Edge Artificial Intelligence for Real-Time Fresh Produce Identification in Retail Weighing Systems

Shi Han Teo, Jun Kit Chaw
Institute of Visual Informatics, Universiti Kebangsaan Malaysia (UKM), Selangor 43600, Malaysia

***Abstract*—Real-time recognition of loose fresh produce is a key requirement for intelligent retail weighing systems, enabling automated replacement of or assistance to manual PLU-based item selection. However, the deployment performance of recent YOLO architectures on embedded edge platforms such as the NVIDIA Jetson Xavier NX remains insufficiently studied in practical retail scenarios. This study aims to benchmark recent YOLO architectures for real-time fresh produce recognition on embedded edge devices. This work presents an Edge–AI retail weighing system that recognizes Malaysian fresh produce using YOLOv9, YOLOv10, and YOLOv11 models on the Jetson Xavier NX. A domain-specific dataset of 8 450 images across 26 classes was created by merging ImageNet and Roboflow sources and applying quality filtering and unified preprocessing. Each model was fine-tuned and optimized with TensorRT at FP16 and INT8 precision. Transfer learning improved accuracy across all models; YOLOv11-Large achieved the highest mAP@0.5 of $\approx$ 0.897 but at a reduced frame rate, while the mid-sized YOLOv10-M delivered an mAP@0.5 of $\approx$ 0.890 with near-real-time performance inference. Inference analysis shows that pre- and post-processing add only a few milliseconds per frame yet become proportionally significant as inference speeds increase; YOLOv11's Non-Maximum Suppression (NMS) head introduces notable latency relative to YOLOv10's NMS-free design. Quantized YOLOv10-M and YOLOv10-N sustain $\approx$ 14–19 FPS , offering the best balance between accuracy and speed. Qualitative tests on market footage confirm robust detection, indicating that these optimized models enable accurate, low-latency produce identification for intelligent retail weighing.**

***Keywords*—Real-Time object detection; Edge Artificial Intelligence (Edge AI); transfer learning; YOLO algorithm; nvidia jetson; fresh produce recognition; TensorRT optimization; model quantization**

I. Introduction

In a grocery store, weight-based pricing is a common way the grocery store charges the customer. Selling the prepackaged goods might be a solution to simplify the process by eliminating the need for in-store weighing at the time of purchase. Loose produce still plays an important role because of the flexibility in purchase quantities and the potential to reduce household food waste [1]. This further highlights that the weighing scales play a critical role in modern fresh produce retail environments. However, the traditional weighing scales required the user to control the PLU(Price Look-Up) subsystem in the software to retrieve the unit price of the item. This will require the staff to memorize the codes and manually control the machine to complete the checkout task. Consequently, it will cause inefficiency during peak hours. These challenges indicate that there is a need for a more intelligent and automated solution by utilizing computer vision and artificial intelligence [2]. To be practical in retail checkout environments, such a solution must operate in real time, without reliance on external connectivity, and with minimal computational overhead.

Edge AI has recently emerged as a key paradigm for high-performance inference in real-time systems. Unlike cloud-based AI inference, edge AI operates locally on the device, thus eliminating the dependency on continuous internet connectivity and minimizing the latency introduced by network transmission [3]. In the context of real-time fresh produce detection, Edge AI plays an important role in maximizing the user experience because it allows rapid inference speed and eliminates the latency that causes a laggy experience by the user [4]. By combining the Edge AI platform with a real-time object detection algorithm, it is possible to come out with a robust retail weighing solution.

Among object detection frameworks, the YOLO (You Only Look Once) object detection model is widely recognized for real-time applications due to its single-stage architecture, which performs object localization and classification simultaneously, unlike two-stage detectors [5]. YOLO algorithms have undergone continuous architecture improvements, leading to a succession of improved versions such as YOLOv1 until YOLOv11. These continuous architecture improvements impact accuracy, training time, and inference performance [6]. Consequently, YOLO is a suitable candidate for deployment in intelligent retail scales to detect fresh produce. However, within this context, there is limited evaluation of the inference speed and detection accuracy for the different versions of YOLO algorithms, particularly when optimized using TensorRT frameworks.

However, when it comes to detecting agricultural produce in Malaysian grocery-stores, applying transfer learning to a YOLO model is essential. The pretrained YOLO algorithm is based on a general dataset such as the COCO dataset. This means that a domain-specific dataset is necessary to train a YOLO model that fits the context of agricultural produce [7]. The well-known general datasets, such as COCO and ImageNet, offer broad object coverage, but are not focused on representing agricultural produce found in Malaysian retail markets. Previous research indicates that domain-specific datasets generally outperform generic ones due to their tailored relevance and coverage [8]. The data set gap here causes uncertainty on how effectively YOLO models perform in recognizing fresh produce with high intra-class variation, such as differences in size, shape, or color of the same item, and high inter-class similarity characteristics, such as similar-looking fruits that general-purpose datasets fail to capture.

When deploying deep learning models on embedded edge devices such as the NVIDIA Jetson series, achieving real-time

inference is often constrained by limited computational and memory resources. To overcome these limitations, TensorRT provides a hardware-aware optimization framework that restructures neural network graphs through layer fusion, kernel auto-tuning, and precision calibration from FP32 to FP16 or INT8, significantly improving throughput while maintaining near-lossless accuracy [9]. By leveraging TensorRT, models can exploit the full potential of Jetson's CUDA and Tensor cores, reducing latency and power consumption—both critical for real-time retail weighing applications.

Despite these advantages, comparative analyses of TensorRT-optimized YOLOv9, YOLOv10, and YOLOv11 models for multi-class retail produce detection under edge constraints remain scarce. Most existing studies benchmark earlier YOLO architectures or focus on general object classification tasks rather than fine-grained produce detection on embedded platforms. This study addresses this gap by evaluating the inference speed and detection accuracy of TensorRT-optimized YOLO models on the Jetson Xavier NX for real-time produce recognition.

This study addresses these challenges by developing a prototype smart retail system based on the NVIDIA Jetson Xavier NX platform and integrating YOLOv11, YOLOv10, and YOLOv9 models to assess inference speed and detection accuracy for agricultural produce commonly sold in Malaysia.

This work aims to advance the field of object detection by presenting a comprehensive evaluation of state-of-the-art YOLO models—namely YOLOv9, YOLOv10, and YOLOv11—in the context of real-time fresh-produce recognition for smart retail systems. The study contributes in the following aspects:

- Identifies architectural design principles for low-latency edge-based object detection by analysing the end-to-end inference behaviour of YOLOv9–YOLOv11 on an embedded GPU platform.
- Quantifies the practical impact of NMS-based versus NMS-free detection heads in retail inference pipelines, demonstrating how post-processing overhead increasingly dominates latency in real-time edge deployment.
- Establishes a reproducible benchmarking framework for evaluating modern YOLO architectures under realistic retail constraints, including unified preprocessing, transfer learning, and TensorRT-based optimization on the NVIDIA Jetson Xavier NX.
- Derives precision–latency trade-offs across model scales and quantization modes (FP16 and INT8), providing deployment-level guidance for selecting suitable configurations under strict real-time constraints.
- Delivers a validated reference configuration for intelligent retail weighing systems, achieving near real-time performance with high detection accuracy, and demonstrating transferability beyond a single dataset or deployment scenario.

