International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 15, No. 3, November 2026, pp. 629∼644
ISSN: 2089-4864, DOI: 10.11591/ijres.v15.i3.pp629-644

Image processing and computer vision on low-cost microcontrollers: a case study on water meter reading

Sohrab Jalali[1], Filiz Ersöz[2], Babak Taeb[3]
[1]Department of Software Engineering, Graduate School of Natural and Applied Sciences, OSTİM Technical University, Ankara, Turkey
[2]Department of Data Science and Analytics, Faculty of Economics and Administrative Sciences, OSTİM Technical University, Ankara, Turkey
[3]Independent Researcher, Ankara, Turkey

Article Info

Article history:

Received Jun 16, 2026
Revised Aug 14, 2026
Accepted Sep 11, 2026

Keywords:

Adaptive sampling
Anomaly detection
Embedded computer vision
ESP32-CAM
INT8 quantization
TinyML
Water meter reading

ABSTRACT

Despite the worldwide move towards smart meters, mechanical water meters are still used extensively in residential and municipal networks. Upgrading them with computer vision avoids costly infrastructure replacement, yet most approaches send data to cloud servers for processing. This paper introduces an end-to-end on-device solution for recognizing water meter digits on the ESP32-CAM board, which costs below $15 and features only 520 KB SRAM. A MobileNetV2-based digit classifier is quantized to INT8. Three firmware improvements extend the inference loop: i) a temporal check confirming a reading only when three consecutive detections match, ii) on-device anomaly detection without server support, and iii) adaptive sampling between 3 s during water usage and 60 s when idle. A Flask-based internet of things (IoT) dashboard named MeterWatch provides live monitoring. The model achieves 2 ms latency and 235.6 KB peak memory. On a 553-image test subset from a custom 6,155-image dataset with 13,470 digit annotations it reaches 99.60% cell-level accuracy including background, 96.89% digit-only accuracy and 87.70% exact five-digit reading accuracy. To the best of our knowledge, this is the first system combining on-device INT8 deployment, water meter digit recognition and intelligent firmware behaviour on the ESP32-CAM. Evaluations used a single meter brand under fixed flash illumination.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Sohrab Jalali
Department of Software Engineering, Graduate School of Natural and Applied Sciences
OSTİM Technical University
Ankara, Turkey
Email: sohrabjallali@gmail.com

1. INTRODUCTION

Edge artificial intelligence (AI) in conjunction with cheap embedded devices has made a new form of intelligence possible to deploy. One of the most advanced devices in this category is the ESP32-CAM which comprises a dual-core system on chip (SoC) device along with an OV2640 camera and 802.11 b/g/n Wi-Fi capability. With its cheap price of less than $15 per unit, this platform represents a good replacement for energy intensive single-board computers like Raspberry Pi and NVIDIA Jetson for massive sensing purposes. It can even be used inside already-existing enclosures at city scale.

One area that is poised to see great gains is water utilities. The traditional mechanical meter is by far the most common device for measuring water consumption in both private homes and city water systems

around the world. However, they still require manual meter reading in almost all cases. This procedure is both inefficient and costly not to mention that it is subject to error from human transcription [1]. Upgrading all of these meters to contemporary digital meters is a financially prohibitive proposition, costing many millions of dollars. A less costly solution involves retrofitting these meters with computer vision capabilities that capture and interpret the digit display in real time [2], [3].

Implementing deep learning on a microcontroller presents several challenges. The regular object detection algorithms consume hundreds of megabytes of memory, alongside the need for GPU or NEON optimization, neither of which is available on the ESP32 board. Apart from the memory requirements, the two core processors on the ESP32 work at a maximum speed of 240 MHz, which is enough for an embedded device application but way slower than the computing power required for a typical inference algorithm. Achieving practical accuracy on real-world meter images which exhibit variable lighting, angular distortion, glare, dirt, and surface wear needs both careful model design and rigorous compression [4], [5]. The evaluation reported here isolates the modeling and compression problem under a controlled fixed-illumination deployment. Generalisation to uncontrolled lighting remains future work.

The proposed paper provides an integrated approach toward solving the above problem. Some of the main points include the following:

- An INT8-quantized MobileNetV2-based digit classifier running entirely on the ESP32-CAM over a fixed 12×12 grid, achieving 2 ms model inference latency, 235.6 KB peak RAM, a 99.60% overall cell-level accuracy including the background class, and a 96.89% digit-only classification accuracy on a 553-image box-annotated test subset.
- A temporal consistency technique where the rolling buffer consists of three readings and any reading is considered validated only if three consecutive readings match, filtering transient errors and sensor noise.
- On-device anomaly detection using firmware-level arithmetic on confirmed readings to flag abnormal consumption events such as pipe leaks, without any server involvement.
- Adaptive sampling: a time-interval-based system that changes the interval length from 60 seconds when the system is idle to 3 seconds when there is any activity. It helps in saving cycles of inference.
- MeterWatch: an internet of things (IoT) dashboard developed using Flask technology which supports live JPEG images with bounding boxes, trend graphs using Chart.js, anomalies, multi-device capability, and session statistics.
- A custom dataset of 6,155 water meter images, including 2,694 images with digit-level bounding box annotations totalling 13,470 boxes, obtained under real-world scenarios.

2. RELATED WORK

2.1. Water meter and utility meter reading

Different approaches have been taken to research automatic water meter reading. Initially, approaches were based on traditional image processing techniques such as thresholding, morphological processing, and template matching. However, these approaches suffer from poor robustness due to factors such as varying lighting conditions and degraded surfaces [1]. Deep learning models proved more effective. In particular, Nguyen *et al.* [2] utilized text recognition techniques along with a deep learning approach. Their model demonstrates competitive performance when applied to a large dataset of meters. The only drawback is that the system uses a GPU server for inference. Zhang *et al.* [3] implemented an attention model at a character level in order to overcome partially occluded or blurred digits. They also apply inference on a GPU server, which is used on cloud computing platforms. Hong *et al.* [4] tested their meter reader model in deployment settings characterized by heavy fouling, low lighting, and severe viewing angles and found that data augmentation significantly enhances accuracy. Their system is executed on workstations. Li *et al.* [5] focused on mechanical water meters and reported 97.9% mAP@0.5 with an improved YOLOv4 detector in natural scenes without the need for embedded deployment. Martinelli *et al.* [6] incorporated meter reading into a larger smart grid monitoring system and used cloud computing infrastructure for inference. Qiao *et al.* [7] developed a compact variant of YOLO named Meter-YOLOv8n that supports word-wheel water meters but still requires server-side deployment.

A major limitation in the survey arises in this context: seven prior studies specifically focus on the task of digit recognition in water meters, of which only one (Neve [8]) targets the ESP32-CAM. Nevertheless, the work of Neve [8] lacks INT8 quantization, any memory profiling, or anomaly and adaptive capabilities. This limitation stems from a technical perspective. First, due to the limited RAM of the ESP32 (520 KB), the stan-

dard format of a machine learning model cannot be used, second, due to the requirement of proper calibration in case of TFLite Micro, and third, due to the need for an annotated dataset suitable for the target meter and deployment conditions. Beyond these gaps, Neve's system relies on manually defined regions of interest for each digit position, configured once through a calibration interface and fixed thereafter, rather than a learned localisation or classification mechanism. The captured images are also described as being sent over Wi-Fi for processing, with the actual location of inference (on-device or server-side) left unspecified. No accuracy, memory footprint, or latency figures are reported, which limits direct quantitative comparison (see subsection 4.7). Our work addresses precisely this gap. Table 1 summarises the comparison.

