Deployment of YOLOv8n for chrysanthemum growth stage detection on an edge AI device

Tran Minh Hai[1,3], Nguyen Minh Thang[2], Chu Minh Quang[2], Nguyen Hung Cuong[2], Dinh Thai Hoang[4], Nguyen Thi Ngan[5] and Bui Dang Thanh[1,2,*]

[1]*Institute of Automation and Control Technology, Hanoi University of Science and Technology, Vietnam*
[2]*School of Electrical and Electronic Engineering, Hanoi University of Science and Technology, Vietnam*
[3]*Faculty of Technology and Engineering, Thai Binh University, Vietnam*
[4]*Vietnam National University of Agriculture, Vietnam*
[5]*Viettri University of Industry, Vietnam*
[*]*Corresponding author E-mail: thanh.buidang@hust.edu.vn*
DOI: https://doi.org/10.64032/mca.v30i3.448

Abstract

This paper presents the deployment and evaluation of the YOLOv8n model for detecting the growth stages of *Chrysanthemum indicum* L. (CIL) on the Infineon PSOC™ Edge E84 AI Kit, a microcontroller-class Edge AI platform integrated with an Arm Ethos-U55 neural processing unit (NPU). A field dataset comprising 2,245 images and 9,517 annotated objects was collected from the Nghia Trai medicinal herb village in Hung Yen Province, Vietnam, and annotated based on the knowledge of agricultural experts. The YOLOv8n model was post-training quantized to INT8, reducing the model size by 73.4% compared with the FP32 TFLite version. A memory placement strategy was applied by storing the read-only model section in Flash while allocating execution memory in system SRAM (SoCMEM). The INT8 version of the model achieved an mAP_{50} of 0.831 on the test set. After deployment on the PSOC™ Edge E84 device, the system achieved a processing latency of 233 ms per image, approximately 4.29 FPS, with an average power consumption of 0.97 W, an energy consumption of 0.226 J per detection, including both inference and post-processing, and memory requirements of approximately 3.27 MiB RAM and 2.68 MiB Flash. The achieved on-board performance demonstrates the technical feasibility of executing INT8 YOLOv8n on the PSOC™ Edge E84 for periodic medicinal chrysanthemum growth-stage monitoring under the evaluated conditions.

Keywords: *Chrysanthemum; Microcontroller-class Edge AI; Post-Training Quantization; YOLOv8*

Abbreviations

CIL	*Chrysanthemum indicum* L.
FPS	Frames per second
FP32	32-bit floating point
INT8	8-bit integer
MCU	Microcontroller Unit
NPU	Neural Processing Unit
GPU	Graphics Processing Unit
PTQ	Post-training Quantization
SoCMEM	System SRAM
TFLite	TensorFlow Lite

1. Introduction

CIL is a perennial herbaceous plant widely distributed across East and Southeast Asia, where it has long been used as both a medicinal herb and a functional food ingredient [1], [2]. Its dried inflorescence, known as *Flos Chrysanthemi Indici*, is officially listed in the Chinese Pharmacopoeia and is recognized in traditional medicine systems in China, Korea, Japan, and Vietnam for its heat-clearing, detoxifying, and anti-inflammatory properties [3]. Modern phytochemical studies have also reported that CIL contains various bioactive compounds, including flavonoids, phenolic acids, sesquiterpene lactones, and polysaccharides, which contribute to its medicinal and commercial value [4]. A critical factor affecting CIL quality is the harvest timing relative to the plant developmental stage, since the concentration of key bioactive compounds can vary across growth stages, from vegetative development and bud formation to flowering and maturity [5]. In Vietnam, CIL is cultivated in specialized medicinal herb-growing areas such as Nghia Trai village in Hung Yen Province, where timely and accurate assessment of the crop growth stage is closely related to product quality and farmers' economic returns. However, current monitoring practices remain largely dependent on manual observation by farmers. This approach is experience-dependent, subjective, labor-intensive, and difficult to scale to larger cultivation areas. These limitations create a practical need for automated in-field tools capable of consistently identifying CIL growth stages from visual data.

Recent advances in machine vision and artificial intelligence have provided effective tools for automated crop monitoring in both controlled and field environments. Machine vision has been widely investigated for plant detection, growth monitoring, quality assessment, disease diagnosis, and yield estimation, particularly in plant factories and precision agriculture systems [6]. These studies show that visual sensing combined with deep learning can reduce dependence on manual observation and improve the consistency of crop monitoring. Nevertheless, practical agricultural applications still face

Received: 18 June 2026; Revised: 30 July 2026; Accepted: 13 August 2026.

challenges caused by complex backgrounds, illumination variation, occlusion, limited computing resources, and the need for real-time or near-real-time inference under deployment constraints.

Among deep-learning-based vision methods, object detection is suitable for plant growth monitoring because it can identify both the location and category of plant organs or growth states within an image. The YOLO family of detectors has been widely adopted in agricultural applications due to its favorable balance between detection accuracy and inference speed. For example, YOLOv8 has been applied to detect the growth stages of chilli plants in a hydroponic growing system, demonstrating the potential of YOLO-based deep learning for automatic plant-stage recognition [7].Recent studies have further extended YOLO-based vision systems to agricultural disease monitoring and embedded deployment. Ngo et al. integrated an improved YOLOv4-tiny detector into a Raspberry Pi-based system for detecting powdery mildew and downy mildew on cucumber plants, demonstrating the feasibility of low-cost embedded crop-monitoring systems [8]. Trinh et al. improved YOLOv8-based rice disease detection by replacing the conventional CIoU loss with Wise-IoU, resulting in improved mAP50 and mAP50:95 performance [9]. YOLO-based detection has also been explored in chrysanthemum-related studies. Qi et al. proposed F-YOLO for detecting the early flowering stage of tea chrysanthemum under field conditions with illumination variation, occlusion, and overlapping flowers [10]. This research direction was further extended through TC-YOLO, which improved tea chrysanthemum detection under unstructured environments [11]. Zhao et al. developed an improved CR-YOLOv5s model for detecting chrysanthemum inflorescences, focusing on buds and blooms under complex backgrounds [12]. More recently, Shi et al. proposed an improved YOLOv8s model for detecting chrysanthemum and classifying flowering stages [13]. These studies confirm that YOLO-based detectors can be effective for chrysanthemum detection and flowering-stage recognition.

However, existing chrysanthemum studies mainly focus on tea chrysanthemum, inflorescence detection, or flowering-stage classification. Most of them emphasize algorithmic improvement and detection accuracy, while deployment on highly resource-constrained embedded hardware is rarely addressed. In particular, limited work has investigated medicinal CIL growth-stage detection across multiple agronomic stages, such as vegetative, budding, flowering, and harvest, on a MCU-class Edge AI platform with an integrated NPU. This gap is practically important because field monitoring systems should not only be accurate, but also deployable on low-power devices with limited memory and computing resources.