The remainder of this paper is organized as follows. Section II surveys related work on edge AI for computer vision and existing produce-recognition approaches, highlighting the need for efficient deployment on embedded devices. Section III details the proposed methodology, including the system architecture, domain-specific dataset construction, transfer learning strategies, and TensorRT-based optimisation of YOLO models. Section IV presents and discusses the experimental results, analysing detection accuracy, inference speed, precision–recall characteristics, and the impact of model size and quantisation across YOLOv9, YOLOv10, and YOLOv11. Finally, Section V offers the conclusion, summarising key findings, discussing remaining challenges and limitations, and suggesting directions for future research.

II. Related Works

A. Edge AI and Deep Learning-Based Computer Vision

The concept of Edge AI has shifted the paradigm of artificial intelligence deployment, especially in real-time applications. The Edge AI approach aims to bring computation closer to the data source, such as video streams, thereby eliminating network latency that often occurs in cloud-based processing [3].

Recent studies have demonstrated the feasibility and efficiency of deploying deep learning models directly on embedded hardware for real-time inference. A real-time myocardial infarction monitoring system using SSD-Inception V2 and MobileNet V2 on the NVIDIA Jetson Nano achieved a mean average precision (mAP) of 76.4 % under low-power operation, setting a benchmark for energy-efficient edge analytics [10]. The YOLOSR architecture combined a Super-Resolution (SR) branch with YOLOv8-small, improving detection accuracy by 10.2 % without added latency (101 ms per frame) on a Jetson Nano edge platform [11]. Complementary works have also validated the effectiveness of Edge AI across domains, including FPGA-based YOLO implementations for Advanced Driver Assistance Systems (ADAS) and surveillance applications [12], and real-time apple detection on embedded GPUs achieving 83.6 % mAP at 30 FPS [13], demonstrating that optimized architectures and hardware accelerators enable near-cloud accuracy on compact edge systems.

While Edge AI has been widely applied in healthcare, surveillance, and autonomous navigation, fewer studies have addressed its use in retail environments. There is limited attention to retail applications, particularly in detecting multiclass grocery fresh produce in Malaysia. Building on prior successes, this study applies Edge AI for real-time produce detection integrated with a retail weighing system as a smart retail solution.

B. Edge AI and Platform-Specific Optimization

Deploying deep learning models on edge devices often demands hardware-aware optimization to achieve real-time inference under limited computational and memory resources. Among the most widely used frameworks, NVIDIA TensorRT provides significant acceleration by performing layer fusion, precision calibration (FP32→FP16 or INT8 quantization), and kernel auto-tuning tailored for GPU-based embedded platforms such as the Jetson Nano, TX2, and Xavier. These optimizations restructure neural network graphs and dynamically allocate tensor memory to reduce latency and improve throughput, enabling efficient deployment of convolutional neural networks in time-critical applications.

Recent studies have demonstrated the tangible benefits of such optimization pipelines across various domains. For instance, a TensorRT-optimized driver drowsiness classification system—based on MobileNetV1 rather than a detection architecture—achieved a 79.27% increase in throughput (from 141.78 FPS to 683.92 FPS) on Jetson Nano while maintaining 98.19% accuracy, highlighting TensorRT's capability to deliver near-lossless precision under quantized execution for lightweight classification tasks [14]. However, this exceptionally high throughput does not represent object detection workloads such as YOLO, which involve additional computation for multi-scale feature heads and non-maximum suppression. Similarly, a YOLOv5-based tennis-ball detection model combined GhostNet, BiFPN, and TensorRT to enhance both detection accuracy (mAP 94%) and inference efficiency through layer fusion and memory optimization on RTX hardware [15]. It is noteworthy, however, that although TensorRT is fundamentally designed to accelerate inference on embedded and edge AI platforms such as the NVIDIA Jetson series, this study conducted all experiments on a high-end desktop GPU (RTX 3060) under a Windows environment rather than on an edge device. Consequently, the reported throughput improvements primarily validate the computational acceleration of TensorRT on general-purpose GPUs, but do not assess its performance or feasibility under the resource and power constraints typical of real-time edge deployments. This distinction underscores a limitation in edge-specific validation, which remains essential for evaluating true on-device efficiency and deployability in embedded AI contexts.

In the agricultural domain, the PcMNet model integrated YOLOv8 with partial convolution and TensorRT acceleration, achieving 92 FPS on Jetson-based edge devices while reducing FLOPs by 37.8% and parameters by 53.3%, thereby validating its suitability for real-time fruit detection in orchard environments [16]. In contrast, the strawberry segmentation model based on Mask R-CNN [17] focused primarily on high-precision semantic segmentation under occlusion but was evaluated on desktop GPUs without edge-oriented deployment. The peanut leaf disease detection study [18] employed a classification-based approach using MobileNetV3, EfficientNet, and ShuffleNet models optimized via TensorRT for Jetson devices. Unlike object detection frameworks, this work focused solely on image-level disease classification without bounding-box localization.

Collectively, these works underscore the importance of platform-specific optimization for achieving real-time, power-efficient inference on edge AI systems. However, the majority of existing research has primarily focused on model-level architectural enhancements—such as backbone redesigns, attention modules, or lightweight feature extractors—rather than conducting comprehensive TensorRT benchmarking across recent YOLO variants. Furthermore, comparative evaluations of quantized (FP16/INT8) detection pipelines on embedded NVIDIA platforms remain limited, particularly in domain-specific applications such as agricultural produce recognition and smart retail environments, where the trade-off between detection accuracy and latency is critical. Addressing this gap, the present study systematically investigates the deployment performance of TensorRT-optimized YOLOv9, YOLOv10, and YOLOv11 models on the Jetson Xavier NX, providing an in-depth analysis of their precision–speed trade-offs and validating their suitability for real-time edge AI inference.

C. YOLO Evolution and Comparative Performance

Since the introduction of the YOLO algorithm in 2015, the single-stage detection paradigm has eliminated the computational overhead associated with region proposal pipelines used in two-stage detectors such as Faster R-CNN. By skipping the region proposal step, YOLO has undergone several massive improvements over successive iterations. The latest breakthroughs are represented in YOLOv9, YOLOv10, and YOLOv11 [6]. The advancements in YOLOv9 introduced GELAN and PGI, which reduce parameters yet achieve higher accuracy. In YOLOv10, the traditional Non-Maximum Suppression (NMS) was eliminated and replaced with Consistent Dual Assignments (CDA), meaning the algorithm no longer produces multiple overlapping bounding boxes. YOLOv11 introduces further innovation with modules such as C3k2 and C2PSA, which help detect smaller objects more effectively [19]. These distinct innovations create an opportunity to explore their impact in domain-specific scenarios, particularly in real-world and practical use cases.