Table 1. Water meter reading studies: platform summary (7 prior works and this work, 6 of the 7 prior works rely on server/cloud inference)

Reference	Platform	On-device	INT8
Neve [8]	ESP32-CAM	Unspecified	No
Nguyen *et al.* [2]	DL server	No	N/A
Zhang *et al.* [3]	CNN server	No	N/A
Hong *et al.* [4]	ML server	No	N/A
Li *et al.* [5]	CNN server	No	N/A
Martinelli *et al.* [6]	IoT + cloud	No	N/A
Qiao *et al.* [7]	YOLO server	No	N/A
This work	**ESP32-CAM**	**Yes**	**Yes**

2.2. On-device inference on ESP32

There are some papers related to on-chip inference in ESP32. However, very few works successfully deploy an end-to-end embedded solution. The research paper most relevant to our case is that of Ben Aicha [9], who conducts INT8-quantized CNN inference on ESP32-CAM for generic image classification. The paper shows how effective the INT8 post-training quantization is in improving the accuracy of the model after the training process. It proves the possibility of performing real-time embedded inference under the memory limitation of the ESP32. Nonetheless, the paper does not consider the cases of water-metering reading and anomaly detection nor adaptive system behavior.

Sineglazov and Khotsyanovsky [10] showed that the implementation of the entire convolutional neural network architecture was possible right on the ESP32 without any pre-processing steps performed externally. The work of Dokic [11] took a simpler approach through using logistic regression on low-resolution images captured by the ESP32 camera. On the other hand, Paul *et al.* [12] applied regression algorithms in embedded systems for the task of reading analog pressure gauges in a laboratory environment.

However, current research using ESP32-CAM mostly involves taking the pictures locally, after which inference is done off-device. According to Narwaria *et al.* [13] and Sulistiyowati *et al.* [14], local picture capture is followed by sending the captured images to a PC for processing with YOLO or OpenCV. Hafidz *et al.* [15] managed to integrate YOLO with ESP32-CAM for stereoscopic depth estimation, where inference is carried out off-device too. In addition, Nowroz *et al.* [16] and Imron *et al.* [17] conducted inference on a cloud-based back-end system. Setiawan *et al.* [18] combined YOLOv8 with ESP32-CAM capture but processed detections server-side. Chang *et al.* [19] carried out object detection using ESP32 edge computing and Nevliudov *et al.* [20] conducted object recognition with a microcontroller-based robot. The aforementioned off-device processing represents the most widely used methodology in ESP32-CAM research, whereas the present research represents only one of the few exceptions from that rule.

2.3. TinyML and model compression

The basis of theory for INT8 quantization is laid by Nagel *et al.* [21] as they analyze the roots of quantization error and develop calibration methods to counteract such negative effects. Theory into practice was achieved through research done by Novac *et al.* [22] whereby they implemented quantized deep neural networks on Cortex-M microcontrollers and analyzed the trade-off curve of accuracy versus memory footprint at different bit-widths. Finally, Shabir *et al.* [23] conduct an analysis comparing pruning against quantization methods on TinyML applications concluding that post-training INT8 quantization is optimal for accuracy and efficiency trade-off.

TinyissimoYOLO was introduced by Moosmann *et al.* [24], which is a fully quantized YOLO variant designed specifically for ultra-low-memory MCUs, reporting sub-1 MB memory operation on ARM Cortex-M class hardware. Further progress in this direction was made by Boyle *et al.* [25] with DSORT-MCU, focusing on developing methods that would enable small object detection on microcontroller units. In addition, Zhang [26] examined quantization-aware training (QAT) as a substitute approach compared to post-training quantization in embedded artificial intelligence systems. MCUNetV2 was developed by Lin *et al.* [27], where inference was done using patches in order to reduce peak activation memory requirements down to 256 KB.

2.4. Embedded internet of things and anomaly detection

Bawankar *et al.* [28] developed a retrofit water-monitoring system that recognizes meter digits from camera images on a Raspberry Pi Zero W edge device and transmits the readings to the cloud for further analysis. Our anomaly detection algorithm is quite unique in nature since it detects abnormalities through vision-based meter readings and completes all the processing logic via ESP32 firmware. Pawar *et al.* [29] conducted a survey on smart energy meters using ESP32 to justify its applicability in utility IoT. In addition, Samanta *et al.* [30] used TinyML in combination with ESP32-CAM for disaster location identification in the real world proving the effectiveness of on-device inference under real deployment constraints. Lastly, Donskoy *et al.* [31] adopted TinyML on the ESP32 for agricultural classification to show that embedded models can maintain reliable accuracy across diverse real-world environments.

3. METHOD

3.1. System overview

Figure 1 shows the pipeline overview. In the first step, the ESP32-CAM captures a JPEG image and then performs its pre-processing and INT8 inference completely onboard before applying the three firmware contributions and sending the output data to MeterWatch via Wi-Fi. The pre-processing procedure, which involves transforming from 320×240 JPEG to 96×96 grayscale, is done on-device by decoding the JPEG format and performing bilinear downsampling, adding approximately 8–12 ms to the pipeline latency. Model inference itself accounts for only 2 ms. Only the final POST request leaves the device.

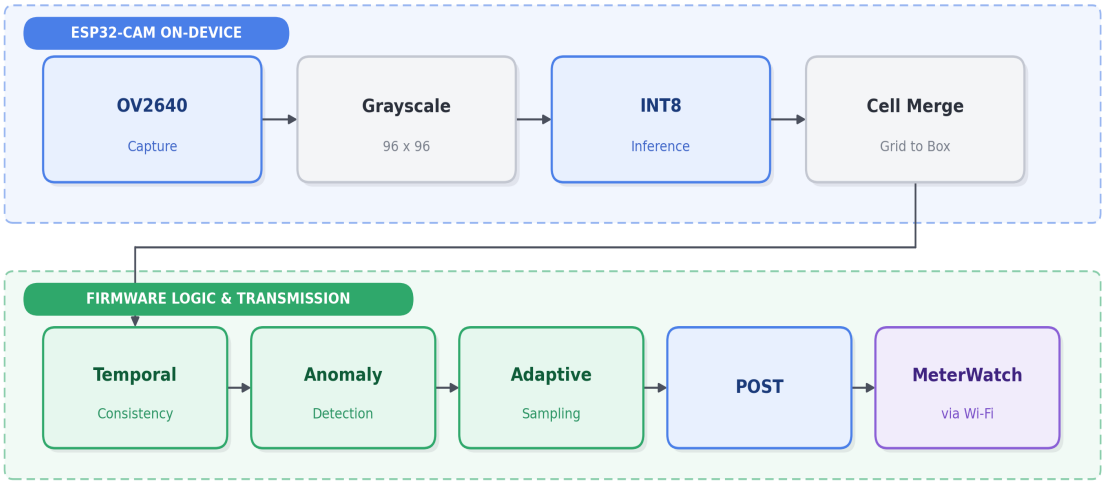


Figure 1. End-to-end system pipeline with all inference and firmware logic executed on the ESP32-CAM

3.2. Hardware platform

The ESP32-CAM board runs on Espressif's ESP32 chip, which has two Xtensa LX6 cores that can hit speeds up to 240 MHz. It packs 520 KB of SRAM, 4 MB of PSRAM and 4 MB of Flash memory, along with a built-in OV2640 camera. Wi-Fi 802.11 b/g/n is supported natively without additional hardware. Priced at under $15 per unit, it is an affordable option for large-scale infrastructure deployment.

3.3. Dataset collection and annotation

The images were captured using a self-designed and manufactured 3D-printed data acquisition device (Figure 2). The device consists of a rigid structure that includes a base with rail on which there are four vertically mounted guide rods to allow movement of the camera mount in all three dimensions. While vertical adjustment is done by moving the camera mount up and down the rods, independent horizontal adjustments can be made by varying the positions of thumb-screw-mounted cameras. This flexibility enabled systematic capture of images across a controlled range of distances and angles.



Figure 2. Custom 3D-printed data acquisition rig with a camera mount repositionable in all three axes

A customized dataset was created using the image capturing of the Baylan-brand mechanical water meter across a range of camera-to-meter distances and observation angles up to $\sim$30° off-axis, with some images exhibiting surface degradation including glare, dirt, and scratches. Illumination was held constant across all captures via the rig's integrated flash LED, matching the illumination used in the production enclosure. Only distance and viewing angle varied systematically. All images were captured at 320×240 pixels in JPEG format.