Edge AI provides a promising direction for this problem by enabling image processing directly on local devices rather than relying entirely on cloud servers. Foundational studies on edge computing suggest that moving computation closer to data sources can reduce communication overhead, lower latency, and improve the responsiveness of Internet of Things (IoT) applications [14]. In agricultural environments, this is particularly important because network connectivity may be unstable, and continuous transmission of image data to cloud servers may not be practical for low-power field monitoring systems. Recent studies on Edge AI and TinyML in agriculture have also emphasized the importance of low-cost, low-power, and offline-capable inference for resource-constrained farming applications [15]. Nevertheless, many existing YOLO deployment studies are still conducted on relatively powerful edge platforms such as NVIDIA Jetson, Raspberry Pi, Coral TPU, or desktop-level GPU systems. For example, Alqahtani et al. evaluated object detection models on Raspberry Pi, Coral TPU, and Jetson Orin Nano platforms using metrics such as mAP, inference time, and energy consumption [16]. Rey et al. also analyzed YOLOv8n and YOLOv8s on Jetson Orin Nano, Jetson Orin NX, and Raspberry Pi 5 for drone-based applications [17]. These platforms provide substantially greater memory and computing resources than MCU-class devices. In contrast, MCU-class platforms impose much stricter constraints on RAM, Flash, runtime buffers, tensor arena allocation, and post-processing implementation. Therefore, deploying YOLO models on MCU-class Edge AI hardware requires not only model quantization, but also careful memory-constrained implementation and system-level evaluation.

The Infineon PSOC™ Edge E84 AI Kit is a relevant platform for investigating this challenge because it integrates an Arm Ethos-U55 NPU for embedded machine learning workloads. A recent study on rice disease detection showed that a quantized YOLOv5n model can be deployed on an Ethos-U55-based MCU platform, suggesting that YOLO-family models can be adapted to this class of hardware [18]. However, YOLOv8n deployment for medicinal chrysanthemum growth-stage detection remains a different problem due to differences in crop type, field dataset characteristics, model output representation, post-processing pipeline, and memory footprint.

Motivated by these gaps, this study deploys and evaluates an INT8 YOLOv8n model for medicinal chrysanthemum growth-stage detection on the PSOC™ Edge E84 AI Kit. A field image dataset collected from Nghia Trai medicinal village in Hung Yen Province, Vietnam, is used to formulate the task as object detection across four growth stages: vegetative, budding, flowering, and harvest. In addition to offline detection accuracy, this study evaluates system-level deployment metrics, including inference and post-processing latency, throughput, RAM and Flash footprint, power consumption, and energy per inference.

The main contributions of this study are summarized as follows.

Contribution 1 - Field dataset and detection-oriented task formulation. This study constructs an annotated field image dataset for medicinal chrysanthemum growth-stage monitoring under practical agricultural conditions. The problem is formulated as an object detection task rather than simple image classification, enabling both localization and classification of chrysanthemum instances across four phenological stages: vegetative, budding, flowering, and harvest.

Contribution 2 - INT8 YOLOv8n deployment with explicit memory placement on PSOC™ Edge E84. This work implements an INT8 YOLOv8n deployment on the PSOC™ Edge E84 AI Kit for chrysanthemum growth-stage detection. To fit MCU-class memory constraints, the read-only model parameters are placed in Flash memory, while runtime buffers, model state, and tensor-arena memory are allocated in SoCMEM. The YOLOv8n output handling is also adapted through dequantization and task-specific post-processing to generate final bounding boxes and growth-stage predictions on the target device.

Contribution 3 - System-level accuracy–efficiency evaluation. A system-level evaluation is conducted for the INT8 YOLOv8n model deployed on the PSOC™ Edge E84 AI Kit. The evaluation includes inference and post-processing latency, throughput, RAM and Flash footprint, power consumption, and energy per inference. In addition, offline accuracy comparisons among PyTorch, FP32 TFLite, and INT8 TFLite formats are reported to quantify the accuracy impact of model conversion and quantization before deployment.

2. Methodology

Figure 1 presents the overall workflow for deploying and evaluating YOLOv8n in this study. The workflow includes five main phases: Hardware Platform, Dataset collection and preparation, Post-training Quantization, Model conversion and embedded deployment, and Performance and resource evaluation. The focus is on the end-to-end deployment of a quantized YOLOv8n detector on the PSOC™ Edge E84 AI Kit. Therefore, the evaluation covers not only detection accuracy but also practical embedded deployment factors, including inference latency, RAM and Flash footprints, power consumption, and energy per inference.

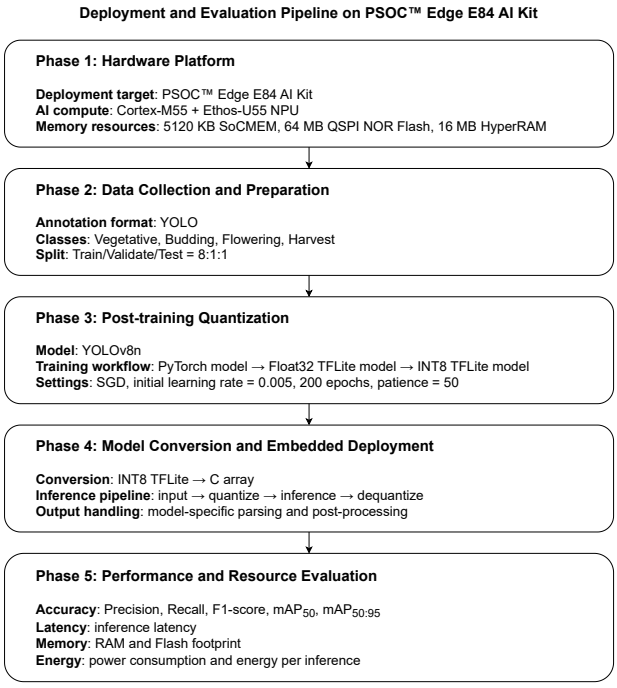

Figure 1: Proposed pipeline for deployment and evaluation on PSOC™ Edge E84 AI Kit

2.1 Hardware platform

The deployment target used in this study was the Infineon PSOC™ Edge E84 AI Kit, a microcontroller-based edge AI platform designed for embedded machine learning applications. The PSOC™ Edge E84 MCU integrates a high-performance Arm Cortex-M55 application processor, a low-power Arm Cortex-M33 processor, and an Arm Ethos-U55 neural processing unit, enabling on-device AI inference under embedded hardware constraints. In terms of memory resources, the platform provides 5120 KB of SoCMEM, which is used for runtime buffers and model state allocation. The kit also includes 512-Mbit (64 MiB) Quad-SPI NOR Flash and 128-Mbit (16 MiB) Octal-SPI HYPERRAM™, which support embedded storage and external memory expansion. Therefore, the PSOC™ Edge E84 serves as a suitable MCU-class test platform for evaluating the feasibility and limitations of on-device YOLOv8n deployment under strict memory, latency, and energy constraints. During the deployment experiments, both the Arm Cortex-M55 processor and the Ethos-U55 NPU were operated at 400 MHz. The same clock configuration was maintained during the latency and power measurements.

2.2 Dataset collection and preparation

The chrysanthemum growth dataset was collected using an iPhone 12 Pro Max at Nghia Trai medicinal herb village in Hung Yen Province, Vietnam, between 21 September 2025 and 17 December 2025. Images were acquired from multiple cultivation plots under various natural outdoor lighting conditions. However, the plot identifiers, chrysanthemum varieties, and quantitative shading levels were not systematically recorded.

The detection target was the apical growth region of an individual chrysanthemum shoot. Each bounding box encloses one apical region rather than the entire plant, branch, or flower cluster. Multiple apical regions in the same image were annotated independently. The Vegetative class contains young leaves and shoots without visible spherical flower buds; Budding contains one or more closed buds; Flowering contains partially opened flowers; and Harvest contains fully opened flowers assessed by the agricultural expert as suitable for harvest. Representative examples are shown in Figure 2.