There are many studies being done to compare various versions of YOLO algorithms in agricultural domains. A study compared YOLOv9–v12 for cattle precision farming [20], where YOLOv12 achieved the highest mAP of 98.3% but with slower inference speed of 36.6 FPS, while YOLOv11 maintained nearly the same accuracy at 97.8% but ran faster at 48 FPS. YOLOv10 also reached 97.1% mAP with 44 FPS, and YOLOv9 was the weakest with 96.2% mAP and 37 FPS. Another study on banana bunch detection [21] across YOLOv1–YOLOv12 reported that YOLOv12n achieved 93% AP50 with a latency of only 5.1 ms, making it well-suited for mobile deployment, while YOLOv11n also showed competitive efficiency under constrained devices. However, not all studies concluded that the higher the version, the higher the mAP values. In multi-class fruit ripeness detection [22] , the YOLOv6 outperformed YOLOv7 and SSD by achieving 99.5% mAP together with an inference speed of 85.2 FPS, showing that earlier versions can still dominate when the class variation is limited. Weed detection study [23] provided further evidence of trade-offs: YOLOv9 achieved the highest mAP@0.5 of 93.5% across multiple species, while YOLOv11 was the fastest with inference time of 13.5 ms, making it more suitable for real-time robotic field deployment. The anchor-based YOLOcF derivative [24] , developed on YOLOv5, also challenged mainstream versions by surpassing YOLOv10n and YOLOv11n in accuracy, showing up to 0.8–1.3% higher mAP, and achieving a record inference speed of 323 FPS, although it remained 1.4% lower than YOLOv9t in overall precision.

Despite many comparative studies being conducted in agricultural domains, there remains a gap in benchmarking YOLO algorithms for detecting loose produce commonly sold in grocery store. In Malaysia, this involves a wide variety of classes and requires a domain-specific dataset that reflects local produce characteristics rather than relying on generic datasets like COCO. This study therefore aims to bridge the gap by conducting a comparative analysis of YOLOv9, YOLOv10, and YOLOv11 tailored to Malaysian retail produce, while also evaluating the accuracy and latency of real-time inference under edge computing constraints.

III. Methodology

A. System Architecture

As shown in Fig. 1, the entire system is implemented based on a three-tier structure, which consists of the Edge Layer, Fog Layer, and Cloud Layer. This design follows a 3-tier architecture, where each layer handles specific responsibilities to ensure efficiency and real-time performance.

At the Edge Layer, a Raspberry Pi is used to prototype the regular weighing scale by connecting the HX711 amplifier board with the load cell. The weight value stream is then stabilized using an average buffer algorithm to reduce noise and ensure stable readings. The NVIDIA Jetson Xavier NX is integrated with a camera, and the YOLO inference source code is deployed and further customized to integrate with MQTT messaging capability, enabling the detection results to be streamed in JSON format. These JSON messages are published to the MQTT broker.

On the customer side. The Flutter-based kiosk application subscribes to these MQTT messages and parses the JSON data. The app uses asynchronous programming to handle concurrency smoothly when frequent produce or weight changes occur. Within the kiosk, a local SQLite database is queried to retrieve the unit price of the detected produce, which is then mapped against the YOLO result and weight reading to compute the final total. The recognized produce, weight, and computed price are displayed on the Android tablet, and once confirmed by the customer, a QR code is generated for checkout.

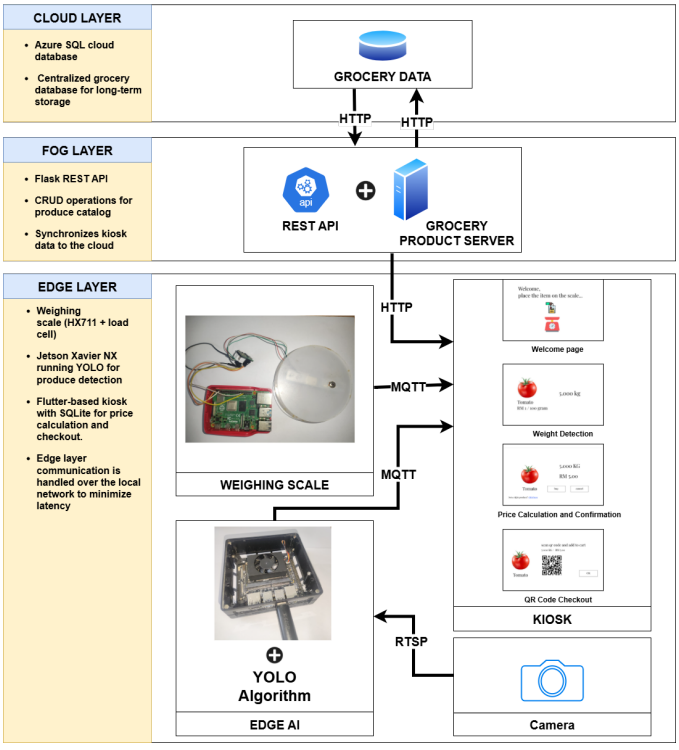

Fig. 1. Three-tier system architecture consisting of Edge, Fog, and Cloud layers.

At the Fog Layer, the system acts as the bridge between the edge devices and the cloud. A Python Flask service with REST API is used to receive both detection results and transaction data, providing a simple way to buffer and synchronize information before pushing it to the central database. To cope with the dynamic pricing of fresh produce, this layer also allows administrators to perform CRUD operations such as updating the produce catalog, adjusting unit prices, or editing nutritional details. Once the kiosk tablet has internet access, it automatically syncs its local records with the fog server, ensuring that both price changes and transaction logs remain consistent across all connected devices. Fig. 2 illustrates the overall architecture of the proposed Edge-AI retail weighing system, including image acquisition, object detection, and inference execution on the embedded device.

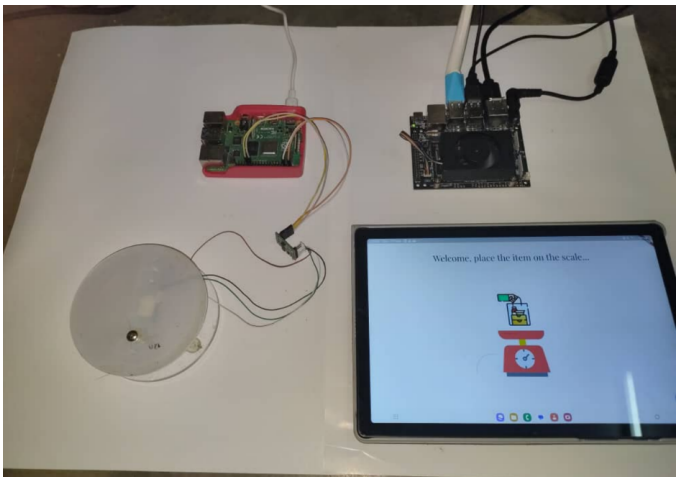

Fig. 2. Prototype setup of the smart retail weighing scale with raspberry Pi, jetson xavier NX, and flutter-based kiosk.

B. Data Acquisition

Data acquisition forms the cornerstone of this study, as it prepares the dataset for domain-specific modeling aimed at detecting commonly sold loose produce in Malaysian retail environments. This process seeks to overcome the limitations of non-contextual generic datasets by drawing and filtering relevant data from ImageNet and Roboflow. While the ImageNet dataset provides extensive image-class coverage, particularly for classification tasks, its object-detection annotations remain limited in quantity. Hence, the Roboflow dataset is utilized to enrich the overall corpus by increasing the number of annotated samples and enhancing the contextual diversity of the data. The overall data acquisition process is shown in Fig. 3.

