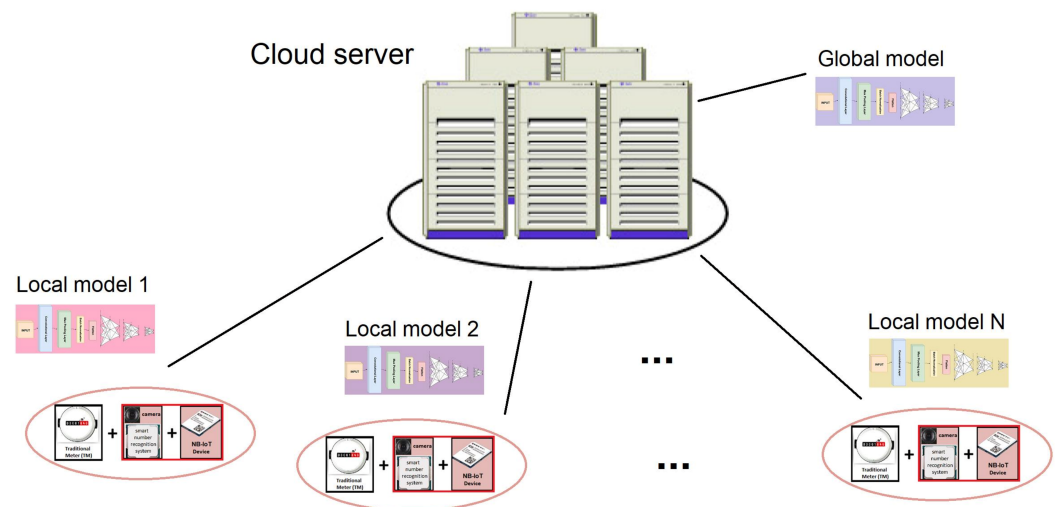


Figure 3. Distribution of ML models on edge devices, which provides the basis for FL.

The main contributions of this work are the following. We present the design, implementation, and analysis of

- a new SM design for the retrofitting of TMs (a fabricated prototype);
- the custom CNN optimized for the usage with resource-constrained hardware;
- CNN model training on the edge device;
- a custom pre-processing of input images to reduce complexity;
- integration of FL in the proposed solution, which enhances the generalization of our digit recognition models, thus improving overall accuracy;
- detailed energy consumption of the proposed system.

The organization of this paper is structured as follows: Section 2 provides a comprehensive analysis of the related work in this field. In Section 3, we present a detailed description of the system architecture, including the design of the edge device, the overall digit recognition scheme, and our experimental setup. Section 4 covers the methodology used—digit image preprocessing, the proposed model architecture, and our approach to FL. Section 5 analyzes the results from the two evaluated test cases and compares our solution with the existing approaches. Here, we also elaborate on the long-term viability of the solution based on power consumption analysis and compare our solution to others. Finally, Section 6 concludes the paper with a summary and suggestions for future research.

2. Related Work

We are interested in studies related to retrofitting fabricated digit-recognition-based SM prototypes optimized for edge, based on FL. Accordingly, the following section is organized into subsections presenting the related work in the relevant areas.

2.1. Low Power Communications

The current state-of-the-art in the field of SMs is such that an increasing number of solutions are utilizing NB-IoT (e.g., Shenzhen gas company in China). An example of an early NB-IoT water metering system is given in [16]. Several commercial case studies in China are reviewed in [17]. Such solutions in Europe were in the mature pilot testing phase a few years ago (e.g., Kamstrup from Denmark [18]).

Other solutions are based on LoRaWAN technologies [19]. Although the design seems to be low power and low cost (STM8 microcontroller unit), there are no data to support such a claim in the paper. Additionally, it can be used only on pulse output water meters. A hybrid water meter solution based on infrared and LoRaWAN is given in [20].

2.2. Dial Meters

Currently, many efforts are being made to retrofit water meters [21]. Similar solutions, however, could be used for other household resources that are measured. In [22], authors presented an analog water meter retrofitted to a SM. It is designed using Raspberry Pi, which is not low-power. Another Raspberry Pi-based solution using LoRa and wireless M-Bus is given in [23]. In [24], an interesting NB-IoT-based retrofitted water flow meter is presented with an energy harvesting circuit. However, the measurements are not based on digit recognition but are acquired by an induction emitter that detects metal rotating on a mechanical water meter. It would be hard to apply this solution to other household resources. Additionally, there is no information related to the MCU part of the SM. A solution based on a water flow sensor and NodeMCU microcontroller using WiFi is described in [25]. A similar solution based on flow meters using ESP8266 is given in [26]. In [27], authors presented a solution for a water meter that emits pulses as water flows. Pulses are captured by the ESP8266 MCU and by WiFi connected to the home Raspberry Pi gateway. A robust solution based on image capturing, number detection, and recognition is given in [28]. However, the model is not usable with constrained devices (e.g. they used NVIDIA Titan V GPU). Another image-capture-based solution is explored in [29]. OpenCV and TensorFlow2 library are explored with three different DL algorithms (R-CNN, YOLOv3, and SSD). The solution is not suitable for edge usage since it takes around 100 MB

for the model and requires high computational power. An interesting solution, using a mobile phone, is given in [30] using YOLOv4, Darknet NN framework, and Tessaract OCR engine. It is also not suitable for resource-constrained devices. Recently, in [31], the authors proposed solar-based energy harvesting techniques to make a solution sustainable. They used Raspberry Pi Zero W as an MCU and WiFi for wireless connection. In another recent work [32], the retrofit solution is based on Raspberry Pi and OpenCV library, both not usable on resource-limited edge devices. Another recent image-capturing-based solution featuring high accuracy and a relatively small model is presented in [33]. A Raspberry-Pi-based solution with solid field trials is presented in [34].

2.3. Smart Meters

A study [35] analyzes the roll-out of smart electricity meters in the European Union, focusing on the cost-benefit analysis of smart metering. One of the early studies favoring CNNs for counter recognition in automatic meter reading (AMR) used them for retrofitting TMs to SMs [36]. The typical AMR system consists of three stages: counter detection, digit segmentation, and digit recognition. The proposed system did not use the image segmentation stage, making it one of the earliest studies of its kind. However, the experiments were conducted on high-end equipment (i.e., AMD Ryzen Threadripper 1920X CPU and NVIDIA Titan Xp GPU). More recently, an offline AMR system based on YOLOv5 was proposed [37]. This solution is not intended for edge use, as it relies on PyTorch and Google Cloud GPU and runs on a personal computer (Intel Core i5). Similar work, based on PyTorch and YOLOv5, for automatic digit reading from dial meters for integration into the smart grid is presented in [38].

Another popular approach for image classification problems is based on MobileNet architecture [39]. To date, the MobileNet architecture has evolved up to version 3. Although its architecture is designed for use on mobile phones, a recent study explored possibilities for deployment on very resource-constrained devices such as microcontrollers (MCUs) [40]. The results are provided for all MobileNet versions and two ShuffleNet architectures [41] using the CNN Analyzer optimizing tool [42]. The results showed that model sizes varied from 167.3 to 293.8 KB and accuracy ranged from 83.3 to 88.8%. Additionally, TinyML frameworks, such as TensorFlow Lite, equip IoT devices with efficient ML processing, making them suitable for cost-effective deployment in environments with limited bandwidth [9].

2.4. Federated Learning

A significant amount of research has been conducted in the field of FL in the IoT [43]. There are systematic reviews on the implementation of FL on resource-constrained devices [10,11]. A study elaborating on the rationale for adopting FL on MCUs and on-device training is presented in [44]. However, in metering scenarios, FL is primarily used for consumption forecasting [13,45]. Recently, authors reviewed an article focusing on TinyML, a lightweight machine learning paradigm, and FL. The authors did not find any reviews covering both fields [12]. Currently, the only study exploring lightweight machine learning and FL on various constrained devices is [46]. However, this study applied these concepts in ECG classification and driver well-being detection scenarios.

To the best of our knowledge, there is no custom retrofitting fabricated digit-recognition-based SM prototype optimized for edge, based on FL.

3. System Architecture

This section focuses on the general concept of our proposed system, which is based on our custom-made edge device. We describe the digit recognition strategy and the related problems encountered in the embedded device environment. Finally, a detailed description of our experimental setup and the methodology employed to create the dataset is provided.

3.1. General Concept

The system architecture presented in this study is outlined as follows. Initially, the edge device utilizes an embedded camera module to capture images. In comparison to

other published studies, which used standard modules or combined multiple existing modules, our approach features a custom-fabricated prototype. The captured images, which are in grayscale format, are cropped and resized to produce six 16×24 pixel images, each representing a single digit. These images are then transmitted to a cloud server, where an advanced high-level entity learning model performs inference to classify the images and assign appropriate labels. The resulting labels are transmitted back to the edge device, where they are matched with their corresponding digit images.

Communication between the edge device and the server side is facilitated through NB-IoT technology. We try to optimize input size for our proposed model, as NB-IoT technology has limitations for the number of bytes that can be transmitted per packet. In our study, each grayscale digit image consists of only 16×24 pixels, corresponding to 384 bytes. On the other hand, the maximum payload capacity for a single NB-IoT packet is 1600 bytes. Therefore, six-digit images can be transmitted using just two NB-IoT packets. This optimization is essential given the energy constraints characteristic of IoT environments. After the labeling process, the digit images are organized into dedicated labeled folders within the device's memory storage.

Subsequently, preprocessing is performed on digit images to produce clean, artifact-free black-and-white images. This preprocessing step is iteratively applied until 150 distinct digit combinations are recorded. To facilitate this, we utilize a digit change detection algorithm that enables the identification of new digits in subsequent images compared to previous captures.

Finally, we designed two experimental test cases to evaluate the system's performance across 10 devices. The first test set consists of two scenarios, where the first scenario contains two successive dataset generations and model training iterations. In the second scenario, FL is used relying on an averaging methodology after each iteration to enhance the average accuracy per device. The second test case represents a scenario in which the system is split in two batches, each containing five devices. Initially, devices from the first batch perform training and FL update. Afterwards, training is performed on the second batch, which is set up using the global model generated by the first batch.