The images were initially annotated by the research team according to expert-defined criteria, and the growth-stage labels were subsequently reviewed by Associate Professor Dr. Dinh Thai Hoang. Incorrect labels were corrected before training, validation, and testing. The annotations were stored in YOLO format using class indices and normalized bounding-box coordinates. The final dataset contains 2,245 source images and 9,517 annotated objects, including 1,923 Vegetative, 2,574 Budding, 2,547 Flowering, and 2,473 Harvest instances. The images were randomly divided into training, validation, and test sets at an approximate ratio of 8:1:1. The values in Table 1 represent annotated bounding boxes rather than source images. To further characterize the scale of the detection targets, Table 2 summarizes the bounding-box size statistics for each growth stage in terms of normalized width, height, and area ratio.

Table 1: Distribution of the chrysanthemum growth-stage dataset

Class	Train	Validate	Test	Total
Vegetative	1594	170	159	1923
Budding	2012	283	279	2574
Flowering	2013	265	269	2547
Harvest	1965	255	253	2473
Total	**7584**	**973**	**960**	**9517**

Table 2: Bounding-box size statistics for each growth stage.

Class	Width (%)	Height (%)	Area ratio (%)
Vegetative	12.9	16.9	2.2
Budding	9.1	11.7	1.0
Flowering	24.2	30.1	7.4
Harvest	40.9	47.6	18.4

(a) Vegetative stage **(b)** budding stage

(c) Flowering stage **(d)** Harvest stage

Figure 2: Representative images of CIL at four growth stages: (a) Vegetative, (b) Budding, (c) Flowering, and (d) Harvest.

2.3 Training and post-training quantization

The complete training, data augmentation, INT8 post-training quantization, and offline evaluation settings used in this study are summarized in Table 3. YOLOv8n was used as the object-detection model in this study because it is the nano variant of the YOLOv8 family and provides a practical trade-off between detection accuracy and computational complexity. The selected YOLOv8n model contains approximately 3.0 million parameters and was trained with an input resolution of 320×320 pixels, making it more suitable for resource-constrained embedded deployment than larger YOLOv8 variants. The YOLOv8 architecture follows a single-stage detection design consisting of three main components: a backbone for hierarchical feature extraction, a neck for multi-scale feature fusion, and a detection head for predicting object locations and class probabilities. Compared with earlier YOLO versions, YOLOv8 adopts an anchor-free detection head, which removes the need for predefined anchor boxes and directly predicts bounding boxes and class scores.

The YOLOv8n model was trained in DeepCraft Studio using the prepared chrysanthemum dataset. The training process used the SGD optimizer with an initial learning rate of 0.005 and a maximum of 200 epochs. Early stopping was configured with a patience value of 50 epochs to reduce the risk of overfitting and to select the best-performing model based on validation performance. Data augmentation was applied during training to improve the model's generalization ability under practical field conditions.

After training, the best-performing model was obtained in PyTorch format as a .pt model. As shown in Figure 3, this PyTorch model was then exported to a floating-point TensorFlow Lite model. The FP32 TFLite model was used as an intermediate representation before applying post-training quantization. In the next step, post-training quantization was performed using a representative calibration dataset to generate the final INT8 TFLite model. The calibration dataset was used to estimate the activation ranges of the model so that weights and activations could be represented using 8-bit integer values.

This workflow produced three model formats: the original PyTorch model, the FP32 TFLite model, and the INT8 TFLite model. These formats were evaluated offline on the same test set to analyze the effect of model export and quantization on detection accuracy. However, only the final INT8 TFLite model was used for embedded integration in the next phase, because it provides a lower memory footprint and is more suitable for MCU-class inference on the PSOC™ Edge E84 AI Kit.

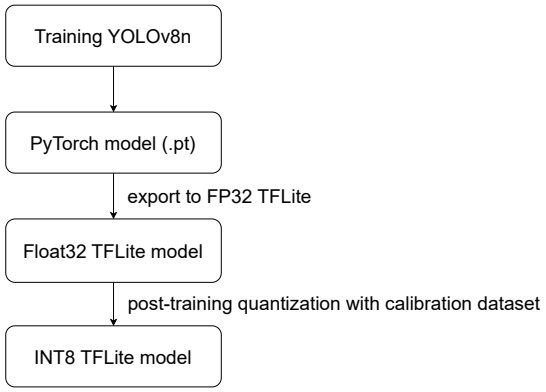

Figure 3: Training export and post-training quantization workflow

2.4 Model conversion and embedded deployment

After the training, export, and post-training quantization process, the final INT8 TFLite model was used for embedded deployment on the PSOC™ Edge E84 AI Kit. The INT8 TFLite model was converted into firmware-compatible C source files using the DeepCraft Model Converter. This step generated a C-array representation of the model parameters, allowing the model to be embedded directly into the ModusToolbox firmware project without requiring an external model file at runtime.

The generated deployment code includes the model initialization function, runtime memory configuration, working buffers, model state memory, and the inference entry point. The memory requirement is separated into three main regions: buffer memory, state memory, and read-only memory. Buffer memory is used as temporary working memory during model execution, including input scratch space, raw output storage, intermediate tensors, and dequantized output tensors used by the post-processing stage. State memory stores the runtime state required by the inference engine, including the model handle, tensor arena, and internal execution context. In con-

Table 3: Training, augmentation, quantization, and offline evaluation settings.

Category	Parameter	Setting
Training	Model	YOLOv8n
	Input resolution	320×320 pixels
	Initial weights	Random initialization (training from scratch)
	Batch size	32
	Maximum epochs	200
	Random seed	0
	Optimizer	SGD
	Initial learning rate	0.005
	Momentum	0.937
	Weight decay	0.0005
	Early-stopping patience	50 epochs
Augmentation	Geometric transformations	Degrees = 0, translate = 0.1, scale = 0.5, shear = 0, perspective = 0
	Flipping	Horizontal = 0.5, vertical = 0
	Image composition	Mosaic = 1.0, mixup = 0, copy-paste = 0
	Color augmentation	HSV hue = 0.015, saturation = 0.7, value = 0.4, BGR = 0
Quantization	Quantization method	INT8 post-training quantization
	Calibration dataset	Validation split
	Quantized components	Weights and activations
Offline evaluation	Software environment	Python 3.11.9; Ultralytics 8.4.19
	Minimum confidence threshold	0.001
	NMS IoU threshold	0.7

trast, read-only memory contains the constant model binary, including quantized weights, biases, and other fixed model parameters that are not modified during inference.

In the previous YOLOv5n deployment for rice disease detection, these model-related memory regions could be placed in SoCMEM because the memory requirement was sufficiently small [18]. However, the generated deployment required 324,000 bytes of buffer memory, 3,101,712 bytes of state memory, and 2,808,352 bytes of read-only memory. The state-memory allocation includes the tensor arena, model handle, and internal execution context. However, the generated deployment report did not expose the tensor-arena size as a separate value; therefore, the tensor-arena allocation cannot be reported independently from the total state-memory requirement. If all three regions were placed in SoCMEM, the total requirement would be 6,234,064 bytes, or approximately 5.95 MiB. This exceeds the 5 MiB SoCMEM capacity of the PSOC™ Edge E84. Therefore, the same SoCMEM-only placement strategy could not be reused for YOLOv8n.

To address this limitation, the read-only model section was placed in Flash memory, while buffer and state memory were kept in SoCMEM for runtime execution. This placement is appropriate because the read-only section contains constant model parameters and is not updated during inference. By storing the read-only model binary in Flash, the limited SoCMEM can be reserved for working buffers, tensor arena, and model state, which require RAM access during inference.