ImageNet is organized according to the WordNet lexical hierarchy, a semantic network in which each synset (synonym set) represents a distinct concept. Within this hierarchy, this structure allows direct mapping between linguistic concepts and their corresponding visual representations. In this study, synsets relevant to the targeted 26 agricultural produce classes were retrieved from the ImageNet21K (Winter 2021 Release) repository by replacing the synset id in the URL with the pattern of https://image-net.org/data/winter21_whole/SYNSET_ID.tar. Each class identifier follows the WordNet ID (WNID) convention (e.g., n07756951 for tomato). In this study, the classname

is retrieved using Python and the Natural Language Toolkit (NLTK) interface to WordNet, because WNIDs were programmatically mapped to their lexical names and definitions as illustrated in the Listing 1.

All ImageNet annotations, originally in Pascal VOC XML format, were converted into YOLO-compatible bounding-box files (cx, cy, w, h) using a Python-based conversion pipeline. Each conversion was visually inspected through overlay validation to ensure proper localization accuracy. During dataset inspection, it was observed that not every retrieved sample was relevant to the intended classes. Therefore, a second-stage quality filtering process was introduced using a ResNet-50 image-classification model pre-trained on ImageNet. The model operated as a fixed feature extractor, computing cosine similarity between each image embedding and a curated exemplar set to assess relevance. Samples scoring below a similarity threshold of 0.8 were discarded. As illustrated in Fig. 4, non-relevant ImageNet samples were identified and removed through a classification-based quality filtering process to improve dataset relevance for produce recognition. The resulting refined subset was then saved into directories named by class to facilitate subsequent merging with the Roboflow dataset.

Algorithm 1 Conversion of WordNet ID (WNID) to Synset Information

Require: WNID string (e.g., "n07756951")
Ensure: Synset name and definition corresponding to the given WNID
Import nltk and wordnet modules
Download WordNet corpus if not already available
function WNID_TO_SYNSET(wnid)
    offset ← integer value of substring of *wnid* starting from index 2
    try: synset ← WordNet.synset_from_pos_and_offset('n', offset)
        return (synset.name(), synset.definition())
    except: return (None, "Error")
end function

To collect the relevant object-detection datasets from Roboflow, produce-specific keywords such as "apple," "banana," and "dragon fruit" were used to perform targeted searches on the Roboflow Universe platform. The search results returned a list of publicly available projects, from which suitable datasets were selected for download. By default, Roboflow datasets are distributed in pre-split form (train, validation, and test). For preprocessing, these splits were first merged into a single dataset to simplify later integration with ImageNet_good, where a new unified split would be generated after merging. All images were standardized to a resolution of 640×640. Manual inspection was then performed to ensure data quality and class consistency. Certain datasets included condition-based subclasses such as "rotten fruit" or "healthy fruit," originally intended to distinguish fruit states. In this study, such subclasses were consolidated into a single class only when favorable for the retail detection use case; otherwise, they were excluded from the final dataset. Roboflow annotations were verified for conformity to YOLO standards. It is found that some files containing segmentation polygons or multiple label regions in which decided to be removed using Python script.

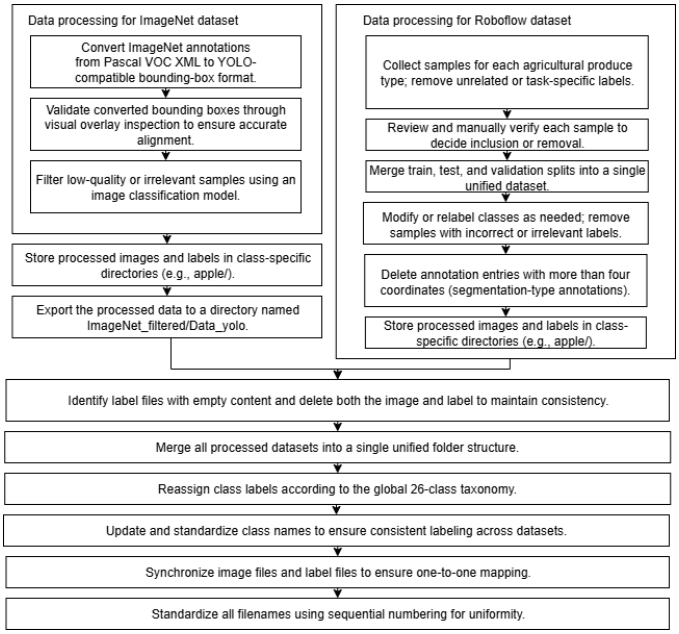

Fig. 3. Workflow of ImageNet and roboflow data preprocessing and unified post-processing to generate the final hybrid dataset for YOLO training.

After both streams were independently processed, they were merged through a unified post-processing pipeline designed to ensure dataset integrity and compatibility. Empty or mismatched label files were automatically identified and removed through filename mapping, while remaining image–label pairs were verified for one-to-one correspondence. Filenames were then standardized via sequential numbering, and directory hierarchies were normalized for uniform ingestion by the YOLO training engine. The resulting hybrid dataset comprised approximately 6 000 verified image samples, evenly distributed across 26 agricultural produce classes. Each sample adhered to a consistent annotation format and naming convention, enabling reproducible evaluation across YOLOv9, YOLOv10, and YOLOv11 variants. This corpus unifies ImageNet's lexical richness with Roboflow's contextual realism, forming a balanced, domain-specific dataset for training and subsequent performance analysis on the NVIDIA Jetson Xavier NX edge platform.

C. Model Training

The model training phase aimed to fine-tune state-of-the-art YOLO architectures for accurate and efficient agricultural produce detection under edge-deployment conditions. Training was carried out using the custom hybrid dataset (8450 images in 26 classes) described in Section VI. To ensure both reproducibility and computational efficiency, all experiments were executed on an NVIDIA RTX 3090 GPU (24 GB GDDR6X, 3840 CUDA cores) using the PyTorch-based Ultralytics YOLO framework. Each YOLO variant (YOLOv9, YOLOv10, YOLOv11) was initialized using pre-trained weights from the COCO dataset (80 classes). Transfer learning was applied by freezing early backbone layers during the initial epochs to preserve generic low-level feature

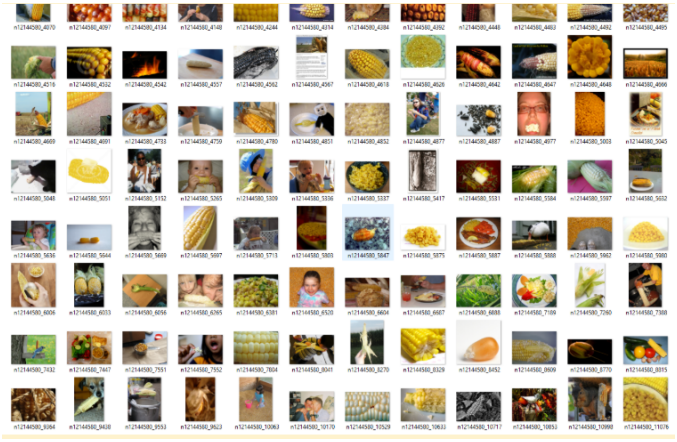

Fig. 4. Non-relevant ImageNet samples identified and removed through classification-based quality filtering using a ResNet-50 model.

representations while allowing full fine-tuning of detection heads. The classification head was modified to predict 26 target classes (nc = 26), replacing COCO's original output layer.