3.2. Edge Node Design

To retrofit the existing mechanical counters, we aimed to develop a low-cost edge node with specific features that would work effectively under resource-constrained conditions. As our approach is focused on environments where high-performance hardware is not available, we designed the system to balance accuracy and efficiency. The developed node contains the following features:

- A 5-megapixel camera to capture the mechanical counter image;
- A backlight LED that facilitates camera usage even in low-light conditions;
- A low-power MCU with sufficient storage and computing capabilities to capture images, store them locally, and run the ML training + inference on-spot;
- A communication module enabling connectivity service in difficult places (closed spaces, basements, etc.) without having to rely on the existence of the local communication infrastructure such as WiFi.

The existing off-the-shelf devices fail to satisfy all of the aforementioned requirements. Therefore, we designed the edge node based on the ESP32-CAM module [47]. In addition to WiFi and Bluetooth Classic/BLE interfaces embedded within the ESP32, we included a cellular module supporting 4G services such as NB-IoT and LTE-M, as well as EGPRS fallback for areas where such new services have still not been deployed. Relying on the existing and widely deployed MNO infrastructure, we thereby strive to achieve a nearly ubiquitous connectivity. The main components and the fabricated prototype are shown in Figure 4.

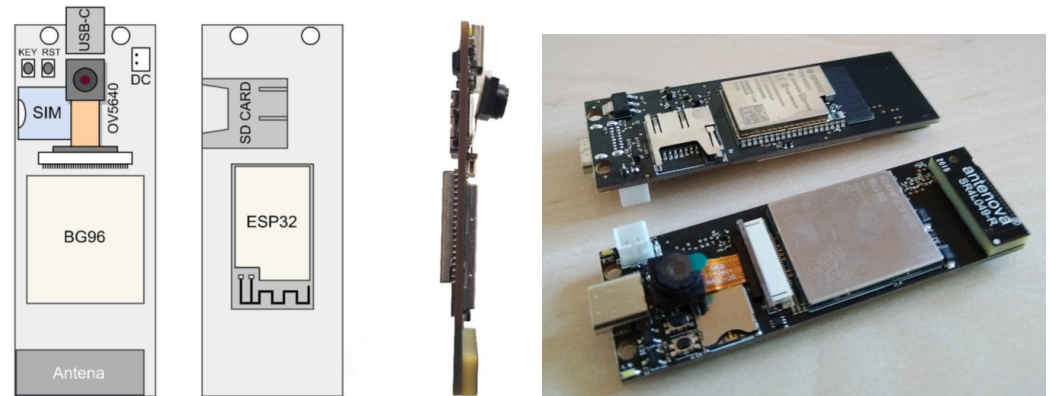


Figure 4. Design and components of edge device is depicted on the left, whereas right images display fabricated devices.

3.3. Digit Recognition Scheme

The “one-size-fits-all” approach to digit recognition assumes that we are running the inference with the same ML model on each of the edge nodes. Following the basic idea that the edge nodes will be used in a variety of environments, with differing counter types, lighting conditions, as well as relative positions between the camera and the counter, the aforementioned approach turns out to be inappropriate. Thus, our strategy relies on the capability of the edge nodes to adapt to the on-site environment by updating, re-training, and improving their local ML models.

Digit recognition is a classification problem, which, by its nature, can be solved utilizing a supervised learning paradigm: input instance is an image that is to be classified into one of the 10 categories corresponding to decimal digits. A sufficiently large dataset needs to be generated and labeled in advance to train an appropriate ML model. The easiest way to do it is to take a series of images and upload them to the server to be labeled (either manually or by using a highly accurate digit recognition model, which is beyond the scope of this work). However, several key points need to be addressed to apply this strategy as efficiently as possible:

- Full-size image produced by a 5-megapixel camera is by itself a cumbersome piece of data, for two reasons. Uploading such an image to the server would be associated with unacceptably high communication costs. On the other hand, to run an inference on such a large input instance, the ML model would have to be too large to fit into a resource-constrained edge device. Therefore, a considerable effort must be put into reducing the size of the input image, without compromising the ability to extract useful information from it.
- The nature of the problem is such that the rate by which the digits on the counter are updated is extremely unbalanced, even between the digits belonging to the same counter: starting from the rightmost (i.e., least significant) digit, every subsequent digit is updated by a tenth of the rate of the previous one. Furthermore, some counters might be used in households with people living inside, thereby changing state significantly faster than others used in remote objects, where the same utility is used seasonally and/or occasionally. This implies the necessity to implement a mechanism to reliably detect when a change in the counter state actually occurred, which would make the system robust and adaptive to the volatile nature of the measurement process.

3.4. Experimental Setup and Dataset Collection

For testing purposes, the digit image dataset was produced using the test setup shown in Figure 5. The edge node with the camera was fixed in front of the mechanical counter. As shown in Figure 5, only a part of the input image (ROI = region of interest) is used as the input for the digit recognition scheme, which is described in the following Section. In a real-life scenario, a mobile application communicates with the edge device via Bluetooth and

enables the user to define the boundaries of the ROI during the setup phase. In addition, after the device is deployed and operational, it will occasionally send a full image to the cloud to allow redefinition of the ROI and/or the camera parameters, should such an action be necessary. It will also be helpful in case of any misrecognition of digits. The digits on the counter change most rapidly in the least significant places or after the decimal point. If a digit is not recognized correctly, occasionally sending (e.g., once in a few days) the entire image of the ROI will allow for easy correction.

The counter we used can be triggered to increment its state by generating an electrical pulse from the same MCU that takes images. That way, by knowing the initial value and the exact number of generated pulses, we created a labeled dataset on the spot. The procedure for the creation of the dataset consisted of the following:

1. Successively incrementing meter values and taking photos
2. Cropping part of the image that contains digits
3. Dividing it into separate digit images (16×24 pixels)

The environmental conditions can significantly influence the recognition process [48]. Accordingly, the experiments were conducted, and the dataset was created in a dark setting (to resemble a real-world scenario, e.g., a manhole), with each image capture made using a flash LED. The captured images are preprocessed using the algorithms described in the next section to standardize the inputs of the ML model. The preprocessing procedures are designed to eliminate the potential effects of differences in the conditions under which the data were collected. Additionally, the entire apparatus was kept in a fixed position, and the validation of the results was carried out under the same laboratory conditions. The dataset consists of 3500 successive meter values, divided into 21,000 digit images.

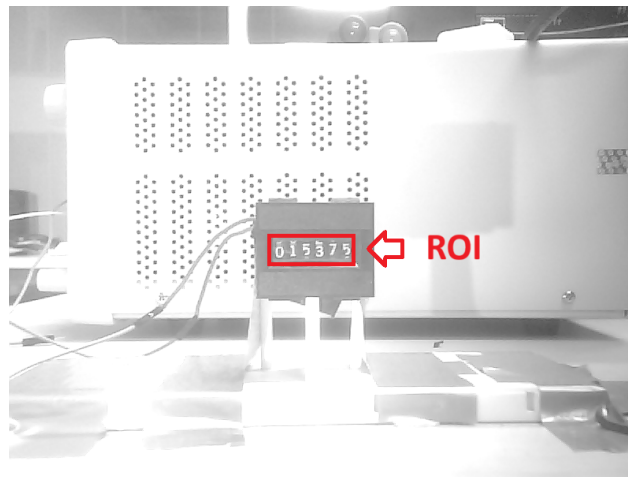


Figure 5. View of metering device through camera lens of the edge device used for creation of datasets.

4. Digit Image Processing

In the following sections, we present the detailed methodology for image preprocessing, employing a coloring algorithm to extract digits effectively from the captured images. Next, we describe a strategy for conserving energy by minimizing unnecessary transmissions, utilizing our proposed algorithm to determine whether the counter's state has changed since the previous capture. Following this, we introduce our custom-designed, lightweight CNN architecture optimized for on-device training. Lastly, we discuss our FL approach, aimed at improving the generalization of digit recognition models deployed on edge devices using a standard model averaging technique.

4.1. Image Preprocessing

As outlined in the Introduction, TMs exhibit significant variability in digit formats, fonts, colors, and sizes. To address this diversity and ensure more uniform data input

for our ML model, our system employs a series of preprocessing techniques designed to standardize the captured images.

The first step in image preprocessing, taken by the camera, is to crop the image and extract single digits, which, in this case, fit into a 16×24 frame, as shown in Figure 6. This proves to be a huge reduction of the input instance size.

Next, a transformation is applied to convert the grayscale image to a black-and-white image, which further decreases the image size by a factor of 8, as each pixel can be represented by a single bit instead of a byte. The initial approach to this transformation was to compare each pixel value to a fixed threshold and classify it as black or white accordingly. However, due to variations in lighting conditions, it is impossible to establish a single threshold value that would produce meaningful output in all cases.