Deployment enclosure: as for the production deployment, the ESP32-CAM is housed in a sealed 3D-printed enclosure that sits directly over the face of the meter. This ensures that any form of ambient light interference is eliminated while providing a consistent illumination environment through the integrated flash LED of the OV2640 sensor. Controlled illumination is one of the key factors responsible for the high reported accuracy (99.60%), as the model operates under conditions closely matching those of data collection: no uncontrolled light sources, since the flash-illuminated capture regime is identical between data collection and production. The camera-to-meter distance and viewing angle are fixed at 8 cm and 0 degrees off-axis specifically in the production enclosure, whereas the training and test data span the wider 5–12 cm / 0–30 degree range listed in Table 2, since the train-test split was performed at image level rather than by condition (see subsection 4.1).

Table 2. Data collection rig parameters

Parameter	Value/range
Camera-to-meter distance	5–12 cm (production: 8 cm)
Horizontal viewing angle	0°–30° off-axis
Vertical tilt angle	0°–15° off-axis
Illumination source	Integrated OV2640 flash LED
Capture resolution	320×240 JPEG
Meter brand	Baylan mechanical (single)

Total number of images in this dataset is 6,155, where 4,961 of them represent the training set while 1,194 images form the testing set (80%/20% ratio). Of these, 2,694 images carry digit-level bounding box annotations (2,141 train, 553 test), each with all five digits individually marked, yielding a total of 13,470 bounding box annotations. Of these annotations, 9,622 were hand-drawn and 3,848 were generated via an automatic suggestion mechanism based on a hand-drawn reference box in the same image, subsequently reviewed and corrected by hand.

The remaining 3,461 images (2,820 train, 641 test) contain no digit-level boxes and no ground-truth label of any kind, including no full meter reading value. The label field for these files carries only a file identifier. Consequently these images were not used in model training: the effective training sample size is limited to the 2,141 box-annotated training images, for which cell-level ground truth could be generated. The 641 box-unlabelled test images were run through inference after training solely to observe whether the model produced a structurally complete five-digit output. In the absence of ground truth this cannot be scored for correctness. 551 of the 641 images (86.0%) yielded a complete five-digit output group, which is a structural-completeness indicator only and not a reading-accuracy metric.

Digit widths range from 25 to 41 pixels and heights from 37 to 61 pixels. The distribution of the number of instances per class shows an imbalance, where the most common class is "3" with 1,965 instances (14.6%), and the least common are "1" and "6" with 1,158 instances each (8.6%). Pre-processing steps applied before training included centre-cropping to the digit region, grayscale conversion, and normalisation to a 96×96 pixel input. Augmentation included random rotation ($\pm15°$), zoom (0.85–1.15$\times$), and Gaussian blur.

3.4. Model architecture selection

The choice of an architecture was limited to the fact that the ESP32 only has 520 KB of SRAM memory. Most OCR libraries (such as Tesseract or EasyOCR) and object detection algorithms (including YOLO, Faster R-CNN, or variations of SSDs) consume hundreds of megabytes of memory, and require x86 or ARM processors with dedicated NEON or CUDA acceleration, hardware entirely absent on the ESP32. In the context of suitable architectures for embedded devices, MobileNetV2 was selected because depthwise separable convolutions allow for fewer multiply-accumulate operations while still providing sufficient representational capacity to recognize digits among ten classes. Compared to MobileNetV1, MobileNetV2 uses an inverted residual structure, which yields better gradient flow and lower activation memory at equivalent parameter counts. The final INT8-quantized model fits within 235.6 KB peak RAM, less than half the available budget.

3.5. Model architecture and training

The digit recognition model is made up of a backbone inspired by MobileNetV2's inverted-residual block structure, significantly narrowed relative to the standard architecture, combined with a dense classification head, accepting 96×96 grayscale input. Depthwise separable convolutions keep the multiply-accumulate count and parameter total within the ESP32's memory budget. The backbone reduces the input spatially to a fixed 12×12 grid of cells, each covering an 8×8-pixel region of the input. Each cell is independently classified via softmax into eleven categories: background or one of ten digit classes. No bounding box coordinates are regressed.

At inference, adjacent cells sharing the same predicted digit class above the confidence threshold are geometrically merged into a single bounding box for display purposes, without any learned localisation component. This merging follows a simple axis-aligned-box (AABB) adjacency test rather than an intersection over union (IoU) threshold: the grid is scanned, each positive cell starts or extends a group, and cells are joined to a group when their boundaries touch or overlap, with a final merge pass across all groups. Cells of different classes never merge into the same group because each cell carries a single softmax-argmax label.

Training labels were generated from the digit-level bounding boxes described in subsection 3.3 as follows: the centre (cx, cy) of each box was computed and mapped to a single grid cell via floor division by the 8-pixel cell stride, cell_x = floor(cx/8) and cell_y = floor(cy/8). Only that single centre cell was assigned the corresponding digit label. Other cells geometrically overlapped by the same box were not separately labelled. All remaining cells defaulted to the background class with no additional rule. No IoU-based overlap threshold was used. The cell-label assignment depends solely on the centre point. Because adjacent digits in a five-digit reading are spaced well beyond one cell width apart, no two digits' centres were observed to fall within the same cell in this dataset.

Training was carried out using cloud GPU infrastructure without ImageNet pretraining. All layers were trained from scratch, starting from a learning rate of 1e-4, with ReduceLROnPlateau (patience=5, factor=0.5) halving the learning rate whenever validation loss failed to improve for five epochs. Training was bounded by EarlyStopping (patience=10, restore_best_weights=True) with a maximum of 1,000 epochs and a batch size of 32. The loss function is a per-cell weighted categorical cross-entropy, with the weight for each digit class computed as a fixed foreground boost factor (15.0) multiplied by the ratio of the median class frequency to that class's frequency. The background class weight is fixed at 1.0. A fixed random seed (1337) was used for the train/validation split and data shuffling. No seed was fixed for in-training-loop randomness such as weight initialisation or batch ordering, so the training procedure itself is not fully deterministic. The development environment used TensorFlow 2.21.0, though the exact version used for the original training run could not be confirmed from the code base. After float-precision training, a dedicated calibration subset of 200 images, collected separately from the training set and not drawn from the training split, was used to derive INT8 activation ranges via representative dataset calibration.

3.6. INT8 quantization

Post-training INT8 quantization converts all weights and activations from FP32 to 8-bit integer representation, following Nagel *et al.* [21] as validated for MCU deployment by Novac *et al.* [22]. The Float32 model was never executed on the ESP32. At 887.2 KB it exceeds the board's 520 KB SRAM budget. Its 98.54% accuracy figure was obtained from a generic evaluation-function output in the training environment (PC/GPU, TensorFlow), not from the detailed 553-image box-annotated confusion-matrix analysis used for the INT8 model in subsection 4.2. Because the two figures derive from different evaluation methodologies, no class-level F1-score is available for Float32 and the two accuracy values are not directly comparable. The compiled INT8 TFLite binary occupies approximately 142 KB (141,952 bytes) on disk as a flatbuffer. The EON compiler converts this into raw C++ arrays and direct function calls, stripping the flatbuffer schema, reflection, and operator-code-table overhead, which is why the resulting 94.8 KB on-device binary is smaller than the source flatbuffer file. Table 3 summarises the comparison.

Table 3. Float32 vs. INT8 model: key metrics

Metric	Float32	INT8	Change
Peak RAM (KB)	887.2	235.6	3.8× ↓
Flash (KB)	—	94.8	—
Inference (ms)	—	2.04 (min 1.98, max 2.17, SD 0.05, n=100)	—
Accuracy (%)	98.54	99.60	+1.06 pp*
F1-score	—	0.9355	—

*Raw arithmetic difference between values obtained under different evaluation methodologies (see text). Not tested for statistical significance and should not be read as evidence that INT8 improves accuracy.

3.7. Firmware contributions

Three mechanisms were added to the ESP32 firmware on top of the basic inference loop.