To identify the execution target of the neural-network operators, the final INT8 TFLite model was analyzed using Arm Ethos-U Vela Compiler version 5.1.0 with the ethos-u55-128 accelerator configuration and performance optimization mode. The compiled TFLite graph contained 269 operators, all of which were mapped to the Ethos-U55 NPU. No operator within the TFLite graph required Cortex-M55 CPU fallback. Therefore, the model achieved 100% NPU operator coverage within the TFLite graph. This value represents operator mapping rather than measured runtime NPU utilization.

This mapping applies only to the operators contained within the neural-network graph. Processing stages outside the TFLite graph, including camera acquisition, image resizing and padding, BGR565-to-RGB888 conversion, INT8 input preparation, output dequantization, YOLOv8n output decoding, confidence filtering, candidate sorting, non-maximum suppression, and bounding-box display, are executed by the Cortex-M55 processor or the corresponding peripherals.

The embedded inference pipeline is illustrated in Figure 4. The image captured from the OV7675 DVP camera has an original resolution of 320×240 pixels in BGR565 UINT8 format. Before inference, the camera frame is preprocessed and converted into a 320×320 RGB888 INT8 input tensor required by the YOLOv8n model. The INT8 input tensor is then passed to the model inference function for on-device inference.

After inference, the model produces a raw INT8 output tensor with a shape of 8×2100. This raw output cannot be directly used for detection visualization because it is still in quantized integer format. Therefore, the INT8 output tensor is dequantized into a floating-point output tensor using the corresponding output scale and zero-point. The resulting FP32 tensor preserves the YOLOv8n output structure and is used for the subsequent post-processing stage.

The dequantized YOLOv8n output is then parsed to extract bounding-box coordinates and class confidence scores. Confidence filtering is first applied to remove low-confidence predictions. Non-maximum suppression is then used to eliminate overlapping detections and retain the most reliable bounding boxes. The final output of this stage consists of the detected chrysanthemum growth-stage instances, including the bounding-box coordinates, confidence scores, and predicted class labels.

2.5 Performance and resource evaluation

The final INT8 TFLite model was evaluated in two complementary settings: offline detection-accuracy evaluation and on-device deployment evaluation. All detection-accuracy metrics reported in this study, including precision, recall, F1-score, mAP_{50}, and $mAP_{50:95}$, were obtained through offline evalua-

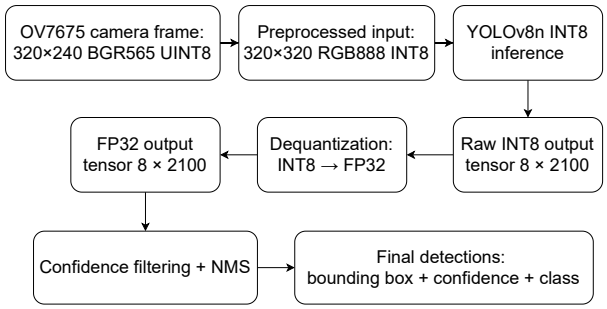

Figure 4: YOLOv8n embedded inference pipeline on the PSOC™ Edge E84 AI Kit

tion on the test images acquired using the iPhone 12 Pro Max. The on-device experiments using the OV7675 camera evaluated deployment-related metrics only, including processing latency, throughput, memory footprint, power consumption, and energy per detection. Therefore, the reported offline accuracy values should not be interpreted as accuracy measurements obtained directly from the deployed OV7675 camera.

For the offline evaluation, precision, recall, F1-score, mAP50, and mAP50:95 were calculated on the held-out test set. Precision measures the proportion of correct detections among all predicted detections, while recall measures the proportion of ground-truth objects correctly detected by the model. F1-score was calculated as the harmonic mean of precision and recall. These metrics are defined as follows:

$$\text{Precision} = \frac{TP}{TP+FP} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP+FN} \tag{2}$$

$$F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

where TP, FP, and FN represent true positives, false positives, and false negatives, respectively. In addition, mAP_{50} and $\text{mAP}_{50:95}$ were used as the main object-detection accuracy metrics. The mean average precision was calculated as:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{4}$$

where N is the number of object classes and AP_i is the average precision of class i. In this study, mAP_{50} evaluates detection performance at an IoU threshold of 0.50, whereas $\text{mAP}_{50:95}$ provides a stricter evaluation by averaging mAP across IoU thresholds from 0.50 to 0.95.

Runtime performance was evaluated using the inference latency measured directly on the PSOC™ Edge E84 platform. This value represents the practical time required to execute model inference and convert the raw model outputs into final detections. Computational memory usage was evaluated using the deployment memory report, where RAM footprint was calculated as the sum of buffer memory and state memory, while Flash footprint corresponded to the read-only model parameters stored in non-volatile memory.

Energy efficiency was evaluated using board-level power consumption and energy per inference. Power consumption was measured using a USB power meter while each model was executed continuously under the same operating conditions. Energy per inference was calculated by multiplying the average power by the measured inference latency. Since the measurement was performed at the board input, the reported power represents board-level consumption rather than the isolated power of the CPU, NPU, or model alone.

3. Results and analysis

3.1 Accuracy–size trade-off and class-wise impact of INT8 quantization

Before deployment on the PSOC™ Edge E84, the YOLOv8n model was evaluated in FP32 and INT8 TensorFlow Lite formats. The FP32 TFLite model was used as the floating-point baseline, while the INT8 TFLite model was generated through post-training quantization and selected for embedded deployment. The purpose of this comparison was to quantify the trade-off between detection accuracy and model size introduced specifically by INT8 quantization.

Table 4: Accuracy–size trade-off between FP32 and INT8 TFLite models.

Model format	mAP_{50}	Model size (MB)
FP32 TFLite	0.856	11.57
INT8 TFLite	0.831	3.08

As shown in Table 4, INT8 post-training quantization reduced the model size from 11.57 MB to 3.08 MB, corresponding to a reduction of approximately 73.4%. This reduction is important for deployment on the PSOC™ Edge E84 because the platform has limited on-chip memory resources and is optimized for INT8 inference. However, the overall mAP_{50} decreased from 0.856 to 0.831, corresponding to an absolute reduction of 0.025, or 2.5 percentage points. This result demonstrates the expected accuracy–size trade-off introduced by post-training quantization.

To determine whether the quantization-induced degradation was uniform across the four growth stages, the class-wise mAP_{50} values of the FP32 and INT8 TFLite models were further compared, as presented in Table 5.

Table 5: Class-wise impact of INT8 quantization on mAP_{50}.

Growth stage	FP32 TFLite	INT8 TFLite	$\Delta\,\text{mAP}_{50}$
Vegetative	0.879	0.838	−0.041
Budding	0.773	0.730	−0.043
Flowering	0.861	0.851	−0.010
Harvest	0.911	0.906	−0.005
Overall	**0.856**	**0.831**	−0.025

The results show that the impact of INT8 quantization was class-dependent. Budding exhibited the largest decrease in mAP_{50}, from 0.773 to 0.730, corresponding to a reduction of 0.043. Vegetative showed a similar reduction of 0.041. In contrast, the decreases for Flowering and Harvest were substantially smaller, at 0.010 and 0.005, respectively. Therefore,

the overall accuracy degradation was mainly associated with the two early growth-stage classes.

To further examine this class-dependent behavior, the normalized bounding-box dimensions were analyzed for each growth stage. The width and height values were calculated relative to the corresponding image dimensions, while the bounding-box area ratio was calculated as $w_{\text{norm}} \times h_{\text{norm}} \times 100\%$.