Optimizer selection was handled automatically by the YOLO framework, which initially evaluated both AdamW and SGD but converged on SGD with the parameters above. Internally, YOLO applied parameter grouping: 167 weights with no decay, 174 weights with 0.0005 decay, and 173 biases without decay—facilitating efficient convergence. Training logs confirmed stable loss reduction and consistent mAP gains across epochs.

D. Model Benchmark

To evaluate model performance, three YOLO variants—YOLOv9, YOLOv10, and YOLOv11—were selected for benchmarking. Each variant was tested across three model sizes (S, M, and L), resulting in a total of nine configurations. All models were exported using TensorRT to apply precision quantization and runtime optimization tailored for edge AI deployment. Although TensorRT supports execution on any NVIDIA GPU, the optimized engines it generates are device-specific, as calibration and kernel tuning depend on the underlying GPU architecture [25]. Therefore, to ensure consistent accuracy and latency measurements that accurately reflect real deployment conditions, both the creation of the TensorRT engine and the inference benchmarks were performed directly on the NVIDIA Jetson Xavier NX, which served as the target edge device in this study. Each model was evaluated using a standardized input resolution of 640×640 pixels, and accuracy was quantified through mean Average Precision (mAP) across 26 produce classes. Baseline mAP values were obtained in FP32 mode (without optimization and quantization), while the TensorRT-optimized models were re-evaluated to discover the potential variation in accuracy. Collectively, the mAP, latency, and FPS metrics were analyzed to highlight the trade-off between detection accuracy and real-time inference performance across all evaluated models.

Mean Average Precision (mAP) was computed at an Intersection-over-Union threshold of 0.5 following the PASCAL VOC standard (mAP@0.5). In addition to this widely used metric, we also report the COCO-style mAP@[0.5:0.95] and the F1 score to provide a comprehensive assessment of detection accuracy. This combination of metrics captures both localization and classification performance while balancing interpretability and computational cost. Inference speed was benchmarked by dividing the total time to process a fixed set of test images by the number of samples, yielding the average latency per image (ms/sample), and the reciprocal of this latency gave the throughput in frames per second (FPS). Reporting mAP@0.5, mAP@[0.5:0.95], F1 and FPS together enables a direct analysis of the trade-off between detection accuracy and real-time performance.

IV. Results and Discussion

A. Training Result

The hybrid dataset consisting of 8450 annotated images across 26 agricultural produce classes was divided into training (70%), validation (20%), and testing (10%) subsets, all standardized to a resolution of 640 × 640 pixels. Each YOLO model—v9, v10, and v11—was fine-tuned using transfer learning from COCO-pretrained weights, ensuring that early convolutional layers retained generic low-level representations while the detection head adapted to domain-specific features.

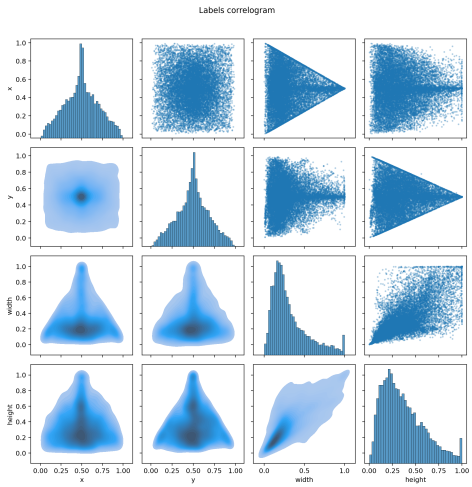

Fig. 5. Correlation matrix of normalized bounding box parameters (x, y, width, height) illustrating the distribution and relationships among annotation variables.

The label correlogram in Fig. 5 indicated that most object centers clustered around the image midpoint, corresponding to the typical placement of fruits on weighing trays. The width–height correlation revealed a narrow triangular distribution, suggesting scale uniformity across the dataset.

Training curves in Fig. 8, Fig. 7 and Fig. 6, collectively indicate that all YOLO variants achieved stable convergence with steadily decreasing box, classification, and distribution focal losses across epochs. The training performance comparison reveals a progressive improvement across YOLO generations. YOLOv9-C achieved its optimal accuracy at epoch 94 with a peak mAP@0.5:0.95 of 0.692, while YOLOv10-L required a longer convergence period, reaching a similar score at epoch 97 due to its deeper network complexity. In contrast, YOLOv11-L attained the highest mAP@0.5:0.95 of 0.700

at epoch 87, indicating faster and more stable optimization. The reduced validation losses and earlier convergence suggest that YOLOv11-L effectively balances learning stability and generalization, outperforming the preceding versions under identical training conditions.

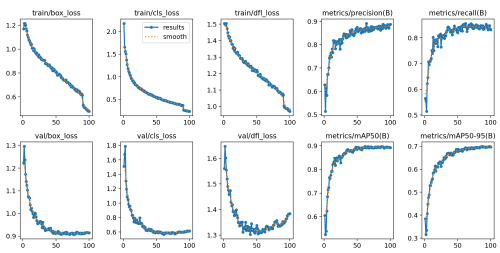

Fig. 6. YOLOv11 training result.

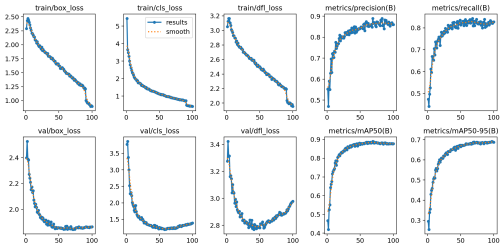

Fig. 7. YOLOv10 training result.

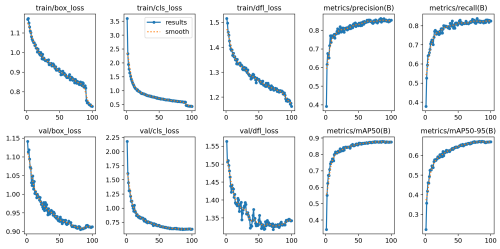

Fig. 8. YOLOv9 training result.