3.7.1. Temporal consistency check

The firmware maintains a rolling buffer of the three most recent successful readings. A reading is labelled *confirmed* only when all three entries are identical. This eliminates single-frame errors, sensor noise, and motion artefacts. Each transmission to the server carries a `confirmed` flag so the dashboard can distinguish verified readings from tentative ones.

3.7.2. On-device anomaly detection

The firmware maintains a reference value that is set at the last reset point, either the previous anomaly event or the confirmed reading at session start, and does not update on every reading. After each confirmed reading, the firmware computes the cumulative difference between the new value and this fixed reference. If this difference exceeds a configurable threshold (default: ten units), the reading is classified as an anomaly, the anomaly flag is set in the transmission payload, and the reference value is reset to the current reading. Because the reference only resets on an anomaly event rather than every cycle, the same rule catches both sudden usage spikes and slow leaks that accumulate gradually across many reading cycles. This enables detection of pipe leaks or unexpected high-usage events entirely on the microcontroller, without any server-side involvement.

The default threshold of ten units was selected empirically based on the maximum observed cumulative consumption increment during normal usage on the test meter. It is exposed as a compile-time constant to allow operators to calibrate it for meters with different flow profiles or usage patterns. The approach is fundamentally different from Bawankar *et al.* [28], who transmit vision-derived readings to the cloud for further analysis.

3.7.3. Adaptive sampling

The reading interval is adjusted dynamically. When the units digit of the meter value stays unchanged across consecutive readings, the interval increases by 5 s per cycle up to a ceiling of 60 s. When the units digit changes, indicating active water flow, the interval immediately drops to 5 s, then decreases by 2 s per cycle down to a floor of 3 s. An anomaly event immediately resets to the minimum interval to enable rapid consecutive readings. This policy cuts unnecessary inference cycles during idle periods without sacrificing responsiveness.

3.8. MeterWatch internet of things dashboard

MeterWatch is a Flask web application that serves as the monitoring front-end for the system. Figure 3 shows the complete hardware–software architecture connecting the ESP32-CAM to the MeterWatch dashboard. On each HTTP POST from the ESP32-CAM, the server receives the raw JPEG image, the recognised meter value, merged-cell bounding box coordinates for display overlay, and the three firmware flags (`confirmed`, `anomaly`, `sampling_interval`). It draws bounding box overlays on the received image and stores all data in memory.

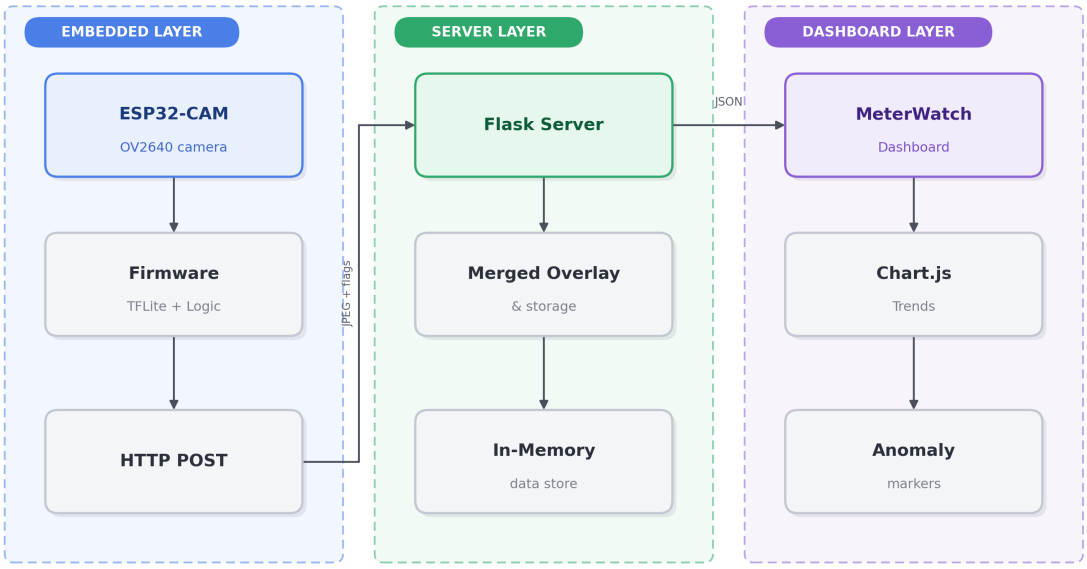


Figure 3. System architecture connecting the ESP32-CAM, the Flask server, and the MeterWatch dashboard

The dashboard frontend, refreshed via AJAX polling every 3 s, provides: a live image grid of the ten most recent unique readings with overlays. A Chart.js trend graph of the last 20 readings with anomaly markers highlighted in red. A connected device panel supporting multiple simultaneous ESP32-CAM units. Real-time session statistics (confirmed reading rate, anomaly rate, average sampling interval, system success rate). Also a full reading history table with per-row highlighting for anomalies, confirmed readings, and error states. Figure 4 shows the MeterWatch web interface itself, illustrating the live reading grid, connected-device panel, session statistics, trend chart, and reading history table described above. The device count, reading total and session statistics in this screenshot demonstrate interface functionality only and do not represent the single-meter experimental results reported in section 4.

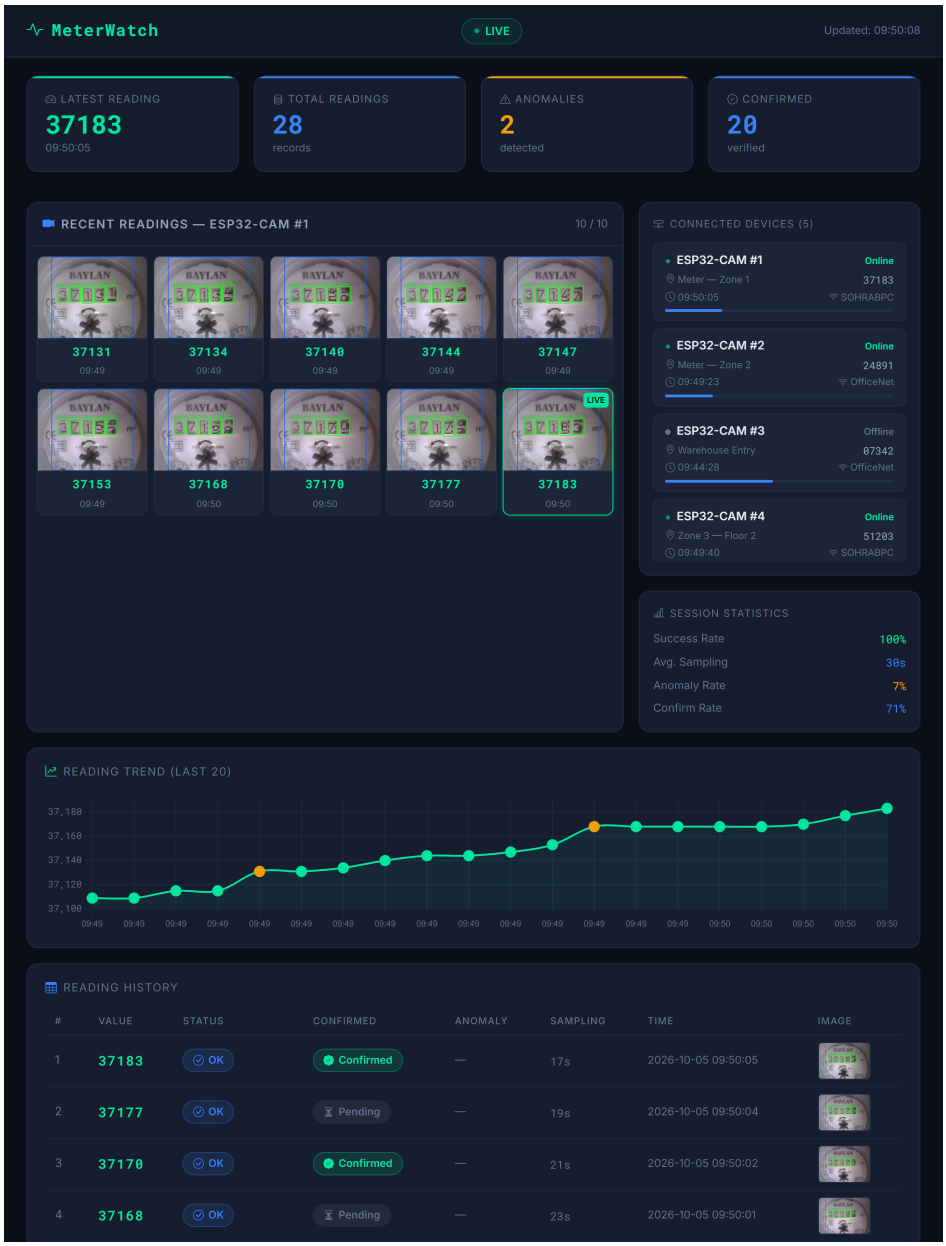


Figure 4. MeterWatch dashboard interface (illustrative screenshot)