As shown in Table 2, Budding had the smallest median bounding-box area ratio, at 1.0%, followed by Vegetative at 2.2%. In comparison, the median area ratios of Flowering and Harvest were 7.4% and 18.4%, respectively. This distribution is consistent with the class-wise quantization results: the two classes containing the smallest detected objects also experienced the largest reductions in mAP_{50}.

At the input resolution of 320×320 pixels, small Budding and Vegetative objects occupy fewer image pixels and fewer spatial locations in the feature maps. Consequently, small perturbations in feature values, confidence scores, or bounding-box coordinates can have a proportionally greater effect on their detection and localization. INT8 representation uses a finite set of numerical levels for weights and activations, introducing rounding errors that can accumulate through the backbone, neck, and detection head. These errors may be particularly influential for small-object bounding-box regression, where relatively small coordinate changes can substantially affect the overlap between predicted and ground-truth boxes.

In addition to their small size, Budding objects frequently appear in dense clusters, are partially occluded by leaves and stems, and have relatively low color and texture contrast against the surrounding foliage. The discriminative activation differences between green buds and the green background may therefore be relatively small. During INT8 quantization, these subtle differences may be represented by only a limited number of quantization levels. Rounding errors and possible clipping of activation values outside the calibrated range may further reduce the confidence scores or perturb the predicted bounding-box coordinates.

Post-training quantization estimates the numerical ranges of model activations using a finite representative calibration dataset. If the calibration samples do not fully capture the activation distribution of small, low-contrast Budding objects, the effective numerical resolution available for these features may be reduced. However, layer-wise activation saturation and quantization errors were not directly profiled in this study. Therefore, calibration-range mismatch and activation clipping are considered plausible contributing mechanisms rather than independently verified causes.

Overall, the results suggest that the higher sensitivity of the Budding and Vegetative classes to INT8 quantization is associated with the combined effects of small object size, weak foreground–background contrast, dense spatial distribution, partial occlusion, and quantization-induced numerical errors. Despite the reduction in accuracy, the INT8 TFLite model retained an overall mAP_{50} of 0.831 while providing a substantial reduction in model size. It was therefore selected for subsequent feasibility evaluation on the PSOC™ Edge E84 platform.

3.2 Detection performance of the INT8 TFLite model

Following the accuracy–size and class-wise quantization analyses, the INT8 TFLite model was evaluated on the test set using precision, recall, F1-score, mAP_{50}, and $\text{mAP}_{50:95}$. The purpose of this evaluation was to determine whether the quantized model retained sufficient detection capability before integration into the PSOC™ Edge E84 firmware.

Table 6: Performance of the INT8 TFLite YOLOv8n model.

Growth stage	P	R	F1-score	mAP_{50}	$\text{mAP}_{50:95}$
Vegetative	0.800	0.728	0.762	0.838	0.487
Budding	0.734	0.649	0.689	0.730	0.474
Flowering	0.768	0.810	0.788	0.851	0.713
Harvest	0.760	0.916	0.831	0.906	0.806
Overall	**0.765**	**0.776**	**0.770**	**0.831**	**0.620**

As shown in Table 6, the INT8 TFLite YOLOv8n model achieved an overall precision of 0.765, recall of 0.776, F1-score of 0.770, mAP_{50} of 0.831, and $\text{mAP}_{50:95}$ of 0.620. These results indicate that the quantized model retained useful detection capability after post-training quantization.

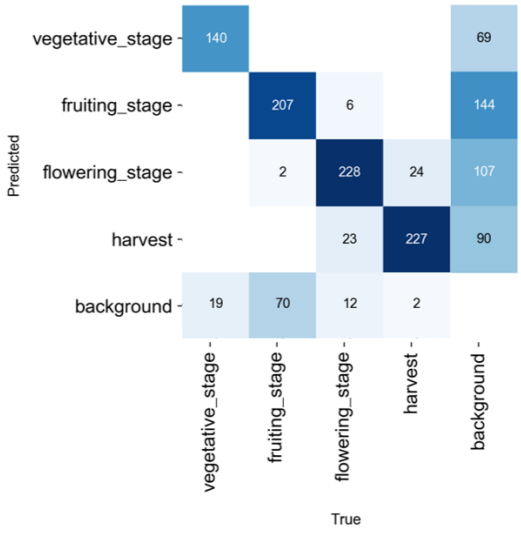

Figure 5: Confusion matrix of the YOLOv8n model on the test set.

To further examine the class-wise detection behavior, Figure 5 presents the confusion matrix, while Figure 6 shows the precision–recall curves of the four growth-stage classes. Both figures were generated through offline evaluation on the same test set used to obtain the results in Table 6.

The confusion matrix shows that 140 Vegetative, 207 Budding, 228 Flowering, and 227 Harvest objects were correctly detected at the operating threshold used to generate the matrix. The results also reveal that the main error associated with the Budding class was confusion with the background rather than confusion with another growth-stage class. Specifically, 70 ground-truth Budding objects were assigned to the background, representing the highest number of missed detections among the four classes. In addition, 144 Budding predictions were associated with the background, indicating that leaves, stems, and other background structures were also frequently detected as buds.

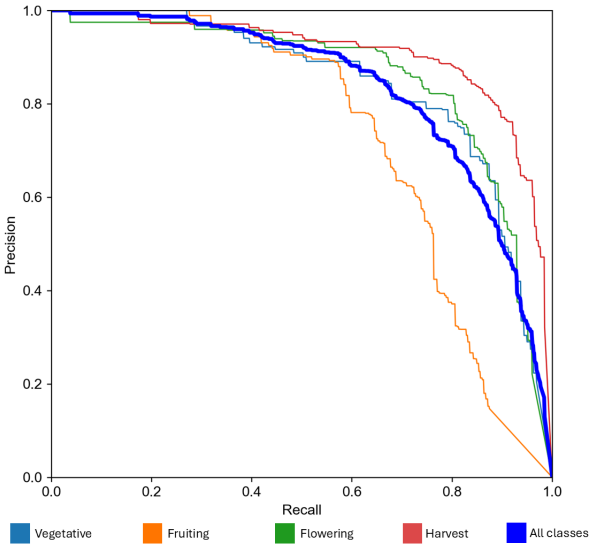

Figure 6: Precision–recall curves for the four growth-stage classes.

Inter-class confusion was most evident between the Flowering and Harvest stages. Twenty-three true Flowering objects were predicted as Harvest, while 24 true Harvest objects were predicted as Flowering. This bidirectional confusion can be associated with the gradual morphological transition between partially opened and fully opened flowers. In contrast, the Vegetative class showed little inter-class confusion, although 19 Vegetative objects were missed. The class-wise proportions derived from this confusion matrix correspond to the fixed operating threshold used to generate the figure and should not be interpreted as replacements for the precision and recall values reported in Table 6.

The precision–recall curves are consistent with the class-wise results presented in Table 6. The Harvest class maintains the highest precision over most of the recall range and achieves the highest AP_{50} of 0.906. Flowering and Vegetative show intermediate performance, with AP_{50} values of 0.851 and 0.838, respectively. In contrast, Budding achieves the lowest AP_{50} of 0.730 and exhibits the earliest and steepest decline in precision as recall increases.

For the Budding class, recovering additional objects at higher recall levels introduces a relatively large number of false-positive predictions. This observation is consistent with the confusion matrix, in which Budding has both the largest number of missed ground-truth objects and the largest false-positive background component. Nevertheless, the overall precision–recall curve corresponds to an mAP_{50} of 0.831, demonstrating that the quantized model maintains useful detection capability across the four growth stages.