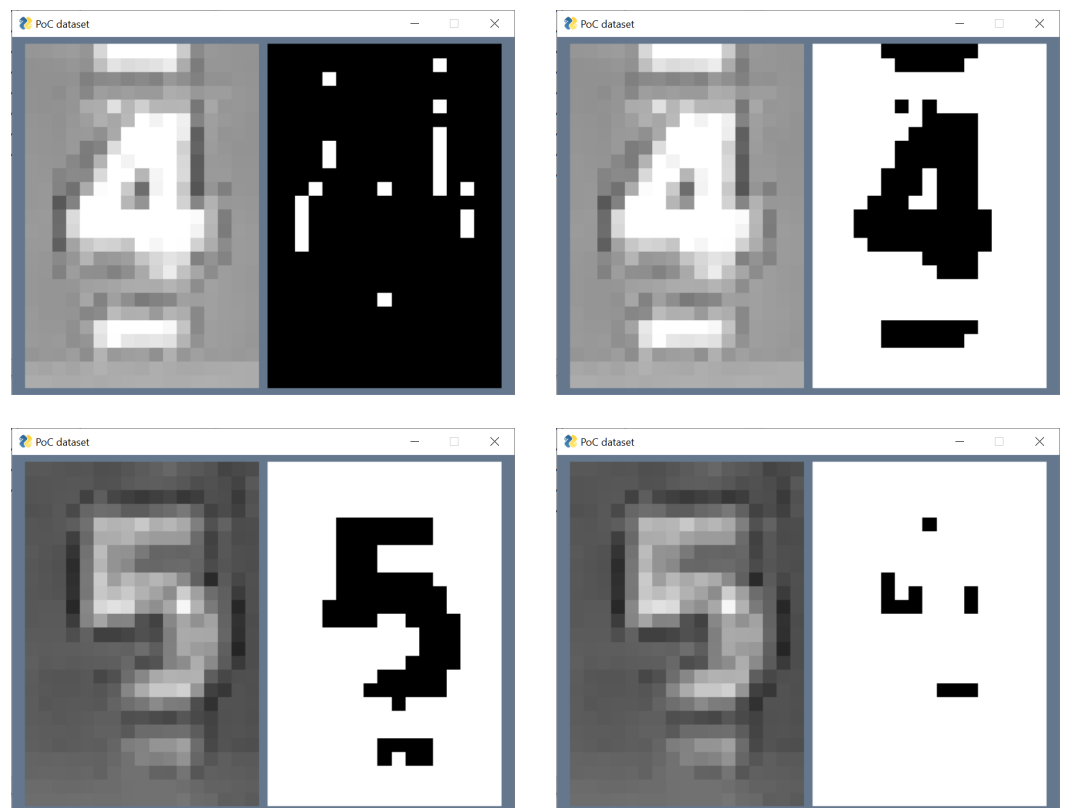


Figure 6. Conversion to B/W image format using a fixed threshold: TH = 128 (left), TH = 192 (right).

A better way to transform images to black and white is by applying the K-means clustering algorithm ($k = 2$). As shown in Figure 7 (middle part), the algorithm makes a solid distinction between black and white areas in the image.



Figure 7. Initial image taken using camera on edge device, intermediate image after B/W conversion and the final image without artifacts.

The remaining issue to be solved is the removal of white artifacts, which are parts of the figures above or below the one that needs to be detected. That is accomplished by utilizing a coloring algorithm, which paints all the disjoint white areas in the picture in different colors. In the example shown in Figure 7, the coloring algorithm will color the white areas in three different colors, as there are three disjoint areas. However, as the lower two areas are both parts of the figure that need to be detected, we apply an additional step, which is to merge any two differently colored areas that have pixels touching in corners. Then, we recognize the biggest colored area and eliminate all the smaller ones as artifacts. What remains is the actual image of the figure centered both horizontally and vertically (right-hand side of Figure 7), which concludes the preprocessing part.

The full sequence of preprocessing operations is described in Algorithm 1.

Algorithm 1 Image preprocessing algorithm

Require: Cropped 16×24 part of the grayscale image containing digit

Ensure: Clean B/W digit image without artifacts

Convert to B/W by running k -means clustering ($k = 2$) on grayscale image

Color disjoint white areas in different colors

Merge differently colored areas that have pixels touching in corners

Identify the dominant color

Paint all the dominant-colored pixels white

Paint all the rest black

4.2. Digit Change Detection

The inherent property of the utility metering scenario is that the least significant digit on the meter can be updated with an extremely variable rate. In extreme cases, update rates can vary between seconds and months. Our strategy is to preserve energy by avoiding unnecessary transmissions as much as possible, as data transmission is a much more costly operation compared to image collection and processing. Therefore, our next goal was to establish the criterion that would allow us to determine whether the state of the counter has changed since the last capture and subsequently transmit the data only if such a criterion was met.

Table 1 depicts a matrix representing median differences between two images representing the same or different digits. It is calculated, as displayed in Figure 8, by comparing each pair of images from the dataset and filling a three-dimensional matrix, in which dimensions represent the first digit, second digit, and number of pixels that differ. Initially, a 3D matrix is set to zero matrix. If the result of comparison between the pair of images (D_i^m, D_j^n) is k , where i is the i -th image in set of images of digit n and j is the j -th image in a set of images of digit m , then the 3D matrix is incremented on position (m, n, k) . Finally, after all pairs of digits have been compared, the mean value per pair (m, n) is calculated.

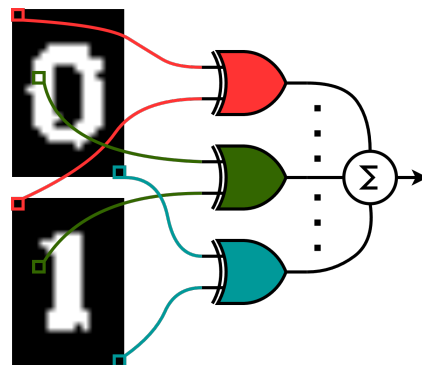


Figure 8. Scheme used for detecting differences between two digit images.

Table 1. Average digit pixel differences where each element represents average number of differing pixels.

Digit	0	1	2	3	4	5	6	7	8	9
0	19	90	62	55	79	56	43	81	47	46
1	90	14	56	60	54	64	71	43	78	72
2	62	56	28	46	77	73	65	48	57	55
3	55	60	46	24	76	50	50	52	43	49
4	70	54	77	76	16	63	64	68	69	75
5	56	64	73	50	63	24	38	68	48	50
6	43	71	65	50	64	38	25	78	38	53
7	81	43	48	53	68	68	78	18	72	64
8	47	78	57	43	69	48	38	72	29	47
9	46	72	55	49	75	50	53	64	47	31

Bold numbers indicate minimum for each column/raw that is used for calculation of general threshold.

The results align with our expectations as shown in Table 1. For instance, digit 0 differs significantly from digits such as 1, 4, and 7 due to its distinct shape, as reflected by the larger pixel difference between them. Conversely, digits with more rounded shapes, such as 6 and 9, exhibit fewer pixel differences when compared to digit 0, suggesting a greater similarity. This pattern is consistent across other digits as well. For instance, digits 8 and 3, which share similar structural features, show a smaller pixel difference. Finally, as expected, the values along the main diagonal are minimal, since they represent the comparison of each digit with itself.

The next step involves determining the optimal threshold for distinguishing whether a digit has been changed. In our paper, we relied on the empirical rule to calculate the threshold value. It follows the steps given in Algorithm 2.

Algorithm 2 Calculation of optimal value for digit image differentiation threshold

Require: Digit pixel differences table

Ensure: Optimal value of the threshold

Extract diagonal (same digits) and non-diagonal values (different digits) of the Digit pixel differences table.

Calculate the mean (μ) and standard deviation (σ) values for both diagonal and non-diagonal values.

Apply the Empirical Rule:

- 68–68% of values lie within 1 standard deviation from the mean.
- 95–95% of values lie within 2 standard deviations from the mean.
- 99.7–99.7% of values lie within 3 standard deviations from the mean.

Based on the chosen rule, set the threshold where the ranges of the diagonal and non-diagonal values start to overlap.

In our case, choosing empirical rule 95% gives a threshold of value of 32.5, which will be used for distinguishing different digits.

With that being said, our approach involves analyzing the pixel differences among each digit (0–9) and developing a predictive mechanism to detect when a digit has been changed. Our algorithm proposed for this task is presented in Algorithm 3.

Algorithm 3 Pixel difference algorithm

Require: Clean least significant B/W digit image without artifacts from the last taken picture
Ensure: the least significant digit change detected
 Take a new picture
 Perform image preprocessing algorithm
 Do the EX-OR operation between corresponding binary pixels of last and new image of least significant digit image
 Sum elements of the resulting pixel matrix
 Compare it with the threshold defined by Table 1
 Conclude if the least significant digit has been changed

4.3. Digit Recognition Model

This paper presents a lightweight variant of a CNN with seven layers designed for the classification task of digit recognition. The architecture follows a common deep-learning paradigm involving convolution, pooling, normalization, flattening, ReLU activation functions, and a softmax output layer. This kind of architecture is well-proven for image classification tasks, leveraging convolution for feature extraction, max pooling for dimensionality reduction, batch normalization for training stability, and fully connected layers for decision-making based on the learned features.

A short description of each network layer is given next and also depicted in Figure 9. Given the resource constraints of the ESP32 relative to modern high-capacity CNN architectures, our objective was to design lightweight yet effective CNN leveraging standard convolutional blocks. The initial section of our CNN comprises a conventional convolutional block, which includes a convolutional layer, max pooling layer, and batch normalization. This block produces a feature map with dimensions $3 \times 4 \times 32$, beyond which further convolutional blocks could not be employed due to the spatial limitations of the feature map.

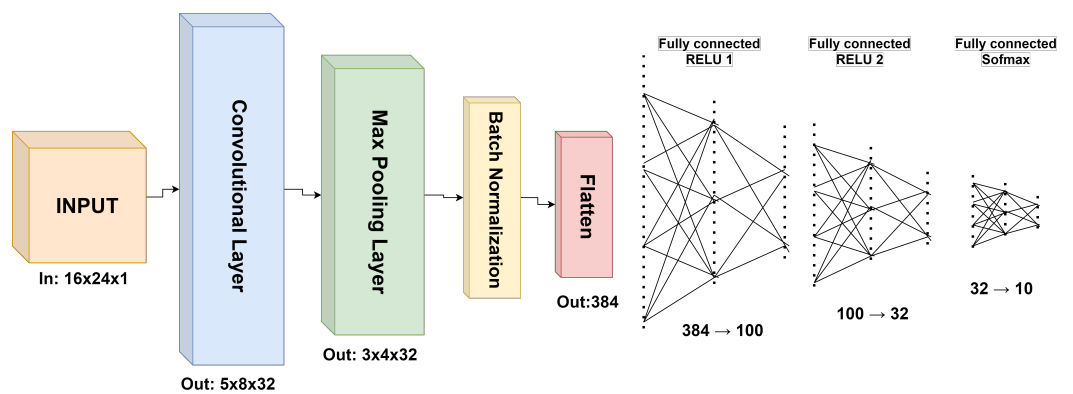


Figure 9. Proposed CNN architecture used for digit recognition.