4. RESULTS AND DISCUSSION

4.1. Experimental protocol

All metrics were collected on a physical Baylan-brand mechanical water meter under the same real-world conditions used during data collection. The full 6,155-image dataset was collected across approximately 6–7 distinct capture sessions. The test images were excluded from model training and INT8 calibration. However, because the split was performed at image level rather than acquisition-session level, the evaluation should be interpreted as an internal frame-level validation rather than an independent session-level test. The train-test split was performed by random image-level partitioning across the complete capture pool rather than by session-level separation. Test images may therefore share capture sessions, lighting setups, and meter positions with training images, differing at the frame level rather than representing fully independent acquisition conditions. This should be taken into account when interpreting the reported accuracy.

4.2. Classification performance

The INT8 model was evaluated on the 553-image box-annotated test subset, comprising 79,632 individually labelled grid cells (2,765 true digit cells and 76,867 true background cells) at a fixed classification confidence threshold of 0.5. The model achieves a 99.60% overall cell-level accuracy including the background class (79,310 of 79,632 cells correctly classified). Because background cells make up 96.5% of all evaluated cells and are classified almost without error (99.69% recall), this figure is dominated by the background class and should not be read as a measure of digit-recognition difficulty. Restricting the evaluation to the 2,765 true digit cells gives a 96.89% digit-only micro accuracy (2,679 of 2,765). Macro precision across the ten digit classes is 90.44%, macro recall is 96.92%, and macro F1 is 0.9355. Class-level recall ranges from 93.89% (digit "6", the class most frequently confused with digits "5" and "7") to 99.16% (digit "2"). Class-level F1 ranges from 0.9142 (digit "8") to 0.9631 (digit "3"). These figures derive from a single Baylan-brand meter and an image-level split across approximately 6–7 capture sessions. Multi-session and cross-brand validation are needed before treating this accuracy as a general performance bound. Complete five-digit meter readings were correctly recognised in 485 of the 553 box-annotated test images, corresponding to an exact-reading accuracy of 87.70%.

Figure 5 shows the row-normalised confusion matrix on the 553-image test subset (79,632 grid cells) with per-class recall on the diagonal. The dominant confusion sources are misclassification of true-class samples as background, reaching 3.45% for digit "8" and 2.62% for digit "6", and genuine inter-digit confusion, most notably digit "6" being misread as digit "5" (1.75%) and digit "7" (1.31%), and digit "9" being misread as digit "8" (2.25%). Four digit-pair confusions exceed 1% of the corresponding true class: "0"→"9" (1.17%), "6"→"5" (1.75%), "6"→"7" (1.31%), and "9"→"8" (2.25%). Background misclassification is therefore not the sole error source. Figure 6 shows the per-class F1 distribution.

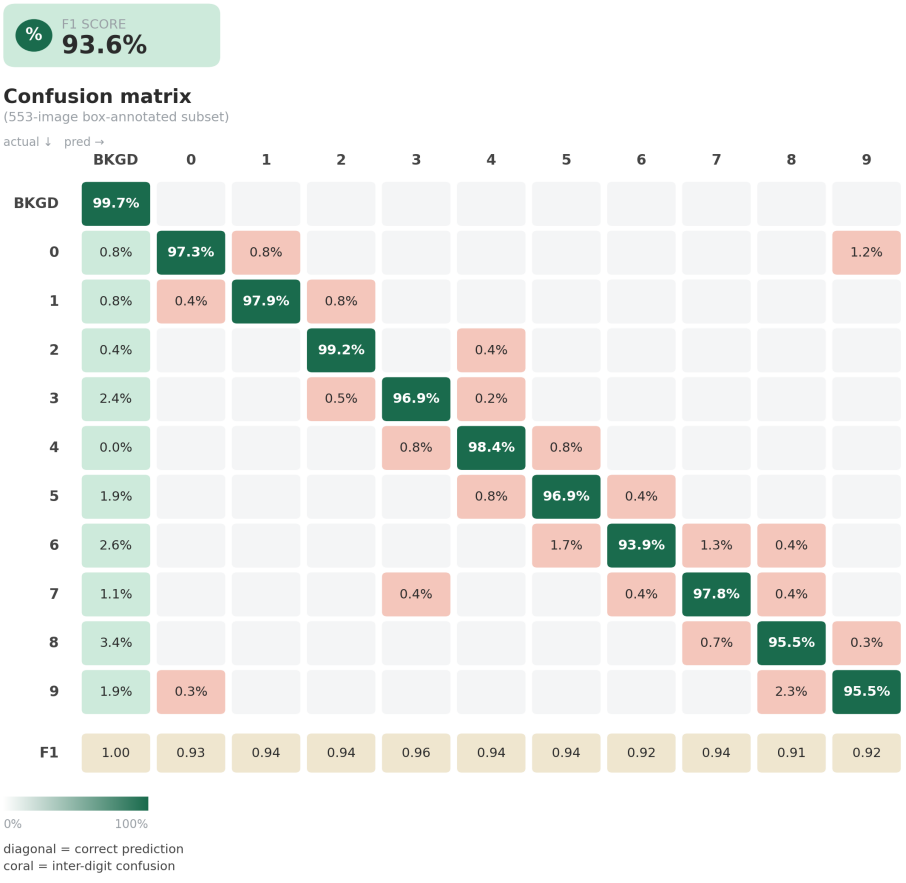


Figure 5. Row-normalised confusion matrix of the INT8 model on the 553-image box-annotated test subset

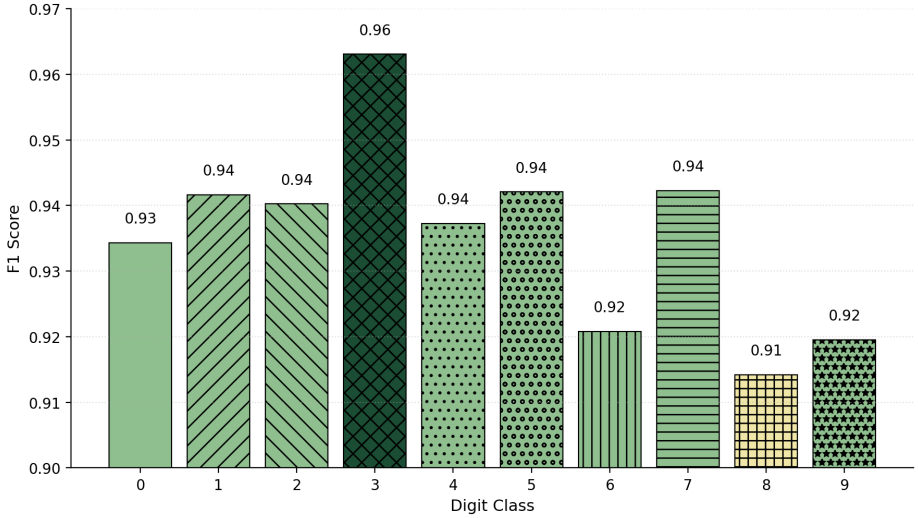


Figure 6. Per-class F1-score of the INT8 model across the ten digit classes

4.3. Memory and latency

Peak RAM on the ESP32 is 235.6 KB, less than half of the 520 KB SRAM budget, leaving headroom for the firmware logic and Wi-Fi stack. This figure is not measured live via a heap-query function. It is the static kTensorArenaSize constant (241,254 bytes in the standard build variant) computed by the TFLite Micro memory planner at model export time, representing the model's tensor arena only. The Wi-Fi stack, HTTP server buffers, and camera frame buffer are excluded and occupy separate memory regions. The tensor arena is allocated in internal SRAM via standard heap allocation (ei_calloc to aligned_alloc). PSRAM is not used for this allocation. The camera frame buffer, by contrast, is allocated separately in PSRAM (ps_malloc). Flash usage is 94.8 KB. Model inference latency was measured on-device using the esp_timer_get_time() high-resolution timer over 100 repeated inference calls, covering only the interpreter's Invoke() step: mean 2.04 ms (min 1.98 ms, max 2.17 ms, SD 0.05 ms), orders of magnitude faster than the cloud round-trip latency inherent to server-based systems such as [16]. The full pipeline latency (capture, preprocessing, inference, and firmware logic) is dominated by image capture and JPEG decompression, with model inference itself accounting for only 2 ms of the total cycle.