Among the four growth stages, Harvest achieved the strongest performance, with a recall of 0.916, an F1-score of 0.831, an mAP_{50} of 0.906, and an $mAP_{50:95}$ of 0.806. Harvest-stage objects were generally larger and more visually distinguishable from the surrounding foliage, which provided clearer cues for detection and bounding-box localization.

Flowering also achieved relatively stable performance, with an mAP_{50} of 0.851 and an $mAP_{50:95}$ of 0.713. The higher $mAP_{50:95}$ values of Flowering and Harvest, compared with those of Vegetative and Budding, indicate that their predicted bounding boxes remained more accurate under stricter IoU thresholds.

Vegetative achieved the highest precision among the four classes, at 0.800, but its recall was lower at 0.728. This indicates that Vegetative predictions were generally reliable when produced, although some true objects were still missed. Its relatively low $mAP_{50:95}$ of 0.487 also suggests reduced localization stability at stricter IoU thresholds.

Budding remained the most challenging growth stage, with the lowest recall of 0.649, F1-score of 0.689, mAP_{50} of 0.730, and $mAP_{50:95}$ of 0.474. The confusion matrix and precision–recall curve confirm that this class produced more missed detections and a weaker precision–recall trade-off than the other stages.

Several visual characteristics contribute to the difficulty of detecting Budding objects. First, early-stage buds frequently occupy a small image area at the 320×320 input resolution, resulting in limited spatial detail for classification and bounding-box regression. Second, young buds are usually green and have relatively low color and texture contrast with the surrounding leaves and stems. Third, buds often appear in dense clusters and may be partially occluded by neighboring buds, leaves, or branches. These conditions increase the likelihood of missed detections, redundant predictions, and suppression of valid candidate boxes during non-maximum suppression.

INT8 post-training quantization may further affect these challenging cases because model weights and activations are represented using a limited number of integer levels. Weak feature responses associated with small and low-contrast buds may therefore be more sensitive to rounding and quantization errors than the stronger responses produced by larger and more visually distinctive Flowering and Harvest objects. Small variations in the quantized outputs may also affect confidence scores and bounding-box regression, thereby increasing the probability that valid Budding candidates are filtered or suppressed. However, because layer-wise activation distributions were not directly profiled in this study, these quantization-related effects are considered plausible contributing factors rather than independently verified causes.

Figure 7 provides a qualitative example of the detection errors observed for the Budding class. The model correctly detects several medium-sized and clearly visible buds, but some small, densely clustered, or partially occluded objects remain undetected. Several overlapping or redundant predictions are also produced in regions containing closely spaced buds and complex leaf structures.

Overall, the INT8 TFLite YOLOv8n model performed more reliably on visually distinctive stages such as Flowering and Harvest, while Budding remained the most difficult stage due to small object size, low foreground–background contrast, dense spatial distribution, and partial occlusion. Despite these challenges, the overall mAP_{50} of 0.831 indicates that the quantized model preserved sufficient detection capability for subsequent deployment on the PSOC™ Edge E84 platform.

3.3 On-device deployment performance on PSOC™ Edge E84

After the memory placement strategy described in the methodology section, the INT8 YOLOv8n model was converted into a firmware-compatible C-array representation and executed on the PSOC™ Edge E84 AI Kit. The on-device

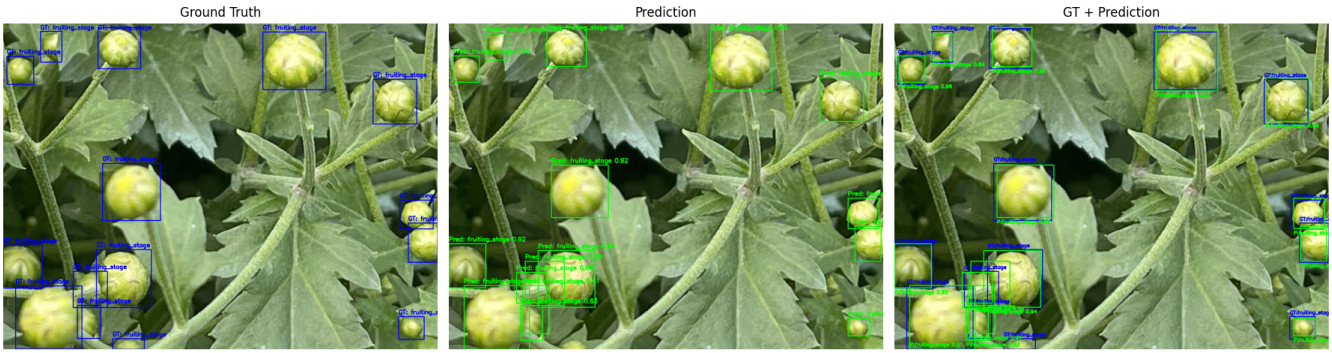

Figure 7: Qualitative detection results for a challenging Budding-stage image: ground-truth annotations, INT8 TFLite YOLOv8n predictions, and the overlay of ground truth and predictions.

deployment performance was evaluated in terms of processing latency, power consumption, energy per detection, and memory footprint. The input power of the prototype was measured using a KWS-V21 inline USB power tester. According to the vendor specifications, the tester has a voltage measurement range of 3.5–20 V and a current measurement range of 0–3.3 A, with a stated DC measurement accuracy of $\pm 1\%$ for both voltage and current.

The measured processing latency was 233 ms per frame. This value includes both neural network inference and YOLOv8n post-processing, rather than inference-only execution time. Specifically, it covers model execution, output dequantization, confidence filtering, candidate sorting, non-maximum suppression, and final output generation. Therefore, it represents the practical time required by the embedded system to produce final detection results from one input frame. Based on this latency, the deployed model achieved an approximate throughput of 4.29 FPS.

The power consumption of the system was estimated from the energy consumed during a 30-minute continuous detection run. During this measurement, the system continuously performed image acquisition, model inference, and post-processing under the same deployment configuration. The measured energy consumption over the 30-minute period was then used to calculate the average operating power, which was 0.97W. Since the latency value includes both inference and post-processing, the corresponding energy value is reported as energy per detection rather than inference-only energy. Each complete detection cycle consumed approximately 0.226 J per frame.

The deployed YOLOv8n model required approximately 3.27 MiB of RAM and 2.68 MiB of Flash memory. The RAM footprint corresponds to the buffer and state memory required during runtime execution, including working buffers, intermediate tensors, the tensor arena, and model execution state. The Flash footprint corresponds to the read-only model section. This section stores the constant C-array representation of the INT8 model, which contains quantized weights, biases, and other fixed model parameters. Since these values are not modified during inference, they can be stored in Flash memory instead of RAM. This memory organization follows the deployment strategy described earlier, where the read-only section is stored in Flash while buffer and state memory remain in SoCMEM.

These results show that YOLOv8n is relatively demanding for an MCU-class platform. However, the model can still be executed on the PSOC™ Edge E84 when INT8 quantization and memory placement strategy are applied. The achieved latency of 233 ms and energy consumption of 0.226 J per detection indicate that the system can support periodic image-based monitoring under the evaluated on-board conditions. In this application, real-time monitoring does not necessarily require continuous high-FPS detection, since plant growth-stage changes occur gradually over time.