The confusion matrices of YOLOv9, YOLOv10, and YOLOv11 shown in Fig. 9, Fig. 10, Fig. 11, exhibit strong diagonal dominance, indicating generally high per-class classification accuracy. However, the distribution and magnitude of non-zero off-diagonal values reveal meaningful distinctions in class-specific robustness and visual separability. The confusion matrices reveal that YOLOv9-C produces the highest misclassification rate, with evident cross-class confusion among visually similar fruits such as apple–tomato and guava–rose apple, indicating weaker feature discrimination. In contrast, YOLOv10 displays the cleanest diagonal with minimal off-diagonal noise, reflecting stronger class separability and more stable optimization. YOLOv11 achieves comparable accuracy but introduces slight background leakage and minor confusion in similar-colour categories, suggesting broader generalization at a small precision cost. Overall, YOLOv10demonstrates the most balanced and precise classification performance, while Fig. 12 compares the PR characteristics of YOLOv9, YOLOv10, and YOLOv11 families across model scales. Fig. 12 compares the PR characteristics of YOLOv9, YOLOv10, and YOLOv11 families across model scales. All nine models exhibit strong detection capability, with mAP@0.5 values exceeding 0.85, reflecting robust precision–recall balance across the dataset. The smooth blue PR curves indicate stable

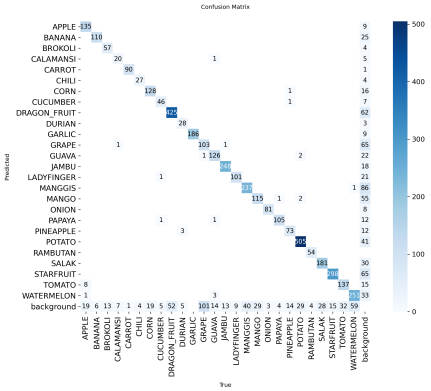

Fig. 9. YOLOv11 confusion matrix.

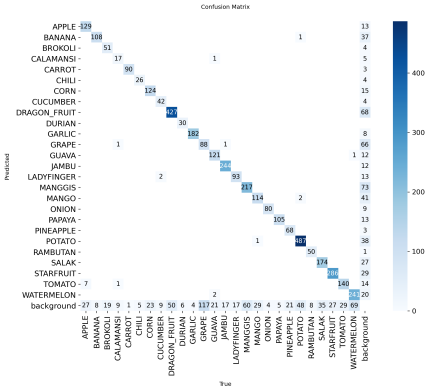

Fig. 10. YOLOv10 confusion matrix.

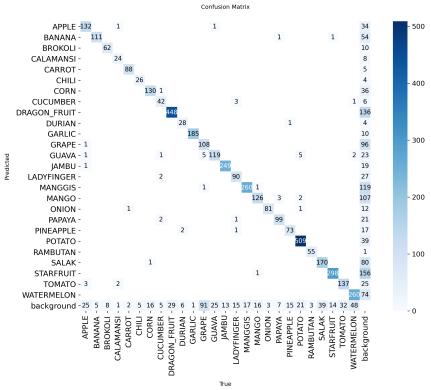

Fig. 11. YOLOv9 confusion matrix.

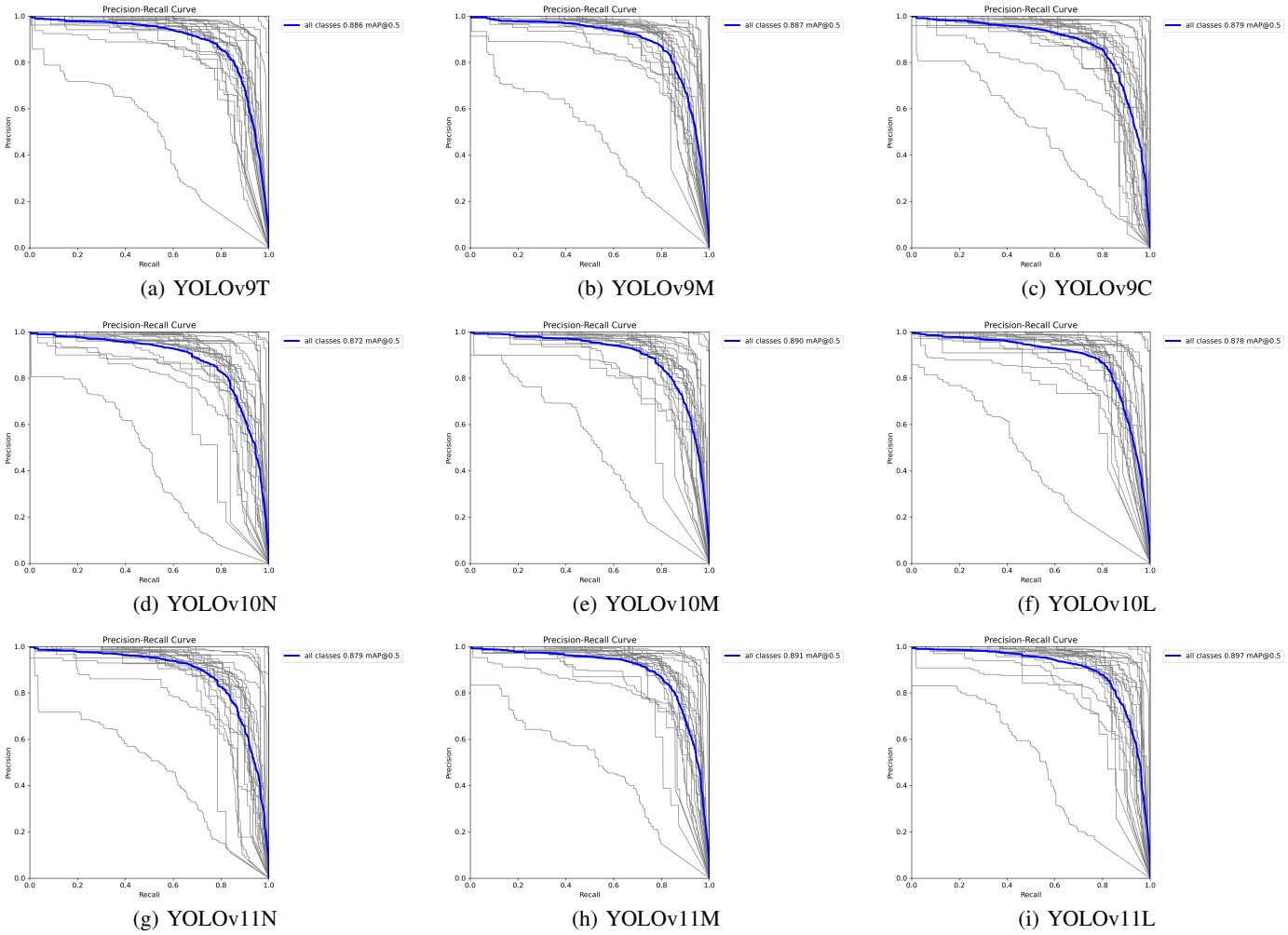

Fig. 12. Precision–Recall (PR) curves for YOLOv9/10/11 models at different scales (Tiny/Nano, Medium, and Compact/Large). Each sub-figure illustrates the trade-off between precision and recall across all classes, highlighting the relative stability and balance achieved by newer architectures.

confidence calibration, while the gray per-class traces reveal clear variation among object categories—showing that not all classes perform uniformly and at least one class remains challenging to detect. Among them, YOLOv11 variants display the most expansive PR envelopes, signifying superior precision–recall trade-offs and inter-class consistency. The YOLOv11L attains the highest mAP@0.5 of 0.897, only slightly above YOLOv11N and YOLOv11M; notably, YOLOv11 exhibits a consistent mAP increase with model size, unlike YOLOv9 and YOLOv10, implying that its additional parameters are effectively utilized for richer feature learning. Nevertheless, the smaller YOLOv11 models maintain comparable robustness to the larger variant, achieving near-identical detection quality with substantially lower computational cost. Overall, all models sustain commendable accuracy above the 0.8 mAP benchmark, with YOLOv11 offering the most balanced and scalable performance for real-time edge deployment.