4.4. Temporal consistency

Under controlled test conditions (500 consecutive frames over 30 minutes on a stationary meter), the temporal consistency mechanism achieved a confirmed reading in 456 of 500 frames (91.2%), a false-positive reading in 4 of 500 frames (0.8%), and left 40 of 500 frames (8.0%) unconfirmed due to buffer incompletion (456+4+40=500). The three-reading buffer eliminates the large majority of single-frame artefacts visible in the raw detection stream. The unconfirmed frames arose from the rolling three-reading window failing to fill within the sampling window, typically at session start or immediately after an anomaly reset, rather than from detection failures or digit substitution errors. This confirmed reading rate was measured in one 30-minute session on one meter. It has not been replicated across sessions or meter units. Because this mechanism checks whether three consecutive readings agree rather than whether they are correct, a digit misclassification that recurs consistently across three consecutive frames under the same capture conditions (for example, one of the inter-digit confusions reported in subsection 4.2) could in principle be confirmed as a valid reading. This is a limitation of consensus-based validation rather than a defect specific to this implementation.

4.5. Anomaly detection

Controlled anomaly scenarios were constructed by simulating two fault types: pipe-leak events, modelled as a continuous slow increment of one unit per reading cycle, and sudden high-usage spikes, modelled as step increases of 15–30 units between consecutive confirmed readings. Fifteen injection trials were conducted across a 60-minute test session (ten leak, five spike). The on-device detector correctly flagged all 15 injected anomalies at the default threshold of ten units (15/15, 100% detection rate). Among 340 anomaly-free confirmed readings in the same session, 10 produced a false anomaly signal (10/340, 2.9%). No server round-trip

was involved in any detection event. It should be noted that these results derive from a single controlled test session with fifteen injected events. While the detection rate reached 100% within this trial, the sample size is insufficient for statistically robust generalisation. A larger-scale evaluation across hundreds of anomaly events and multiple sessions is required to establish confidence intervals and validate the threshold setting.

4.6. Adaptive sampling

During a 30-minute idle test, the average sampling interval stabilised at 52 s, reducing inference calls by ∼94% compared to a fixed 3 s baseline. During active consumption periods the interval recovered to 3–5 s within two sampling cycles. This reduction directly lowers ESP32 CPU utilisation during idle periods, freeing compute cycles for the Wi-Fi stack and firmware logic without any additional hardware cost.

4.7. Comparison with prior work

Table 4 places the proposed system against the two most relevant prior works: Neve [8] (same hardware, water meter task, no INT8) and Ben Aicha [9] (INT8 on ESP32-CAM, general image classification). This system is the only one that combines on-device INT8 inference, water meter reading, anomaly detection, and adaptive sampling. Inference latency for this work refers to model-only execution (2 ms), and processing latency including JPEG decoding and preprocessing (excluding sensor capture) is approximately 10–14 ms.

Table 4. Detailed comparison with most relevant prior work

Feature	Neve [8]	Ben Aicha [9]	This work
Hardware	ESP32-CAM	ESP32-CAM	ESP32-CAM
INT8 quantization	No	Yes	Yes
Water meter task	Yes	No	Yes
Custom dataset	Not rep.	No	Yes (6,155 images)
Temporal consistency	No	No	Yes
On-device anomaly det.	No	No	Yes
Adaptive sampling	No	No	Yes
IoT dashboard	No	No	Yes (MeterWatch)
Inference latency (ms)	Not rep.†	120	**2 (model) / 10–14**
Peak RAM (KB)	Not rep.†	450	**235.6**
Accuracy (%)	Not rep.†	92.3	**99.60**

†Not reported in the original work.

The comparison with Neve [8] is necessarily qualitative for the unreported metrics, since that work does not publish accuracy, memory, or latency figures. The more substantive technical distinction is architectural: Neve's system depends on manually configured, fixed digit regions established through a calibration step, whereas the present system infers digit identity per grid cell through the learned classifier described in subsection 3.5, with digit position implicit in the fixed cell layout rather than requiring per-installation manual calibration.

4.8. Discussion

The primary finding is that post-training INT8 quantization does not produce a measurable degradation in classification accuracy on the water meter digit task. The INT8 model shows a higher accuracy than the Float32 baseline measured under a different evaluation methodology (see Table 3), but because the two values were not obtained under the same test conditions this difference cannot be claimed as a genuine improvement. This absence of degradation is consistent with results reported in [21] and [22], which found that representative calibration data preserves accuracy under INT8 quantization. The 3.8× RAM reduction leaves the ESP32's SRAM budget largely free, providing practical headroom for concurrent firmware tasks and the Wi-Fi stack.

With model inference completing in 2 ms, the system bottleneck shifts to image capture and preprocessing. The three firmware contributions (temporal consistency checks, anomaly arithmetic, and interval adjustment) add negligible runtime overhead, meaning the intelligent behaviours come essentially for free from a latency perspective.

The temporal consistency mechanism introduces a confirmation latency of up to 9 s at the minimum 3 s sampling interval. For a utility metering application where billing resolution is measured in months and consumption data is aggregated over hours, this lag is entirely acceptable. It brings a substantial reliability benefit: the sub-1% false-positive reading rate means the dashboard rarely presents an incorrect value to the operator.

The adaptive sampling result, a 94% reduction in idle inference calls, is primarily a compute-efficiency contribution. Reducing inference from ∼20 calls/min to ∼1.15 calls/min during idle periods substantially lowers the processor's duty cycle, leaving more headroom for the Wi-Fi stack and firmware logic on a resource-constrained microcontroller. The present system is mains-powered via USB, so power-saving and thermal-longevity benefits are not the primary motivation and remain unmeasured. Adaptive sampling is included chiefly to reduce redundant computation.

Re-analysis of the confusion matrix (subsection 4.2) shows that digit "8" has the highest background-misclassification rate (3.45%) and that digit "6" has the lowest overall recall (93.89%), driven primarily by inter-digit confusion with "5" and "7" rather than background rejection. Digits span only 25–41 pixels in width in the 320×240 capture (subsection 3.3) and shrink further after downsampling to 96×96, and this low resolution can reduce discriminability in low-contrast regions, plausibly contributing to both the background misclassification and inter-digit confusion patterns discussed above. Figure 7 illustrates this by showing a representative test frame as captured at 320×240 (Frame A) and as supplied to the model at 96×96 grayscale (Frame B), with boxes marking the approximate five digit regions. A missed digit (background misclassification) triggers a non-confirmed reading rather than a silently incorrect one, preserving system integrity at the cost of confirmation latency. A misread digit that is mistaken for another digit class does not carry this same protection, as discussed in subsection 4.4.