3.4 Comparison with a lightweight baseline

As shown in Table 7, YOLOv5nu achieved a lower processing latency of 175 ms, corresponding to approximately 5.71 FPS, whereas YOLOv8n required 233 ms and achieved approximately 4.29 FPS. YOLOv5nu also consumed less energy per detection and required a smaller Flash footprint. However, YOLOv8n achieved an offline mAP_{50} of 83.1%, which was 0.9 percentage points higher than the 82.2% obtained by YOLOv5nu. These results demonstrate an accuracy–efficiency trade-off: YOLOv5nu provides greater deployment efficiency, whereas YOLOv8n provides slightly higher detection accuracy. Because the target application requires periodic growth-stage monitoring rather than continuous high-frame-rate processing, YOLOv8n was retained as the main model in this study.

Table 7: Comparison between the INT8 YOLOv5nu and YOLOv8n models.

Metric	YOLOv5nu	YOLOv8n
Offline mAP_{50} (%)	82.2	83.1
Processing latency (ms)	175	233
Energy per detection (J)	0.169	0.226
RAM footprint (MiB)	3.27	3.27
Flash footprint (MiB)	2.29	2.68

3.5 Comparison with representative studies

To position the proposed system relative to recent studies, two complementary comparisons are presented. The first comparison considers YOLO-based chrysanthemum detection studies and focuses on the agricultural task and detection accuracy. The second comparison considers YOLO deployment on resource-constrained Edge AI platforms and focuses on processing speed, power consumption, energy per detection, and memory requirements. Since the compared studies use

different datasets, input resolutions, detection targets, and hardware platforms, the reported values are used for contextual comparison rather than direct one-to-one benchmarking.

3.5.1 Comparison with chrysanthemum detection studies

Table 8 compares the proposed method with recent YOLO-based chrysanthemum studies. Previous studies achieved higher reported mAP50 values; however, they mainly considered one to three flowering-related classes and were evaluated using desktop-class GPUs. In contrast, the present study addresses four agronomically distinct stages of medicinal chrysanthemum, including Vegetative, Budding, Flowering, and Harvest, and deploys the quantized model on an MCU-class platform. Therefore, the lower mAP50 should be interpreted in the context of a more diverse detection task, INT8 post-training quantization, a lower input resolution of 320 × 320, and substantially stricter hardware constraints.

3.5.2 Comparison with Edge AI deployment studies

Table 9 compares the proposed system with representative object detectors deployed on Edge AI platforms. TinyissimoYOLO-based systems achieve substantially higher throughput and lower energy consumption, partly because they use an 88 × 88 input resolution and highly specialized compact architectures. Consequently, their performance is not directly comparable with the deployed YOLOv8n model using a 320 × 320 input.

Compared with the Raspberry Pi 4 and Coral TPU deployment, the proposed system has lower throughput but operates on an MCU-class platform with more restrictive runtime-memory and processing constraints. The proposed system consumes approximately 0.226 J per completed detection, which is lower than the reported 0.936 J per request for the Raspberry Pi 4 and Coral USB TPU configuration. In addition, the present study explicitly reports both runtime RAM and Flash footprints and demonstrates a memory-placement strategy that enables YOLOv8n execution within the available SoCMEM capacity.

These results demonstrate that the proposed system is competitive primarily in terms of deployment feasibility and balanced system-level performance. It maintains an mAP50 of 0.831 while executing a complete YOLOv8n detection pipeline on a low-power MCU-class platform using approximately 3.27 MiB RAM and 2.68 MiB Flash. Although its throughput is not intended for high-frame-rate video processing, the achieved 4.29 FPS is compatible with the intended concept of periodic growth-stage monitoring under the evaluated conditions.

3.6 Limitations

Although the dataset was collected from multiple cultivation plots with variations in plant morphology, canopy density, background conditions, illumination, and occlusion, all plots were located within the same geographical cultivation area in Nghia Trai medicinal herb village, Hung Yen Province. Therefore, the current evaluation does not capture possible variations associated with other geographical regions, soil conditions, cultivation practices, or local microclimates. In addition, although different chrysanthemum varieties may have been present among the cultivation plots, cultivar identities and the number of samples corresponding to each cultivar were not systematically recorded. Consequently, the current results should not be interpreted as evidence that the model generalizes independently of chrysanthemum variety.

The dataset was collected during a single cultivation season, from 21 September to 17 December 2025. Thus, inter-seasonal and inter-annual variations in plant appearance, weather, illumination, and cultivation conditions were not represented in the present evaluation. Furthermore, all images used for offline training and testing were captured using an iPhone 12 Pro Max, whereas the deployed prototype acquires images using an OV7675 camera. Differences in resolution, color response, noise characteristics, dynamic range, and image preprocessing between the two camera systems may introduce a camera-domain shift. Because an independently annotated OV7675 test set was not available, the resulting accuracy degradation on images acquired by the deployed camera could not be quantified in this study.

Another limitation concerns the image-level random partitioning of the dataset. Although the use of top-down and oblique viewpoints increased visual diversity and reduced exact image duplication, different views of the same plant or images obtained during the same acquisition session may remain strongly correlated. Therefore, the random split may have allowed correlated samples to occur in both the training and test sets, potentially producing an optimistic estimate of model generalization. Future datasets will retain plot, plant, date, and acquisition-session identifiers and will be evaluated using group-wise splits and fully independent external test sets.

Finally, the present evaluation was conducted as a single-image object-detection experiment and an on-board performance assessment rather than as a longitudinal field trial. The system has not yet been evaluated continuously across complete cultivation cycles or under long-term changes in illumination, temperature, humidity, and environmental conditions. Future work will therefore extend the dataset to multiple cultivation regions and growing seasons, systematically document chrysanthemum cultivars, and include images from multiple camera systems. Independent test sets separated by cultivation plot, season, cultivar, and imaging device will also be considered to provide a more rigorous evaluation of model generalization and practical field robustness.

4. Conclusion

This study presented an INT8 YOLOv8n-based embedded detection system for medicinal chrysanthemum growth-stage monitoring on the PSOC™ Edge E84 AI Kit. Instead of treating growth-stage recognition as a simple image classification problem, the task was formulated as an object detection problem, enabling both localization and classification of chrysanthemum instances across four phenological stages: vegetative, budding, flowering, and harvest. A field image dataset was constructed and annotated under practical agricultural conditions to support this detection-oriented monitoring task.

The final INT8 YOLOv8n model achieved an mAP_{50} of 0.831 after post-training quantization, while reducing the model size by 73.4% compared with the FP32 TFLite model. For embedded deployment, the INT8 model was converted into firmware-compatible C-array representation and integrated into the PSOC™ Edge E84 project. Because of memory requirements of YOLOv8n, the read-only model section was stored

Table 8: Comparison with existing chrysanthemum detection studies

Study	Chrysanthemum task	Number of classes	Dataset size	Input size	mAP_{50}	Hardware	FPS
Qi et al. [10]	Detecting the flowering stage of tea chrysanthemum	1	12,040 images	608×608	89.53%	NVIDIA GTX 1080 Ti	44.36
Qi et al. [11]	Detecting the flowering stage of tea chrysanthemum	1	1,000 images	416×416	92.49%	Tesla P100	47.23
Zhao et al. [12]	Detecting flowering periods of yellow chrysanthemums	2	250 images	640×640	93.9%	Tesla V100 NVLink	19.08
Shi et al. [13]	Detecting flowering stages of tea chrysanthemum	3	2,464 images	640×640	97.7%	NVIDIA RTX 3090	Not reported
This study	Detecting four growth stages of medicinal chrysanthemum	4	2,245 images	320×320	83.1%*	PSOC™ Edge E84 with Ethos-U55	4.29

* The mAP_{50} of this study was evaluated offline on the test set using the INT8 TFLite model before embedded deployment.