B. Model Performance

The performance evaluation across 26 distinct produce classes as show in Table I demonstrated that model size is the dominant factor influencing both performance magnitude and consistency. As shown in Fig. 13, the Medium (M) and Large (L) or Complex (C) configurations consistently achieved the highest median mean Average Precision (mAP) scores across all three YOLO versions (v9, v10, v11). These models also exhibited the tightest interquartile ranges (IQR), indicating superior robustness and consistency in their detection accuracy across the diverse set of classes. Conversely, the smallest models (Tiny (T) for v9 and Nano (N) for v10/v11) demonstrated significantly lower median mAP and the widest score distributions. This suggests that while all three YOLO architectures are comparable, the minimal capacity of the smallest models leads to high performance variance and unreliable results on this dataset.

A granular analysis of the per-class mAP scores revealed a substantial degree of performance heterogeneity, ranging from near-perfect detection to critical failure points. The models achieved highly acceptable performance, exceeding $\mathbf{0.80}$ mAP, on visually distinct and easily segmentable classes. For instance, *Garlic* registered an average per-class mAP of $\mathbf{0.888}$, indicating that the model accurately localized and classified the object in $\mathbf{88.8}\%$ of the test cases when averaged across the nine evaluated configurations. *Carrot* demonstrated

TABLE I. PER-CLASS MAP ACROSS 26 CLASSES AND THREE YOLO FAMILIES (VALUES SHOWN AS S/M/L)

ID	Class	YOLOv9 (T/M/C)	YOLOv10 (N/M/L)	YOLOv11 (N/M/L)
C1	apple	0.58/0.58/0.61	0.63/0.63/0.61	0.59/0.65/0.63
C2	banana	0.79/0.80/0.77	0.73/0.79/0.79	0.77/0.78/0.80
C3	broccoli	0.65/0.58/0.59	0.63/0.58/0.59	0.61/0.62/0.62
C4	calamansi	0.72/0.74/0.67	0.57/0.71/0.62	0.64/0.67/0.66
C5	carrot	0.85/0.89/0.90	0.87/0.90/0.90	0.85/0.88/0.90
C6	chili	0.56/0.57/0.55	0.55/0.58/0.59	0.55/0.58/0.58
C7	corn	0.58/0.59/0.60	0.55/0.60/0.59	0.54/0.61/0.62
C8	cucumber	0.70/0.74/0.71	0.68/0.73/0.75	0.66/0.73/0.78
C9	dragon fruit	0.49/0.49/0.46	0.50/0.48/0.47	0.50/0.49/0.49
C10	durian	0.69/0.75/0.75	0.69/0.73/0.76	0.70/0.74/0.74
C11	garlic	0.87/0.90/0.90	0.88/0.89/0.89	0.88/0.89/0.89
C12	grape	0.32/0.32/0.31	0.27/0.35/0.32	0.30/0.34/0.33
C13	guava	0.65/0.69/0.70	0.68/0.68/0.69	0.69/0.70/0.70
C14	rose apple	0.50/0.54/0.54	0.52/0.54/0.53	0.52/0.54/0.55
C15	okra	0.58/0.65/0.63	0.59/0.62/0.64	0.59/0.60/0.63
C16	mangosteen	0.49/0.45/0.46	0.47/0.47/0.46	0.48/0.46/0.46
C17	mango	0.69/0.76/0.78	0.71/0.75/0.77	0.73/0.78/0.76
C18	onion	0.66/0.68/0.69	0.67/0.68/0.64	0.70/0.69/0.67
C19	papaya	0.80/0.79/0.80	0.80/0.81/0.81	0.80/0.82/0.83
C20	pineapple	0.73/0.77/0.76	0.74/0.77/0.77	0.73/0.75/0.74
C21	potato	0.80/0.79/0.80	0.79/0.78/0.80	0.80/0.81/0.82
C22	rambutan	0.75/0.84/0.82	0.78/0.85/0.82	0.81/0.79/0.81
C23	snake fruit	0.73/0.85/0.87	0.82/0.85/0.84	0.79/0.85/0.86
C24	starfruit	0.74/0.81/0.82	0.77/0.80/0.81	0.74/0.80/0.82
C25	tomato	0.77/0.80/0.78	0.78/0.77/0.79	0.74/0.80/0.79
C26	watermelon	0.70/0.72/0.73	0.69/0.73/0.72	0.68/0.74/0.72

similarly strong results, with a per-class mAP $\approx$ **0.882**. However, this success contrasts sharply with the challenges posed by other items. The class *Grape* was identified as a universal failure point, registering a critically low per-class mAP $\approx$ **0.318** across all model configurations. This extreme outlier—visually represented by the lowest whisker in every box plot (Fig. 13)—indicates a fundamental difficulty that is not mitigated by increasing model complexity or upgrading the YOLO version. Other low-performing classes included *Manggis* (per-class mAP $\approx$ **0.467**) and *Dragon Fruit* (per-class mAP $\approx$ **0.486**).

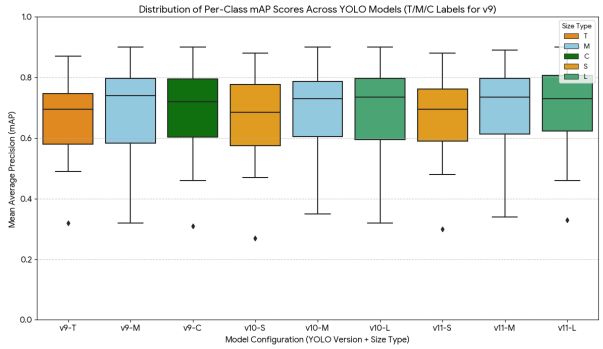

Fig. 13. Distribution of Per-Class mAP scores across YOLO models and sizes. YOLOv9 uses Tiny (T), Medium (M), and Compact (C) labels, while YOLOv10 and YOLOv11 use Nano (N), Medium (M), and Large (L).

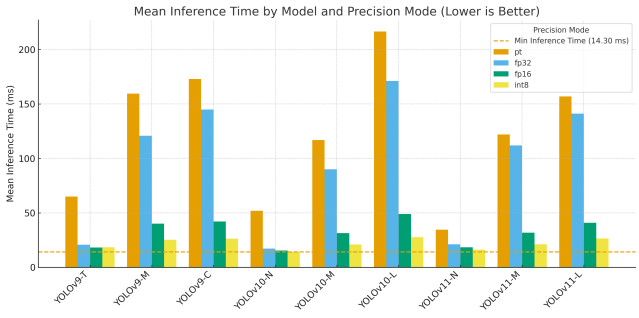

Fig. 14. Total inference time across YOLOv9, YOLOv10, and YOLOv11 models under different precision modes (`pt`, `fp32`, `fp16`, and `int8`). The dashed red line indicates the global minimum total inference time. Image resolution is fixed at 640×640.