A potential advantage of the approach is its portability across target hardware. Because the model is exported as a standard TFLite INT8 binary, it can in principle be re-targeted to any platform supported by TensorFlow Lite, including Raspberry Pi, NVIDIA Jetson Nano, or other MCU families. Adapting the system to a different meter brand or display style is expected to require only the collection of a few hundred additional images for fine-tuning, since the backbone retains generic edge and pattern representations learned from training on the current dataset. This expectation is grounded in the general behaviour of transfer learning reported in the literature but has not been empirically validated within this study. All reported results derive from a single meter brand (Baylan), a single meter unit, and controlled laboratory conditions with an image-level random train-test split. Cross-brand fine-tuning experiments and field validation are needed before claims of broad commercial or large-scale deployability can be substantiated.

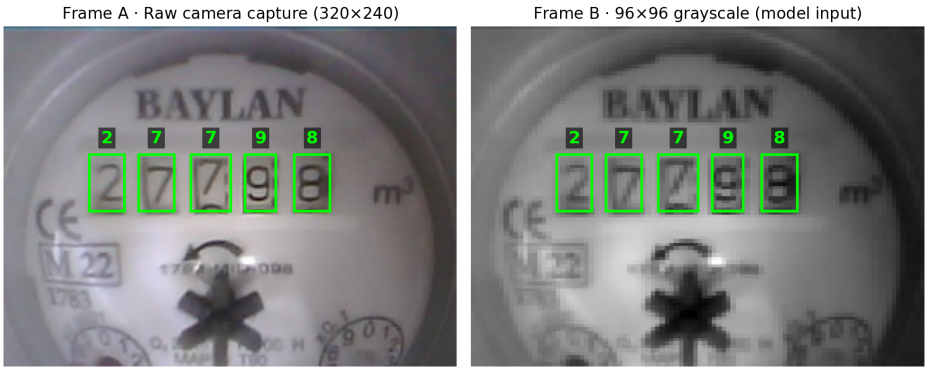


Figure 7. Preprocessing pipeline for a representative test frame

5. CONCLUSION

This paper presented an end-to-end on-device water meter digit recognition system for the ESP32-CAM microcontroller. An INT8-quantized MobileNetV2 classifier achieves 2 ms model inference latency, 235.6 KB peak RAM, a 99.60% overall cell-level accuracy including the background class, and an 87.70% exact five-digit reading accuracy on a 553-image box-annotated test subset drawn from a custom 6,155-image dataset. Digit classification, temporal validation, adaptive sampling, and anomaly detection execute entirely on the ESP32-CAM without external computational support. The optional MeterWatch visualisation and monitoring interface requires a local Flask server and Wi-Fi connectivity.

Three firmware contributions elevate the system beyond basic inference: temporal consistency filtering brings the false-positive reading rate below 1%, on-device anomaly detection identifies abnormal consumption events without server involvement, and adaptive sampling reduces idle inference cycles by 94%.

A Flask-based MeterWatch dashboard completes the stack with live visualisation, multi-device support, and anomaly monitoring.

Among the seven prior studies on water meter reading, six rely on external computation. Only one prior work addresses this task on the ESP32-CAM and the present system extends it with quantization, memory profiling, and three intelligent firmware behaviours not previously demonstrated in the water meter literature. To the best of our knowledge, this is the first system to combine on-device INT8 deployment, water meter digit recognition, and intelligent firmware behaviour on the ESP32-CAM platform.

Several limitations bound the current study. All data was collected from a single Baylan-brand meter under controlled laboratory conditions with a fixed enclosure and flash illumination. Without validation on at least two or three additional meter brands or a field deployment, performance generalisability remains an open question. The backbone is expected to adapt reasonably quickly if retrained on a few hundred additional images from a new meter brand, but cross-brand performance has not been empirically demonstrated. The anomaly detection threshold is a fixed empirical constant calibrated on one meter profile. Deployments with significantly different usage patterns may require recalibration. Furthermore, the anomaly detection evaluation comprised only fifteen injected events in a single session, which is insufficient to yield statistically meaningful detection rates. Future validation should include multi-session trials with hundreds of anomaly injections to derive confidence intervals. A broader statistical limitation applies to the reported metrics. The overall cell-level accuracy (99.60%), digit-only accuracy (96.89%), and exact-reading accuracy (87.70%) were calculated from an image-level test subset containing samples drawn from approximately 6–7 acquisition sessions. In contrast, the confirmed-reading rate, adaptive-sampling performance, and anomaly-detection rate were obtained from separate single experimental trials. Independent session-level replication, larger anomaly-event samples, and confidence-interval estimation are therefore necessary before these figures can be treated as stable performance bounds. MeterWatch currently stores all session data in server memory. A process restart results in complete loss of reading history, anomaly logs, and session statistics, and integration of a persistent database back-end (e.g., SQLite) is a prerequisite for production deployment. Finally, all evaluations were conducted under stable temperature conditions. The effect of ESP32 thermal variation on inference latency over extended deployments warrants further study.

ACKNOWLEDGMENTS

Figures 5 and 6 were generated programmatically in Python/Matplotlib directly from the raw model prediction outputs. The AI tool (Claude) was not involved in computing or verifying the numerical values shown, only in auxiliary schema layout and dashboard visualisation (Figures 3 and 4) and in formatting assistance. All scientific content, numerical values, and interpretations were computed from raw model outputs and verified by the authors.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Sohrab Jalali	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
Filiz Ersöz	✓	✓			✓					✓		✓	✓	
Babak Taeb		✓	✓	✓		✓		✓	✓	✓	✓			

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal Analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject Administration
Fu : **Fu**nding Acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author, S.J., upon reasonable request.

REFERENCES

[1] R. Laroca, V. Barroso, M. A. Diniz, G. R. Gonçalves, W. R. Schwartz, and D. Menotti, "Convolutional neural networks for automatic meter reading," *J. Electron. Imaging*, vol. 28, no. 1, 2019, doi: 10.1117/1.jei.28.1.013023.

[2] B. V. Nguyen *et al.*, "Water meter reading based on text recognition techniques and deep learning," *IEEE Access*, vol. 13, pp. 41422–41434, 2025, doi: 10.1109/ACCESS.2025.3547225.

[3] S. Zhang, Y. Wei, Y. Li, and C. Zhou, "Water meter reading recognition method based on character attention mechanism," *PLOS ONE*, vol. 20, no. 9, Art. no. e0332119, 2025, doi: 10.1371/journal.pone.0332119.

[4] Q. Hong *et al.*, "Image-based automatic watermeter reading under challenging environments," *Sensors*, vol. 21, no. 2, pp. 1–18, 2021, doi: 10.3390/s21020434.

[5] J. Li, J. Shen, K. Nie, R. Du, J. Zhu, and H. Long, "Reading recognition method of mechanical water meter based on convolutional neural network in natural scenes," *J. Adv. Comput. Intell. Intell. Inform.*, vol. 28, no. 1, pp. 206–215, 2024, doi: 10.20965/jaciii.2024.p0206.

[6] F. Martinelli, F. Mercaldo, and A. Santone, "Water meter reading for smart grid monitoring," *Sensors*, vol. 23, no. 1, pp. 1–14, 2023, doi: 10.3390/s23010075.

[7] S. Qiao, Y. Yuan, and R. Qi, "Meter-YOLOv8n: A lightweight and efficient algorithm for word-wheel water meter reading recognition," *Int. J. Adv. Comput. Sci. Appl.*, vol. 16, no. 4, 2025, doi: 10.14569/IJACSA.2025.0160422.

[8] P. Neve, "Digital transformation of conventional water meter using ESP32-CAM," *Int. J. Sci. Res. Eng. Manag.*, vol. 9, no. 6, pp. 1–9, 2025, doi: 10.55041/ijsrem48197.

[9] F. Ben Aicha, "Optimized embedded AI: Efficient implementation of CNNs on ESP32-CAM for real-time image classification," *Computing*, vol. 107, no. 11, Art. no. 206, 2025, doi: 10.1007/s00607-025-01559-z.

[10] V. Sineglazov and V. Khotsyanovsky, "Camera image processing on ESP32 microcontroller with help of convolutional neural network," *Electron. Control Syst.*, vol. 2, no. 72, pp. 26–31, 2022, doi: 10.18372/1990-5548.72.16939.