To address this, we utilized a flatten layer to transform the 3D matrix into a 1D array, serving as the input for subsequent fully connected (dense) layers. We incorporated three fully connected layers, each with ReLU, ReLU, and softmax activation functions, respectively, to progressively reduce the dimensionality of the intermediate feature maps. Finally, the last fully connected layer employs a softmax function, yielding a prediction corresponding to the specific digit value displayed on the metering device.

4.3.1. Convolution Layer

The input to the network is a tensor with dimensions $16 \times 24 \times 1$, representing a single-channel (grayscale) image with a spatial size of 16×24 pixels. This layer applies

32 convolutional kernels, each of size $3 \times 3 \times 1$, over the input. No padding is applied, and the stride for the convolution is set to 3 in both the x and y directions. The convolution operation outputs a feature map of dimensions $5 \times 8 \times 32$, where 32 is the number of filters (kernels) applied, and 5×8 is the spatial size of each feature map.

4.3.2. Max Pooling Layer

The max-pooling operation reduces the spatial dimensions of the feature maps while retaining the most salient features. This layer uses a pooling window of 2×2 and reduces the spatial dimensions from 5×8 to 3×4 . The number of channels (32) remains unchanged.

4.3.3. Batch Normalization Layer

This layer normalizes the activations from the previous layer to accelerate and stabilize the training process. It applies batch normalization across the three-dimensional output with a momentum value of 0.900002.

4.3.4. Flatten Layer

The three-dimensional tensor from the previous layer is flattened into a one-dimensional vector in order to prepare the data for the subsequent fully connected layers. The total number of elements in this flattened layer is $3 \times 4 \times 32 = 384$.

4.3.5. ReLU Layer (First Fully Connected Layer)

This fully connected layer receives the flattened input of size 384 and applies a rectified linear unit (ReLU) activation function. The output dimension is 100, representing the first hidden layer of the fully connected network.

4.3.6. ReLU Layer (Second Fully Connected Layer)

The second fully connected layer takes the output from the previous layer (of size 100) and, again, applies a ReLU activation function. The output dimension is 32, representing a second hidden layer.

4.3.7. Softmax Layer (Third Fully Connected Layer)

The final layer of the model is a fully connected layer with a softmax activation function. This layer generates a probability distribution across ten output classes, which are, in our case, the digits 0 through 9. The softmax function ensures that the output values sum to 1, making them suitable for this classification task.

Furthermore, our approach facilitates not only edge inference but also comprehensive model training. To achieve this, we implemented a standard gradient descent backpropagation algorithm that processes each data batch during training. The calculated gradients are employed to update the model weights in accordance with the designated batch size. In our implementation, we optimized the training procedure by excluding dropout layers. Additionally, the application is developed in a bare-metal environment, avoiding the use of external frameworks or libraries to minimize overhead and memory footprint. This design enhances the applicability of our approach across a wide range of edge devices.

4.4. Federated Learning Approach

FL is a decentralized approach that allows multiple devices to collaboratively train ML models without sharing their raw data in order to increase the performance of each model, as well as any new model that can be included in the system. This method ensures data privacy by transmitting only models rather than sensitive information to a central server for aggregation.

Having safety concerns in mind, in our study, FL is used to enhance the generalization of our digit recognition models located on edge devices and to create a more robust model capable of accurately predicting values under different environmental parameters.

In our study, we utilize a standard averaging methodology to implement FL, which is based on averaging the same parameters (weights or biases) across all participating models to create a single global model. However, the batch normalization layer is treated as an exception to this averaging process, as the layer should remain unchanged during the average procedure. Consequently, if there are C devices, each with a CNN model, the FL process will yield C distinct models, where all layers—except for the batch normalization layer—are identical, and the batch normalization layer is customized for each device. This means that the transmission of models between server and devices can be carried out with or without batch normalization parameters. In our case, the model is transmitted in total.

5. Results

In this section, we describe two experimental test cases used in our study, along with their corresponding results. Additionally, we measure and evaluate the energy consumption to showcase the efficiency of the proposed architecture in real-life working conditions. Finally, we provide a comparison of our solution with other similar approaches found in the literature.

5.1. First Test Case

Our system integrates digit change detection and model training, as well as FL. To evaluate its performance, we design the first test case with two experimental scenarios. The first scenario serves as a baseline for comparison with the second, focusing exclusively on model training using two datasets. The second scenario incorporates the benefits of FL. Both scenarios are illustrated in Figure 10.

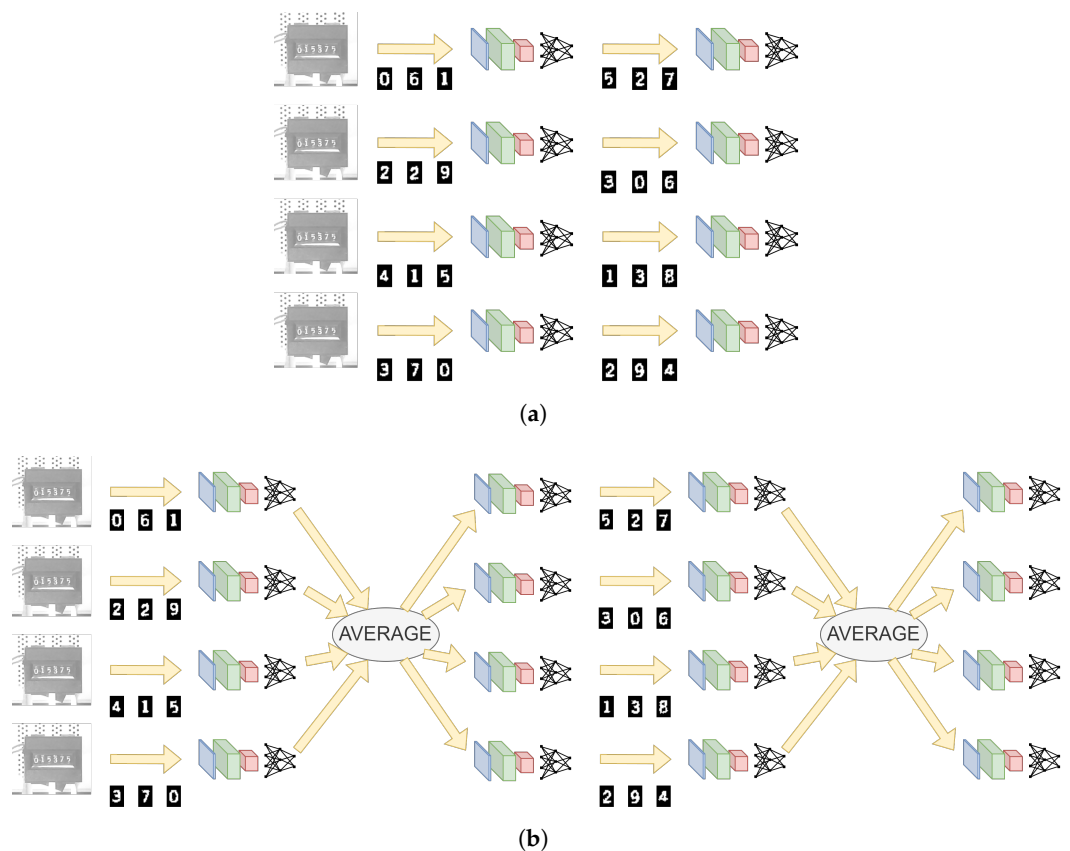


Figure 10. The first test case comprises two distinct scenarios representing different training methodologies, one incorporating federated learning (FL) and the other without its use. (a) Scenario 1, which does not use FL. (b) Scenario 2, which utilizes averaging methodology for FL.

In Scenario 1, shown in Figure 10a, we implement a two-stage training process for the model on individual devices. The first stage is started by gathering N images, which are

sent to a global server for labeling. These labeled digit images are subsequently used for model training on the device in the first iteration. In the second stage, a new batch of N images is collected and labeled in the same manner, allowing for further model training. The described procedure is repeated across all C devices, leading to C distinct models.

Scenario 2, depicted in Figure 10b, introduces FL following each training stage. Initially, each device trains its model using the digit image dataset it collected. Afterward, the devices share their model parameters through a federated averaging methodology as described in Section 4.4. After parameter sharing, each device continues training its model on the next batch of digit images, resulting in further refinement of all individual device models. The first averaging process is more substantial, as the models originate from diverse random seeds and datasets, leading to considerable variance among them. In contrast, the second averaging is more conservative, since the models have become more similar despite the additional training.

The described scenarios were evaluated using ten client devices (C = 10). Each device was assigned a combination of two datasets, each containing 150 images, leading to two datasets of 900 digits, referred to as D1 and D2. These datasets are the same for both scenarios. Furthermore, an additional testing dataset was created consisting of 500 images (3000 digit images in total) and designated for testing both scenarios at each stage. Furthermore, all training sessions consist of 25 epochs. The results presented in Table 2 display the accuracy calculated on this testing dataset.

The row labeled S = 0 and T = D1 in Table 2 report the accuracy of models trained solely on dataset D1. The next row, marked with S = 1 and T = D2, presents the results obtained after completing the first scenario. These results were derived from consecutive training on D1 and D2 without utilizing FL.

Table 2. Results of the evaluation for both scenarios.

		C									
S	T	0	1	2	3	4	5	6	7	8	9
0	D1	90.37	88.70	87.07	78.90	92.53	91.03	87.06	79.36	91.34	87.70
1	D2	92.07	97.43	87	85.10	91.34	23.33	89.63	93.13	95.8	91.23
2	Avr1	78.20	74.77	74.43	73.20	76.33	76.63	76.13	76.20	75.80	75.07
	D2	94.33	93.56	91.77	95.23	95.4	94.64	90.63	94.07	94.03	92.33
	Avr2	93.5	93.30	92.87	93.90	93.93	93.90	94.20	93.67	93.00	93.14

The bold values represent maximal values per column.