Table 9: Comparison with existing Edge AI studies

Study	Model	Hardware	Input size	FPS	Power	Energy	Memory
Moosmann et al. [19]	TinyissimoYOLO	MAX78000 MCU with CNN accelerator	88×88	Up to 180	Not reported	196 μJ/inference	398–422 kB weight memory; 25 kB input buffer; runtime RAM not reported
Moosmann et al. [20]	TinyissimoYOLO	GAP9 RISC-V MCU with NE16 CNN accelerator at 150 MHz	88×88	190.8	20.04 mW	105 μJ/inference	441 KiB
		GAP9 RISC-V MCU with NE16 CNN accelerator at 370 MHz	88×88	471.7	70.30 mW	149 μJ/inference	441 KiB
Alqahtani et al. [16]	YOLOv8n	Raspberry Pi 4	320×320	1.32	Not reported	2772 mJ/request	Not reported
		Raspberry Pi 4 with Coral USB TPU	320×320	29.4	Not reported	936 mJ/request	Not reported
This study	YOLOv8n	PSOC™ Edge E84 with Ethos-U55	320×320	4.29	0.97 W	226 mJ/detection	3.27 MiB RAM; 2.68 MiB Flash

in Flash memory, while SoCMEM was reserved for runtime buffer and state memory. This memory placement strategy allowed the model to execute within the constraints of the MCU-class platform.

The deployed system achieved a processing latency of 233 ms per frame, corresponding to approximately 4.29 FPS. The measured average power consumption was 0.97 W, and each complete detection cycle consumed approximately 0.226 J, including both inference and post-processing. The final deployment required approximately 3.27 MiB of RAM and 2.68 MiB of Flash. Overall, these results demonstrate the technical feasibility of deploying INT8 YOLOv8n on the PSOC™ Edge E84 for periodic chrysanthemum growth-stage monitoring under the evaluated conditions. However, the present results should not be interpreted as demonstrating field readiness across different geographical regions, growing seasons, chrysanthemum varieties, or imaging devices. Future work will focus on improving budding-stage detection and conducting multi-site, multi-season, multi-camera, and longer-term field evaluations using independently separated test sets.

References

[1] H. Hadizadeh, L. Samiei, and A. Shakeri, "Chrysanthemum, an ornamental genus with considerable medicinal value: A comprehensive review," *South African Journal of Botany*, vol. 144, pp. 23–43, 2022. DOI: 10.1016/j.sajb.2021.09.007

[2] Y. Shao, Y. Sun, D. Li, and Y. Chen, "Chrysanthemum indicum l.: A comprehensive review of its botany, phytochemistry and pharmacology," *The American Journal of Chinese Medicine*, vol. 48, no. 4, pp. 871–897, 2020. DOI: 10.1142/S0192415X20500421

[3] J. Li et al., "Evaluation of chrysanthemi indici flos germplasms based on nine bioactive constituents and color parameters," *PLoS One*, vol. 18, no. 4, e0283498, 2023. DOI: 10.1371/journal.pone.0283498

[4] G. Liu, E.-A. Alexa, and T. Zhang, "The medicinal landscape of chrysanthemum indicum l.: Bridging traditional wisdom and modern evidence," *Nutraceuticals*, vol. 6, no. 1, 2026, ISSN: 1661-3821. DOI: 10.3390/nutraceuticals6010017

[5] P. Yang et al., "Optimal harvest period and quality control markers of cultivated flos chrysanthemi indici using untargeted/targeted metabolomics, chemometric analysis and in vivo study," *Journal of Ethnopharmacology*, vol. 334, p. 118 533, 2024, ISSN: 0378-8741. DOI: 10.1016/j.jep.2024.118533

[6] Z. Tian, W. Ma, Q. Yang, and F. Duan, "Application status and challenges of machine vision in plant factory—a review," *Information Processing in Agriculture*, vol. 9, no. 2, pp. 195–211, 2022, ISSN: 2214-3173. DOI: 10.1016/j.inpa.2021.06.003

[7] F. Schneider, J. Swiatek, and M. Jelali, "Detection of growth stages of chilli plants in a hydroponic grower using machine vision and yolov8 deep learning algorithms," *Sustainability*, vol. 16, no. 15, 2024, ISSN: 2071-1050. DOI: 10.3390/su16156420

[8] Ngô Quang, T. D. Ngo, and D. T. Bui, "Improving the yolo v4 algorithm applied to the powdery mildew and downy mildew monitor system on cucumber plants," *Journal of Measurement, Control and Automation*, vol. 2, no. 2, 60–68, 2022.

[9] C.-D. Trinh et al., "Improving yolov8 deep leaning model in rice disease detection by using wise - iou loss function," *Journal of Measurement, Control and Automation*, vol. 29, no. 1, 1–6, 2025.

[10] C. Qi, I. Nyalala, and K. Chen, "Detecting the early flowering stage of tea chrysanthemum using the f-yolo model," *Agronomy*, vol. 11, no. 5, 2021, ISSN: 2073-4395.

[11] C. Qi, J. Gao, S. Pearson, H. Harman, K. Chen, and L. Shu, "Tea chrysanthemum detection under unstructured environments using the tc-yolo model," *Expert Systems with Applications*, vol. 193, p. 116 473, 2022, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2021.116473

[12] W. Zhao, D. Wu, and X. Zheng, "Detection of chrysanthemums inflorescence based on improved cr-yolov5s algorithm," *Sensors*, vol. 23, no. 9, 2023, ISSN: 1424-8220.

[13] G. Shi, J. Ji, T. Li, W. Li, Y. Li, and G. Zhang, "Detecting chrysanthemum to classify flowering stages using improved yolov8s," *Transactions of the Chinese Society of Agricultural Engineering*, vol. 41, no. 7, pp. 192–199, 2025. DOI: 10.11975/j.issn.1002-6819.202409108

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016. DOI: 10.1109/JIOT.2016.2579198

[15] R. Samanta and B. Saha, *Affordable precision agriculture: A deployment-oriented review of low-cost, low-power edge ai and tinyml for resource-constrained farming systems*, 2026. arXiv: 2603.15085 [cs.AI].

[16] D. K. Alqahtani, M. A. Cheema, and A. N. Toosi, "Benchmarking deep learning models for object detection on edge computing devices," in *Service-Oriented Computing*, W. Gaaloul, M. Sheng, Q. Yu, and S. Yangui, Eds., Singapore: Springer Nature Singapore, 2025, pp. 142–150, ISBN: 978-981-96-0805-8.

[17] L. Rey et al., "A performance analysis of you only look once models for deployment on constrained computational edge devices in drone applications," *Electronics*, vol. 14, no. 3, 2025, ISSN: 2079-9292. DOI: 10.3390/electronics14030638

[18] Q. Chu Minh et al., "Edge ai rice disease detection: A hardware performance comparison," *Journal of Measurement, Control and Automation*, vol. 30, no. 2, 18–24, 2026. DOI: 10.64032/mca.v30i2.410

[19] J. Moosmann, M. Giordano, C. Vogt, and M. Magno, "Tinyissimoyolo: A quantized, low-memory footprint, tinyml object detection network for low power microcontrollers," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, 2023, pp. 1–5. DOI: 10.1109/AICAS57966.2023.10168657

[20] J. Moosmann, H. Müller, N. Zimmerman, G. Rutishauser, L. Benini, and M. Magno, "Flexible and fully quantized lightweight tinyissimoyolo for ultra-low-power edge systems," *IEEE Access*, vol. 12, pp. 75 093–75 107, 2024. DOI: 10.1109/ACCESS.2024.3404878