[11] K. Dokic, "Microcontrollers on the edge—Is ESP32 with camera ready for machine learning?" in *Proc. ICISP*, 2020, pp. 213–220, doi: 10.1007/978-3-030-51935-3_23.

[12] J. Paul, L. Schmid, M. Klaiber, and M. Rössle, "Analog gauge reading using regression on embedded systems," *Procedia Comput. Sci.*, pp. 2002–2011, 2024, doi: 10.1016/j.procs.2024.09.670.

[13] R. P. Narwaria, A. Kumar, A. Ahirwar, A. K. Tiwari, and A. K. Prajapati, "Smart object detection using ESP32-CAM based on YOLO algorithm," in *Proc. ICoICI*, 2024, pp. 817–820, doi: 10.1109/ICoICI62503.2024.10696374.

[14] I. Sulistiyowati, H. M. Ichsan, and I. Anshory, "Object sorting conveyor with detection color using ESP-32 camera Python based on OpenCV," *J. Electr. Eng. Comput. Sci.*, vol. 9, no. 1, pp. 61–68, 2024, doi: 10.54732/jeecs.v9i1.7.

[15] D. M. Hafidz, D. D. Sendjaya, M. O. Hasanuddin, and P. H. Rusmin, "Low-cost stereo vision system using ESP32-CAM and YOLO," in *Proc. ICWT*, 2024, doi: 10.1109/ICWT62080.2024.10674670.

[16] S. T. Nowroz, N. M. Saleh, F. Amsaad and M. I. Ibrahem, "A Cloud Computing-Enabled ESP32-CAM System for Real-Time Object Recognition with Feedback," in *2025 1st Int. Conf. Secure IoT, Assured Trusted Comput. (SATC)*, Dayton, OH, USA, 2025, pp. 1-5, doi: 10.1109/SATC65530.2025.11137265.

[17] Imron, B. Satria, S. Karim, and F. Ramadhani, "Cloud storage for object detection using ESP32-CAM," *TEPIAN*, vol. 5, no. 2, pp. 50–57, 2024, doi: 10.51967/tepian.v5i2.2994.

[18] A. T. Setiawan *et al.*, "YOLOv8 Implementation on ESP32-CAM for IoT-Based Human and Vehicle Detection," in *2025 IEEE Int. Conf. Data Software Eng. (ICoDSE)*, Batam, Indonesia, 2025, pp. 525-530, doi: 10.1109/ICoDSE68111.2025.11351787.

[19] Y. H. Chang, F. C. Wu, and H. W. Lin, "Design and implementation of ESP32-based edge computing for object detection," *Sensors*, vol. 25, no. 6, pp. 1–26, 2025, doi: 10.3390/s25061656.

[20] I. Nevliudov, N. Demska, V. Yevsieiev, K. Kolesnyk, S. Maksymova, and O. Miliutina, "Object recognition for a humanoid robot based on a microcontroller," in *Proc. MEMSTECH*, 2022, pp. 61–64, doi: 10.1109/MEMSTECH55132.2022.10002906.

[21] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv Preprint*, 2021, doi: 10.48550/arXiv.2106.08295.

[22] P.-E. Novac, G. B. Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and deployment of deep neural networks on microcontrollers," *Sensors*, vol. 21, no. 9, pp. 1–32, 2021, doi: 10.3390/s21092984.

[23] M. Y. Shabir, G. Torta, and F. Damiani, "TinyML model compression: A comparative study of pruning and quantization," *Telecommun. Syst.*, vol. 88, no. 4, 2025, doi: 10.1007/s11235-025-01363-2.

[24] J. Moosmann *et al.*, "TinyissimoYOLO: A quantized, low-memory footprint, TinyML object detection network for low power microcontrollers," in *Proc. IEEE AICAS*, 2023, doi: 10.1109/AICAS57966.2023.10168657.

[25] L. Boyle, J. Moosmann, N. Baumann, S. Heo, and M. Magno, "DSORT-MCU: Detecting small objects in real-time on microcontroller units," *IEEE Sensors J.*, vol. 24, no. 24, pp. 40231–40239, 2024, doi: 10.1109/JSEN.2024.3425904.

[26] J. Zhang, "Quantization-aware training for embedded AI systems," in *Proc. AICSS*, 2025, pp. 431–442, doi: 10.1145/3776759.3776826.

[27] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," in *Proc. NeurIPS*, 2021.

[28] N. Bawankar, A. Kriti, S. S. Chouhan, and S. Chaudhari, "IoT-enabled water monitoring in smart cities with retrofit and solar-based energy harvesting," *IEEE Access*, vol. 12, pp. 58222–58238, 2024, doi: 10.1109/ACCESS.2024.3392852.
[29] V. Pawar, A. Phule, and A. Metkari, "Review of smart energy meter implementations using ESP32 microcontroller and IoT platforms," *Int. J. Adv. Res. Sci. Commun. Technol.*, pp. 375–391, 2025, doi: 10.48175/ijarsct-29423.
[30] R. Samanta, B. Saha, and S. K. Ghosh, "A low-power low-cost system for disaster locations detection using ESP32 CAM and TinyML," in *Proc. COMSNETS*, 2025, pp. 907–910, doi: 10.1109/COMSNETS63942.2025.10885754.
[31] D. Donskoy, V. Gvindjiliya, and E. Ivliev, "TinyML classification for agriculture objects with ESP32," *Digital*, vol. 5, no. 4, pp. 1–14, 2025, doi: 10.3390/digital5040048.

BIOGRAPHIES OF AUTHORS



Sohrab Jalali received his B.Sc. degree in Computer Engineering from Ankara University, Ankara, Turkey, in 2022, and his M.Sc. degree in Software Engineering from OSTİM Technical University, Ankara, in 2026. He is currently a senior software developer focusing on artificial intelligence, computer vision, and autonomous systems. His research interests include edge AI and TinyML deployment on embedded hardware, image processing, deep learning, large language models, and natural language processing. He has contributed to defense technology projects involving autonomous drone systems, AI-powered perception, and embedded real-time control. He can be contacted at email: sohrabjallali@gmail.com.



Prof. Dr. Filiz Ersöz is a Professor at OSTİM Technical University, Ankara, Turkey. She received her bachelor's degree in Statistics from Anadolu University (1989) and her Ph.D. in Biostatistics from Ankara University (1998), and obtained the title of associate professor in Quantitative Decision Methods in 2011. Before academia, she worked as a Statistician at the Turkish Statistical Institute, as an Operational Research Expert at the Turkish Land Forces Command, and as a Probability and Stochastic Processes Specialist at the Turkish Military Academy. From 2012 to 2023, she was a faculty member in the Department of Industrial Engineering at Karabük University, and from 2023 to 2024 she served as a visiting professor at the Artificial Intelligence Research and Development Group of the University of Pisa. She is the author of several academic books, including Data Mining Techniques and Applications, Statistical Data Analysis with IBM SPSS, and Simulation and Modeling. Her research interests include data mining and predictive analytics, machine learning and artificial intelligence, decision support systems, simulation and modeling, operations research and optimization, and statistical quality control. She can be contacted at email: filiz.ersoz@ostimteknik.edu.tr.



Babak Taeb received his B.Sc. degree in Electrical Engineering and Telecommunication from Islamic Azad University, Science and Research Branch. He subsequently received his M.Sc. degree in Biomedical Engineering (Bioelectric) from the same institution, Tehran, Iran, with a thesis focused on improving the performance of brain-actuated wheelchairs using interactive error-related potentials. With 12 years of professional experience in software engineering and applied research, he has developed deep expertise across artificial intelligence, computer vision, image processing, autonomous systems, UAV software development, embedded AI, real-time systems, deep learning, and sensor fusion. He has served as a contributor to EU-funded research and innovation programs in smart agriculture, leading the development of computer vision and embedded AI solutions for greenhouse automation. He is currently serving as a Software Team Leader, where he directs the design and deployment of AI-powered perception and autonomous control systems. He can be contacted at email: babaktaeb@gmail.com.