The improved performance observed in the second iteration of FL using the averaging methodology can be attributed to the distinct initialization conditions of the models. Initially, the models are configured with randomized weights and biases, resulting in substantial heterogeneity across individual models. Following the first round of training, these models diverge considerably in their weight and bias configurations. Consequently, averaging these disparate models produces a global model that differs notably from any individual local model, which can lead to a less effective aggregation in the first iteration of FL.

In contrast, the second training iteration begins with a unified global model, meaning that each local model in this round is derived from a common starting point. As a result, the individual models exhibit variations of a shared structure, and differences between them are significantly reduced compared to the first iteration. Thus, averaging in the second iteration causes less disruption to the individual models, enabling a more cohesive integration. This approach facilitates a more effective distribution of learned knowledge across devices.

Rows designated as S = 2 indicate both intermediate and final model accuracy in Scenario 2. It is noticeable that the initial federated averaging significantly diminishes model accuracy as expected. However, additional training on dataset D2 leads to marked accuracy

improvement in comparison to models trained without FL. Finally, the results suggest that the second application of FL yields, on average, superior performance compared to models trained without this methodology, as shown in Table 3.

Interestingly, the results indicate that Scenario 2 produces comparable outcomes before and after the second federated averaging procedure. The ratio between the models labeled $S = 2, T = \text{Avr2}$ and $S = 2, T = \text{D2}$ is 0.999, suggesting that “knowledge” is effectively shared among the client devices without a significant loss of accuracy on average.

Table 3. Average accuracy for both scenarios.

Average	$S = 0, T = \text{D1}$	$S = 1, T = \text{D2}$	$S = 2, T = \text{Avr1}$	$S = 2, T = \text{D2}$	$S = 2, T = \text{Avr2}$
Accuracy	87.40	91.41 *	75.67	93.60	93.54

*—Average does not include accuracy for device client 5. The bold values represent maximal values in row.

5.2. Second Test Case

The second test case is designed to simulate real-world scenarios in which devices are deployed and trained in batches at different time intervals, as illustrated in Figure 11. In this test case, ten clients are divided into two batches of five clients each. The first batch consists of the initially deployed devices, following a training process similar to Scenario 2. These devices collect an initial dataset of N images and train their initial, randomized models on these data. The next step involves FL using a model averaging approach, followed by another cycle of data collection, model training, and FL. As a result, the cloud stores a second version of the global model, generated after two rounds of training.

Subsequently, the second batch of devices is introduced. Their initial model is set to the global model available at that point. These devices follow the same process of two rounds of dataset collection (each with N images), model training, and FL updates using the averaging methodology. The key difference is that for these FL updates, the cloud incorporates all the latest models produced during training. This includes models from the first batch, which were generated after training on their second dataset.

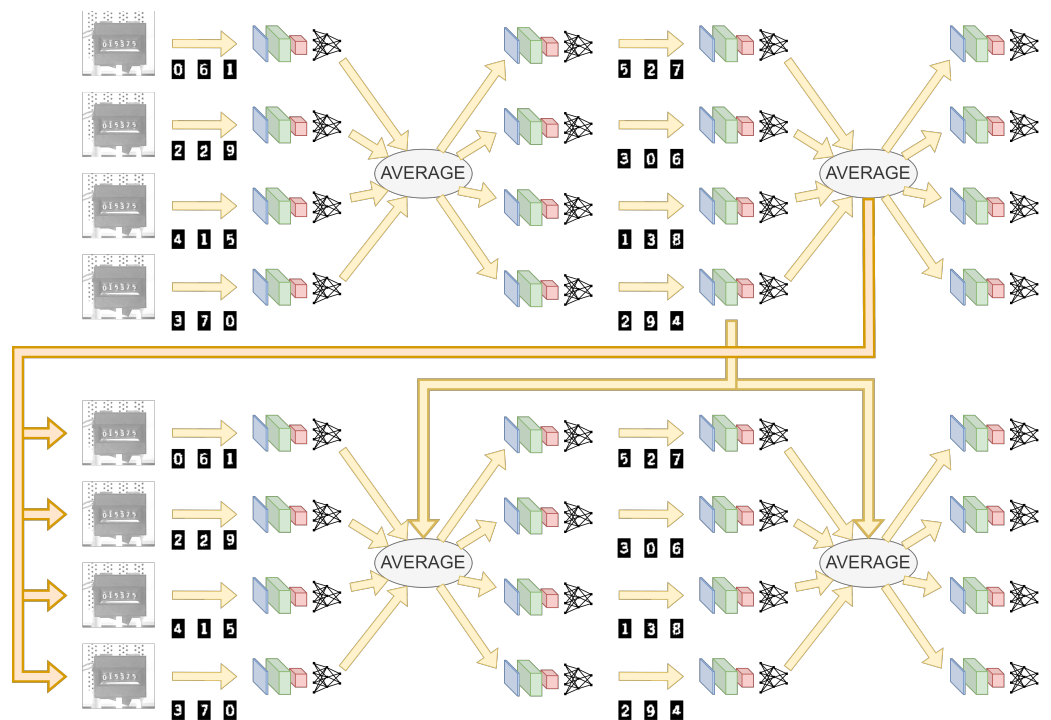


Figure 11. Second test case displaying training scheme where second batch of devices is trained based on results of training on first batch.

The evaluation of the second test case is conducted using the same parameter settings as in the first test case. The number of images per training dataset, N , is set to 150, while the test set consists of 500 images. Each image is divided into six digit images, resulting in 900 digit images per training dataset and 3000 digit images for the testing dataset. Furthermore, each training session consists of 25 epochs. The accuracy evaluation results on testing dataset are presented in Table 4.

Table 4. Results of the evaluation for second test case.

T	Batch 1					T	Batch 2				
	0	1	2	3	4		5	6	7	8	9
D1	90.37	88.70	87.07	78.90	92.53						
F1	66.73	62.33	65.03	66.33	70.20						
D2	95.50	93.80	90.93	95.53	95.27						
F2	92.80	94.27	93.43	94.30	94.60						
D2 *	95.50	93.80	90.93	95.53	95.27	D1	96.73	96.90	97.73	96.07	95.97
F3	96.33	96.77	96.60	97.50	97.27	F3	96.77	97.03	96.87	96.63	96.30
D2 *	95.50	93.80	90.93	95.53	95.27	D2	97.30	96.67	96.83	96.50	96.47
F4	96.33	96.57	96.77	97.53	97.57	F4	97.50	97.23	97.03	96.67	96.87

*—Results repeated without training for Batch 1. The bold values represent maximal values per column.

The columns labeled T in Table 4 indicate the point at which model accuracy was evaluated. D1 refers to accuracy following training on the first dataset, while D2 represents results after the second training session on the second dataset. Rows labeled as D2 * contain repeated results for devices in Batch 1 after training on the second dataset. Finally, rows beginning with F represent the results following FL updates.

As observed in the first test case, the initial FL update leads to a reduction in model accuracy. As in the first test case, a similar trend is evident in the second FL update, where it is shown that the second FL iteration did not significantly alter the overall average accuracy.

However, the inclusion of models from the second batch highlights the importance of FL updates. As shown in the columns labeled Batch 2 in Table 4, the training results after the first dataset are notably higher than those from Batch 1. This suggests that FL plays a critical role in generating global models that serve as robust initial configurations for newly deployed devices. Moreover, performing FL updates using the latest models from all devices results in improved global models. This is evident in the final row, which shows the accuracy of the final global model deployed to each device, with most columns exhibiting their highest accuracy in this row.

This finding is significant for two reasons. First, even though training was stopped for devices in Batch 1, the FL update considerably improved their accuracy. Second, devices in Batch 2 also experienced an increase in accuracy, reaching their maximum values. Furthermore, Table 4 demonstrates that all maximum accuracy values were achieved following FL updates, underscoring the value of FL in enhancing model performance

Table 5 presents the average accuracy results for specific rows. Columns labeled B = 1 show the average accuracies for Batch 1 before and after the second iteration of the FL update. The results indicate that the second FL iteration does not significantly affect the overall average accuracy for Batch 1. In contrast, the columns B = 1, T = D2 and B = 1,2, T = D2 represent the average accuracy for models after the final training session using the second dataset for Batch 2 and for both batches combined, respectively. These columns reveal that Batch 2, initialized with the global model generated from training on Batch 1, slightly outperforms Batch 1.

The final column contains the average accuracy for the last row in Table 5, where the value of 97.01 represents the overall maximum for both Table 4 and across both test

cases. This indicates that the FL update not only improves the performance of newly added devices but also enhances the accuracy of devices that were already deployed.

Table 5. Average accuracy for second test case.

Average	B = 1, T = D2	B = 1, T = F2	B = 2, T = D2	B = 1,2, T = D2	B = 1,2, T = F4
Accuracy	94.21	93.88	96.75	95.48	97.01

The bold values represent maximal values in row.

5.3. Energy Consumption Analysis

With the lithium-ion battery we used for our test cases (3.6 V, 1000 mAh), the total energy budget available is: $E_{battery} = 3.6 \text{ V} \times 1000 \text{ mAh} = 3600 \text{ mWh}$.

ESP32 supports several power saving modes [49], enabling our device to achieve current consumption as low as 10 μA in deep sleep mode. That way, the daily energy consumption of a sleeping device is $E_0 = 3.6 \text{ V} \times 10 \mu\text{A} \times 24 \text{ h} = 864 \mu\text{Wh}$. We measured the energy costs during two critical operations, using an Otii Arc Pro Power profiler device:

- Image collection and processing
- Data transmission over NB-IoT link

The typical values for the energy consumption during different operations are shown in Table 6. The first measurement (Figure 12) was for the operation when the edge node wakes up, captures the image from the camera, executes the preprocessing and inference, and goes back to sleep. In such a case, the total energy spent for the operation is $E_1 = 64 \mu\text{Wh} + 149 \mu\text{Wh} + 36 \mu\text{Wh} = 249 \mu\text{Wh}$.

Table 6. Energy consumption during different operations.

Operation	Energy [μWh]
MCU init	64
Image capture	149
Preprocessing + inference	36
MCU init	123
Network connection	1360
Data transmission	570

The other operation (Figure 13), which serves for the estimation of the communication costs, assumes that the edge node wakes up, connects to the network, transmits a data packet, and goes back to sleep. The NB-IoT module that goes to power save mode (PSM) upon each transmission skips the network registration phase on all subsequent transmissions after the initial one [50]; therefore, the cost of the network registration is here omitted from the calculation: $E_2 = 123 \mu\text{Wh} + 570 \mu\text{Wh} = 693 \mu\text{Wh}$.

As expected, the data transmission operation turns out to be more expensive than the image capture + processing operation by the factor of $E_2/E_1 = 2.78$. This increases to a factor of 8.24 when network registration is necessary, which must be carried out every time when the NB-IoT service is not available. This clearly justifies our approach to try to detect whether the digits were updated and to transmit data only if necessary.

In a hypothetical use case when measurements are taken once every hour, the data are transmitted once a day, and the rest of the time is spent in deep sleep mode, the daily energy consumption is $E_{daily} = E_0 + 24 E_1 + E_2 = 7533 \mu\text{Wh}$. The total battery lifetime can be calculated as $E_{battery}/E_{daily} = 477$ days. This is a fairly intensive usage scenario, while in practice, the number of processing/transmission operations can be significantly reduced, thereby prolonging the battery lifetime to several years.

To enable comparison with other solutions available in the literature, we calculated the average current consumption as $I_{average} = E_{daily}/24 \text{ h}/3.6 \text{ V} = 87 \mu\text{A}$. This is by at least

an order of magnitude more energy-efficient in comparison with other solutions, as shown in the following subsection.

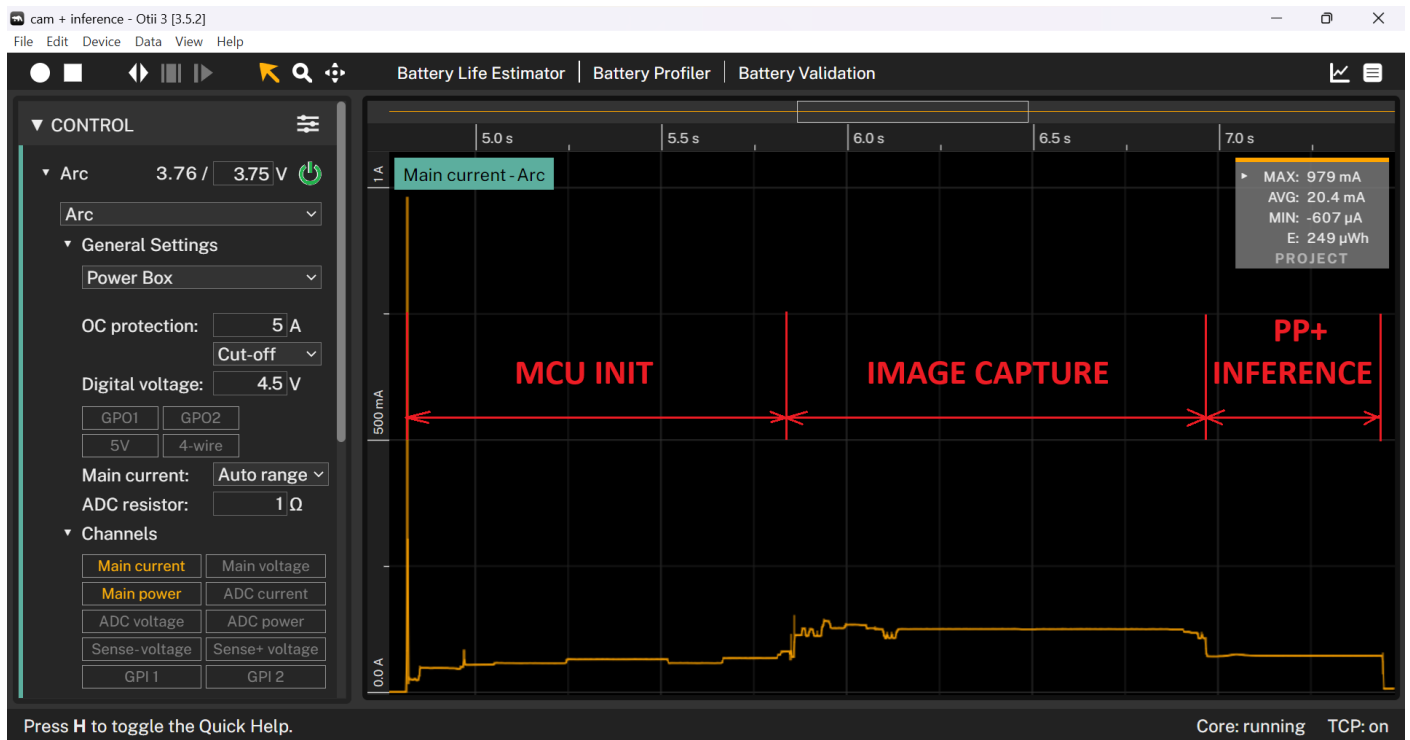


Figure 12. Power consumption profile of image capture + preprocessing + inference.

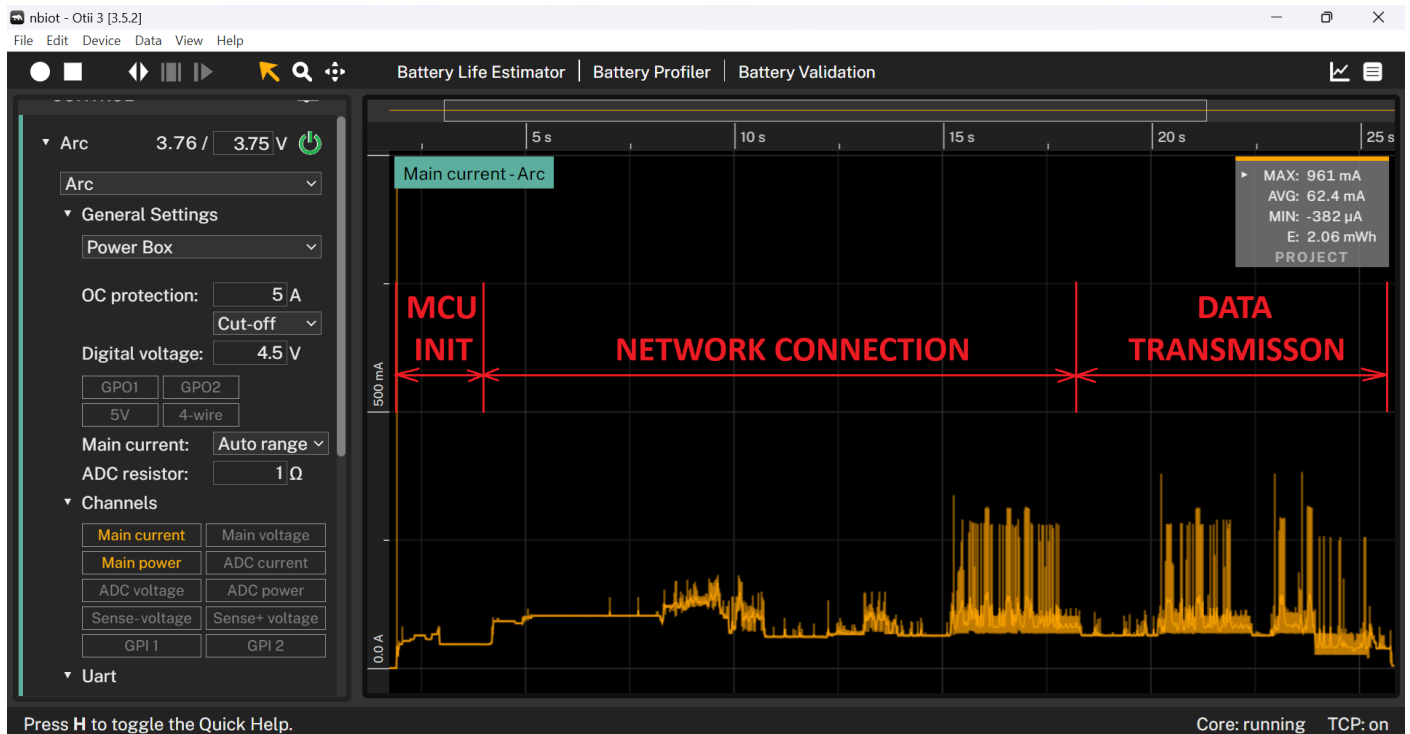


Figure 13. Power consumption profile of data packet transmission via NB-IoT.

5.4. Comparison with Other Solutions

Table 7 provides a comprehensive comparison of key attributes across several related works in this field. Some key observations are the following.

Table 7. Comparisons of related work.

Paper	[34]	[31]	[30]	[33]	[40]	OUR
Device type	PE	PE	MA	-	MCU	NP
Comp. device	RPi3b+	RPi Z	M	-	STM32	ESP32
Communication	WiFi	WiFi	Mobile	WiFi	-	NB-IoT
Framework	OCV, TF	-	Darknet	TF	CNN A.	-
Recognition	AoI+D	AoI	I	I	I	AoI+D
Power supply	B	BEH	B	N/A	N/A	B
ML model	CNN	RF	CNN	CNN	CNN	CNN
FL	-	-	-	-	-	+
Quantitative comparison						
Image size	28 × 28	-	224 × 224	60 × 200	96 × 96 × 3	16 × 24
Model size	N/A	30 MB	N/A	4.3 MB	[167:294] KB	172 KB
Battery [mAh]	9800	5000	High	N/A	-	1000
Consumption	>420 mA	>78 mA	High	N/A	-	87 μA (avg.)
Battery lifetime	46 h	578.3 d	-	-	-	477 d
Accuracy [%]	98.70	97.69	98.67	98.70	[83:88.8]	97.01

PE—prototype using existing modules; NP—new prototype; MA—mobile app.; OCV—OpenCV; RF—random forest; TF—tensorflow; AoI—area of interest; I—image; D—digit; B—battery; BEH—battery with energy harvesting.

As can be seen from the table, various platforms were used across the related works, including Raspberry Pi models (RPi3b+ and RPi Zero W) and smartphone devices. Our approach utilizes Espressif’s ESP32, a low-power MCU, which represents a significant contribution in terms of energy efficiency compared to the other solutions that rely on more power-demanding devices.

In contrast to previous systems that utilize different ML frameworks, such as TensorFlow (TF) or Darknet, our approach employs a lightweight, custom-made, barebone CNN architecture, optimized for direct training on edge devices. Furthermore, our method integrates CNNs with FL, which sets it apart from earlier works that do not leverage this technique.

Power consumption is a critical factor in embedded systems, especially those deployed in resource-constrained environments. As previously mentioned, in our approach, constrained resources in terms of the hardware capabilities of the edge devices lead to lower consumption (87 μA on average) than alternatives (420 mA and 78 mA on average). We used the smallest battery (1000 mAh), compared to other solutions and achieved a solid estimated battery lifetime of 477 days. Additionally, our approach utilizes NB-IoT technology for data transfer, which is, in general terms, more power-efficient compared to WiFi or mobile network solutions used by other systems.

Memory efficiency is also highlighted, where our model has a significantly smaller footprint (172 KB) compared to the larger memory requirements of other systems (up to 30 MB).

The accuracy comparison reflects that our solution presents results that are around 1.7% lower than those of competitors. However, we consider it a solid result, having in mind the constrained hardware resources we used to achieve it. The only other work we can compare our work with, but only in terms of lightwightness (using ESP32), is [46]. The authors used ML and FL on resource-constrained devices in different scenarios than ours and achieved an accuracy of up to 86.5%.

6. Conclusions

In this paper, we present a low-cost and power-efficient solution for retrofitting the existing mechanical-based metering infrastructure. This solution is based on a low-power MCU platform utilizing NB-IoT network technology for communication between edge devices and the server. The device is fitted with a low-cost camera that produces images of the metering device, resulting in datasets consisting of 3500 six-digit images. Batches of 150 images are labeled by the high-level entity learning model and preprocessed in order to remove unwanted artifacts and convert them to black-and-white format. Reduction from the original image size of 640×480 pixels to 16×24 format brings a reduction of input instance size by 99.87%. Furthermore, conversion from 8-bit grayscale to black-and-white bitmap brings an additional eight-fold reduction in size, totaling 48 bytes per image, as opposed to the 300 KB provided by the camera. The resulting digit image datasets are used to train models on each device using a custom-tailored CNN architecture optimized for our system.

We devised two test cases for testing. The first test case consists of two scenarios, where the first one performs two subsequent dataset generation and model training iterations. On the other hand, the second scenario is enhanced by the introduction of FL based on model averaging methodology after each training iteration on the device. The two scenarios showed the advantage of using FL, where the model obtained by the combination of two training iterations and intermediate FL averaging produces better average accuracy than the model obtained by only two iterations of training. To conclude, the resulting average accuracy of models generated using FL is 93.6%.

The second test case demonstrates the influence of FL on a system comprising two batches of devices deployed at different times. The first batch undergoes training with FL, as described in Scenario 2 of test case 1. However, the initialization of the second batch, which is based on the global model generated from training on Batch 1, shows a notable impact, with faster improvements in accuracy for Batch 2. Additionally, further training on devices in Batch 2 significantly affects the models in Batch 1, as the FL update “shares” new knowledge, leading to a substantial accuracy increase in Batch 1. Ultimately, the highest accuracy of 97.01% is achieved following the final FL update, representing the best overall performance.

The choice of low-end components and hardware is justified by the power-efficient design of the edge device, making it capable of operating for more than a year in an extensive use case and even more when the frequency of readings is reduced. This is extremely important for the feasibility and scalability of the system in a real environment. Usage of a power-hungry high-end device would impose a huge maintenance problem associated with the need for daily recharging/replacement of batteries for thousands of devices while not bringing any significant boost in the performance.

Current Limitations, Real-World Application, and Future Work

This paper presents the results achieved after completing the “Proof of Concept” project titled “Federated Learning-Based Number Recognition for Smart Metering Applications Using NB-IoT Wireless Technology”. The results demonstrate that the proposed solution could be applied in real-world metering scenarios.

Currently, our solution works well if there is a signal from the MNO (either using NB-IoT or falling back to GSM). If there is no MNO signal in the region, a possible and viable solution would be to redesign our system to work with LoRa/LoRaWAN. However, the cost of such a versatile solution would be significantly higher.

For a real-world application, it is necessary to consider several additional aspects. Most mechanical meters are located underground (e.g., in manholes) or in cabinets behind concrete walls, where radio conditions are poor. For this reason, we proposed NB-IoT featuring coverage extension modes to boost the signal in such scenarios. Poor lighting conditions can also have a significant impact [48]. To address this, we designed a prototype with white LEDs. Another important aspect is the appropriate casing and placement of

the prototype. It should be mounted in a secure plastic casing with suitable waterproof, dustproof, and disruption-proof fittings and mounting points. To address the imperfect digit recognition, we propose sending an occasional complete ROI image to the server to verify the digit recognition process periodically. Finally, appropriate adjustments (e.g., ROI) and model fine-tuning need to be performed during the installation and setup process for the particular scenario and TM.

For the future work, the following should be considered to improve the efficiency of the proposed system:

- test more sophisticated FL techniques to achieve even better performance on the edge;
- experiment with various CNNs and the other model architectures;
- subject the system to additional test cases;
- build an appropriate secure plastic 3D model with appropriate fittings, as a case for the fabricated prototype;
- test performance under different real-world metering scenarios.

Author Contributions: Conceptualization, M.L., D.V. and I.M.; Methodology, M.L., D.V. and I.M.; Software, V.N., D.B. and M.L.; Investigation, V.N. and D.B.; Resources, M.L.; Data curation, V.N. and D.B.; Writing – original draft, V.N., D.B., M.L. and I.M.; Supervision, I.M.; Project administration, I.M.; Funding acquisition, D.V. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the Ministry of Science, Republic of Serbia, through project No. 451-03-65/2024-03/200156 “Scientific and Artistic Research Work of Researchers in Teaching and Associate Positions at the Faculty of Technical Sciences, University of Novi Sad”, by the Innovation Fund, Serbia, Proof of Concept project 5367, and was funded in part from the Horizon 2020 research and innovation staff exchange grant agreement No. 101086387.

Data Availability Statement: The data are available at <https://github.com/dbortnik/digitsDataset>, accessed on 30 October 2024.

Acknowledgments: The authors appreciate the support from Dragan Danilovic and A1 Serbia for providing access to the A1 NB-IoT service.

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

Abbreviations

The following abbreviations are used in this manuscript:

SM	Smart meter
REST	Representational state transfer
MQTT	Message queuing telemetry transport
CoAP	Constrained application protocol
TM	Traditional meter
NB-IoT	Narrowband Internet of Things
PSM	Power saving mode
MNO	Mobile network operator
ML	Machine learning
MCU	Microcontroller unit
CNN	Convolutional neural network
FL	Federated learning
ROI	Region of interest

References

1. Popli, S.; Jha, R.; Jain, S. A Survey on Energy Efficient Narrowband Internet of Things (NB-IoT): Architecture, Application and Challenges. *IEEE Access* **2018**, *7*, 16739–16776. [[CrossRef](#)]
2. Peiris, S.; Lai, J.; Kumaraswamy, M.; Hou, H. Smart Retrofitting for Existing Buildings: State of the Art and Future Research Directions. *J. Build. Eng.* **2023**, *76*, 107354. [[CrossRef](#)]

3. Prakash, S.; Stewart, M.; Banbury, C.; Mazumder, M.; Warden, P.; Plancher, B.; Reddi, V.J. Is TinyML Sustainable? *Commun. ACM* **2023**, *66*, 68–77. [[CrossRef](#)]
4. David, R.; Duke, J.; Jain, A.; Janapa Reddi, V.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Wang, T.; et al. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. In Proceedings of the Machine Learning and Systems, Virtual, 5–9 April 2021; Volume 3, pp. 800–811.
5. El-Ouazzane, R. *A Tsunami of TinyML Devices Is Coming*; EE Times: Portland, OR, USA, 2023.
6. Lin, J.; Zhu, L.; Chen, W.M.; Wang, W.C.; Gan, C.; Han, S. On-device training under 256KB memory. In Proceedings of the 36th International Conference on Neural Information Processing Systems—NIPS '22, Red Hook, NY, USA, 28 November–9 December 2022.
7. Yang, F.; Jin, L.; Lai, S.; Gao, X.; Li, Z. Fully Convolutional Sequence Recognition Network for Water Meter Number Reading. *IEEE Access* **2019**, *7*, 11679–11687. [[CrossRef](#)]
8. Konečný, J.; McMahan, H.; Yu, F.; Richtárik, P.; Suresh, A.; Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv* **2016**, arXiv:1610.05492. [[CrossRef](#)]
9. Schizas, N.; Karras, A.; Karras, C.; Sioutas, S. TinyML for ultra-low power AI and large scale IoT deployments: A systematic review. *Future Internet* **2022**, *14*, 363. [[CrossRef](#)]
10. Imteaj, A.; Thakker, U.; Wang, S.; Li, J.; Amini, M.H. A Survey on Federated Learning for Resource-Constrained IoT Devices. *IEEE Internet Things J.* **2022**, *9*, 1–24. [[CrossRef](#)]
11. Ferreira da Silva, L.; Sadok, D.; Endo, P. Resource optimizing federated learning for use with IoT: A systematic review. *J. Parallel Distrib. Comput.* **2023**, *175*, 92–108. [[CrossRef](#)]
12. Qi, P.; Chiaro, D.; Piccialli, F. Small models, big impact: A review on the power of lightweight Federated Learning. *Future Gener. Comput. Syst.* **2025**, *162*, 107484. [[CrossRef](#)]
13. Abdulla, N.; Demirci, M.; Ozdemir, S. Smart Meter-Based Energy Consumption Forecasting for Smart Cities Using Adaptive Federated Learning. *Sustain. Energy Grids Netw.* **2024**, *38*, 101342. [[CrossRef](#)]
14. Li, X.; Zhang, L.; Zhou, J.; Wu, F.; Sattler, F.; Li, Q. Fedbn: Federated Learning on Non-IID Features via Local Batch Normalization. *arXiv* **2021**, arXiv:2102.07623. [[CrossRef](#)]
15. Chen, Y.; Xu, Y.; Cai, W.; Li, Z.; Hu, L. FedHealth 2: Weighted Federated Transfer Learning via Batch Normalization for Personalized Healthcare. *arXiv* **2021**, arXiv:2106.01009. [[CrossRef](#)]
16. Huijun, Y. Internet of Things Water Meter Based on NB-IOT Communication Infrastructures. PRC Patent Application No. 108548582, 25 May 2018
17. Young, N.; Li, S.; Wang, Y.; Zhang, X. Mobile Internet of Things Case Study, Greater China. In *GSMA IoT Case Study*; GSMA: London, UK, 2018. Available online: <https://www.gsma.com/iot/wp-content/uploads/2018/03/GSMA-IoT-Case-Study-Greater-China-EN-March-2018.pdf> (accessed on 28 September 2024).
18. Kamstrup NB-IoT Portfolio. Available online: <https://www.kamstrup.com/en-en/news-and-events/news/kamstrup-adds-nb-iot-to-portfolio> (accessed on 28 September 2024).
19. Vivek, B.; Viswanath, S.; Swarnalatha, P. Design and Development of LoRaWAN-Based Module for Pulse Output Water Meter. In *Recent Trends in Computational Intelligence and Its Application*; CRC Press: Boca Raton, FL, USA, 2023; pp. 201–216. [[CrossRef](#)]
20. Slany, V.; Ourednicek, V.; Svacina, O.; Svec, J.; Svoboda, M.; Erban, T. New Hybrid IoT LoRaWAN/IRC Sensors: Smart Water Metering System. *Comput. Mater. Contin.* **2022**, *71*, 5201–5217. [[CrossRef](#)]
21. Ismail, S.; Dawoud, D.; Ismail, N.; Marsh, R.; Alshami, A. IoT-Based Water Management Systems: Survey and Future Research Direction. *IEEE Access* **2022**, *10*, 35942–35952. [[CrossRef](#)]
22. Lall, A.; Vaid, K.; Yadav, R.; Sharma, K.; Varma, R.; Chaurasia, P. Making Analog Water Meter Smart Using ML and IoT-Based Low-Cost Retrofitting. In Proceedings of the 2021 8th International Conference on Future Internet of Things and Cloud (FiCloud), Virtual, 23–25 August 2021; pp. 157–162. [[CrossRef](#)]
23. Alvisi, S.; Franchini, M.; Marinelli, A.; Sarni, S.; Zecchin, A. Wireless Middleware Solutions for Smart Water Metering. *Sensors* **2019**, *19*, 1853. [[CrossRef](#)]
24. Pimenta, N.; Chaves, P. Study and Design of a Retrofitted Smart Water Meter Solution with Energy Harvesting Integration. *Discov. Internet Things* **2021**, *1*, 1–15. [[CrossRef](#)]
25. Herath, I. Smart Water Buddy: IoT Based Intelligent Domestic Water Management System. In Proceedings of the 2019 International Conference on Advancements in Computing (ICAC), Malabe, Sri Lanka, 5–7 December 2019; pp. 380–385. [[CrossRef](#)]
26. Arsene, D.; Craciunescu, A.; Dogariu, C.; Savulescu, I.; Bejan, L. Advanced Strategies for Monitoring Water Consumption Patterns in Households Based on IoT and Machine Learning. *Water* **2022**, *14*, 2187. [[CrossRef](#)]
27. Fuentes, H.; Mauricio, D. Smart Water Consumption Measurement System for Houses Using IoT and Cloud Computing. *Environ. Monit. Assess.* **2020**, *192*, 602. [[CrossRef](#)]
28. Laroca, R.; Silva, J.; Baldo, F.; Britto, A.; Minetto, R. Towards Image-Based Automatic Meter Reading in Unconstrained Scenarios: A Robust and Efficient Approach. *IEEE Access* **2021**, *9*, 67569–67584. [[CrossRef](#)]
29. Liang, Y.; Liao, Y.; Li, S.; Zhang, Y.; Peng, S.; Tang, Z. Research on Water Meter Reading Recognition Based on Deep Learning. *Sci. Rep.* **2022**, *12*, 12861. [[CrossRef](#)]
30. Ktari, J.; Frikha, T.; Hamdi, M.; Elmannai, H.; Hmam, H. Lightweight AI Framework for Industry 4.0 Case Study: Water Meter Recognition. *Big Data Cogn. Comput.* **2022**, *6*, 72. [[CrossRef](#)]

31. Bawankar, N.; Kriti, A.; Chouhan, S.; Chaudhari, S. IoT-Enabled Water Monitoring in Smart Cities With Retrofit and Solar-Based Energy Harvesting. *IEEE Access* **2024**, *12*, 58222–58238. [[CrossRef](#)]
32. Lall, A.; Terala, A.; Goyal, A.; Chaudhari, S.; Rajan, K.; Chouhan, S. Behavioural Analysis of Water Consumption Using IoT-Based Smart Retrofit Meter. *IEEE Access* **2024**, *12*, 113597–113607. [[CrossRef](#)]
33. Zhao, S.; Lu, Q.; Zhang, C.; Ahn, C.; Chen, K. Effective Recognition of Word-Wheel Water Meter Readings for Smart Urban Infrastructure. *IEEE Internet Things J.* **2024**, *11*, 17283–17291. [[CrossRef](#)]
34. Naim, A.; Aaroud, A.; Akodadi, K.; El Hachimi, C. A Fully AI-Based System to Automate Water Meter Data Collection in Morocco Country. *Array* **2021**, *10*, 100056. [[CrossRef](#)]
35. Vitiello, S.; Andreadou, N.; Ardelean, M.; Fulli, G. Smart Metering Roll-Out in Europe: Where Do We Stand? Cost Benefit Analyses in the Clean Energy Package and Research Trends in the Green Deal. *Energies* **2022**, *15*, 2340. [[CrossRef](#)]
36. Laroca, R.; Barroso, V.; Diniz, M.; Gonçalves, G.; Schwartz, W.; Menotti, D. Convolutional Neural Networks for Automatic Meter Reading. *J. Electron. Imag.* **2019**, *28*, 013023. [[CrossRef](#)]
37. Nigar, N.; Faisal, M.; Shahzad, H.; Muhammad, I.; Shahid, M.; Oki, O. An Offline Image Auditing System for Legacy Meter Reading Systems in Developing Countries: A Machine Learning Approach. *J. Electr. Comput. Eng.* **2022**, *2022*, 4543530. [[CrossRef](#)]
38. Martinelli, F.; Mercaldo, F.; Santone, A. Water Meter Reading for Smart Grid Monitoring. *Sensors* **2023**, *23*, 75. [[CrossRef](#)]
39. Howard, A.G. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
40. Brockmann, S.; Schlippe, T. Optimizing Convolutional Neural Networks for Image Classification on Resource-Constrained Microcontroller Units. *Computers* **2024**, *13*, 173. [[CrossRef](#)]
41. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In Proceedings of the Computer Vision—ECCV 2018, Munich, Germany, 8–14 September 2018; Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y., Eds.; Springer: Cham, Switzerland, 2018; pp. 122–138.
42. CNN Analyzer. Available online: https://github.com/subrockmann/tiny_cnn (accessed on 20 October 2024).
43. Khan, L.U.; Saad, W.; Han, Z.; Hossain, E.; Hong, C.S. Federated Learning for Internet of Things: Recent Advances, Taxonomy, and Open Challenges. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1759–1799. [[CrossRef](#)]
44. Llisterri Giménez, N.; Monfort Grau, M.; Pueyo Centelles, R.; Freitag, F. On-Device Training of Machine Learning Models on Microcontrollers with Federated Learning. *Electronics* **2022**, *11*, 573. [[CrossRef](#)]
45. Duttgupta, A.; Zhao, J.; Shreejith, S. Exploring Lightweight Federated Learning for Distributed Load Forecasting. In Proceedings of the 2023 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), Glasgow, UK, 31 October–3 November 2023; pp. 1–6. [[CrossRef](#)]
46. Ficco, M.; Guerriero, A.; Milite, E.; Palmieri, F.; Pietrantuono, R.; Russo, S. Federated learning for IoT devices: Enhancing TinyML with on-board training. *Inf. Fusion* **2024**, *104*, 102189. [[CrossRef](#)]
47. ESP32-CAM Camera Development Board. Available online: <https://docs.ai-thinker.com/esp32-cam> (accessed on 28 September 2024).
48. Salomon, G.; Laroca, R.; Menotti, D. Deep learning for image-based automatic dial meter reading: Dataset and baselines. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, Glasgow, UK, 19–24 July 2020; pp. 1–8.
49. Introduction to Low Power Mode for Systemic Power Management. Available online: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/low-power-mode/low-power-mode-soc.html> (accessed on 20 October 2024).
50. Lukic, M.; Sobot, S.; Mezei, I.; Vukobratovic, D.; Danilovic, D. In-depth Real-World Evaluation of NB-IoT Module Energy Consumption. In Proceedings of the 2020 IEEE International Conference on Smart Internet of Things (SmartIoT), Beijing, China, 14–16 August 2020; pp. 261–265. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.