C. Inference Speed Analysis

The inference–speed analysis based on the collected data as shown in Table II reveals that pre–processing and post–processing contribute little to overall latency: mean pre-processing times stay between 4.13 and 4.89 ms, while post-processing ranges from about 2.73 to 11.93ms . Mean inference times, however, vary by more than an order of magnitude. On NVIDIA Jetson, the YOLOv10–nano (N) model converted to TensorRT and quantized to FP16 or INT8 delivers the fastest throughput, completing detection in roughly 14–15ms per image and achieving 18.0–18.9 frames per second. By contrast, the unoptimized PyTorch large (L) models require about 216 ms for inference and thus cannot meet real-time requirements. TensorRT optimization reduces latency dramatically; for example, converting the YOLOv10-L model from PyTorch to TensorRT with FP16 shrinks inference time from $\approx$ 216ms to about 49ms, and further to $\approx$ 28 ms with INT8 quantization . These results show that model size and numerical precision are the primary determinants of inference speed, while the pipeline's fixed overhead remains minor. As illustrated in Fig. 14, precision reduction consistently lowers total inference time across all YOLO generations evaluated, with INT8 quantization yielding the global minimum latency.

A closer examination of post-processing reveals that YOLOv11 exhibits consistently higher latency than YOLOv10, especially in its Nano variants. This behaviour stems from architectural differences: YOLOv11 employs a traditional *Detect head* that performs Non-Maximum Suppression (NMS) to filter redundant bounding boxes, whereas YOLOv10 is *NMS-free* due to its dual-assignment training strategy that inherently mitigates duplicate detections. The additional NMS step in YOLOv11 adds computational overhead during output decoding, explaining why models like YOLOv11-N INT8 spend nearly three times longer in post-processing compared with the NMS-free YOLOv10 equivalents. Consequently, YOLOv10 maintains smoother end-to-end latency for real-time agricultural produce detection.

D. Accuracy Analysis

Table III provides a comprehensive comparison of YOLOv9–YOLOv11 models across Nano, Medium and Large scales and four precision modes (native, FP32, FP16 and INT8), summarising precision, recall, F1-score and mAP. The

TABLE II. INFERENCE SPEED ACROSS CONFIGURATIONS AT INPUT 640 × 640. TIMES IN MILLISECONDS (MS). BEST TOTAL LATENCY (**LOWER IS BETTER**) AND BEST FPS (**HIGHER IS BETTER**) ARE BOLDED PER MODEL SIZE

Family	Size	Precision	mean_pre	mean_infer	mean_post	total (mean)	median	FPS (mean)	FPS (median)
YOLOv10									
YOLOv10	L	native	4.471	216.482	3.011	259.909	207.166	3.847	4.827
YOLOv10	L	fp32	4.246	171.214	3.449	213.103	185.447	4.693	5.392
YOLOv10	L	fp16	4.361	49.036	3.272	90.932	60.795	10.997	16.449
YOLOv10	L	int8	4.227	27.663	3.214	**68.423**	38.935	**14.615**	25.684
YOLOv10	M	native	4.249	116.936	2.900	154.087	117.947	6.490	8.478
YOLOv10	M	fp32	4.293	89.927	3.298	131.295	102.701	7.616	9.737
YOLOv10	M	fp16	4.212	31.324	3.268	72.470	42.643	13.799	23.450
YOLOv10	M	int8	4.225	21.007	3.172	**57.606**	32.055	**17.359**	31.196
YOLOv10	N	native	4.300	51.822	2.732	93.256	40.570	10.723	24.649
YOLOv10	N	fp32	4.261	17.160	3.143	57.671	27.796	17.340	35.976
YOLOv10	N	fp16	4.602	15.416	3.238	**53.022**	26.348	**18.860**	37.954
YOLOv10	N	int8	4.559	14.304	3.169	55.403	25.351	18.049	39.446
YOLOv11									
YOLOv11	L	native	4.130	157.032	11.327	195.890	179.216	5.105	5.580
YOLOv11	L	fp32	4.351	141.207	11.929	192.783	162.635	5.187	6.149
YOLOv11	L	fp16	4.243	40.765	11.230	90.160	58.489	11.091	17.097
YOLOv11	L	int8	4.291	26.506	4.817	**59.563**	38.623	**16.789**	25.891
YOLOv11	M	native	4.268	122.071	11.329	172.209	146.175	5.807	6.841
YOLOv11	M	fp32	4.482	112.014	11.565	152.633	130.541	6.552	7.660
YOLOv11	M	fp16	4.187	31.702	11.250	70.367	49.260	14.211	20.300
YOLOv11	M	int8	4.315	21.196	10.591	**60.103**	38.678	**16.638**	25.855
YOLOv11	N	native	4.275	34.536	10.460	73.475	46.990	13.610	21.281
YOLOv11	N	fp32	4.520	21.147	11.089	60.639	39.233	16.491	25.489
YOLOv11	N	fp16	4.832	18.361	11.140	58.401	36.706	17.123	27.243
YOLOv11	N	int8	4.890	16.109	11.201	**56.130**	35.009	**17.816**	28.564
YOLOv9									
YOLOv9	C	native	4.239	172.955	11.345	223.763	195.335	4.469	5.119
YOLOv9	C	fp32	4.342	144.871	11.806	185.208	166.672	5.399	6.000
YOLOv9	C	fp16	4.301	42.021	10.846	80.989	59.686	12.347	16.754
YOLOv9	C	int8	4.206	26.214	10.596	**64.149**	43.787	**15.589**	22.838
YOLOv9	M	native	4.268	159.408	11.193	204.774	156.395	4.883	6.394
YOLOv9	M	fp32	4.432	120.861	11.794	161.623	143.076	6.187	6.989
YOLOv9	M	fp16	4.209	40.107	10.635	78.265	57.794	12.777	17.303
YOLOv9	M	int8	4.209	25.219	9.895	**62.674**	42.060	**15.956**	23.776
YOLOv9	T	native	4.430	64.900	11.028	109.722	73.024	9.114	13.694
YOLOv9	T	fp32	4.213	20.625	11.131	59.235	38.176	16.882	26.194
YOLOv9	T	fp16	4.541	18.197	11.579	58.217	36.669	17.177	27.271
YOLOv9	T	int8	4.527	18.409	11.071	**57.562**	36.737	**17.372**	27.220

accuracy evaluation across the YOLOv9–YOLOv11 families reveals several notable trends. Among all tested configurations, the YOLOv11-L model in PyTorch baseline achieves the highest performance with mAP@0.5 $= 0.896$ and an F1 $= 0.86$, surpassing YOLOv10 and YOLOv9 counterparts and confirming YOLOv11's architectural advantage. Within the YOLOv10 family, the medium (M) variant yields slightly better accuracy (F1 ≈ 0.848) than both the large (0.845) and nano (0.837) models, suggesting that accuracy saturates beyond a certain parameter count and that mid-sized models strike the best balance between complexity and learning stability. Precision-mode comparison further indicates that half-precision (FP16) inference is effectively lossless relative to FP32, with identical F1 values around 0.842 across all variants. These results affirm that FP16 maintains numerical fidelity while reducing memory usage—a desirable property for embedded deployment. Conversely, INT8 quantisation introduces a moderate but consistent performance degradation of about 3–4 percentage points in F1. For example, YOLOv10-L drops from 0.845 (PyTorch) to 0.811 (INT8), and YOLOv10-M declines from 0.848 to 0.806, primarily due to reduced numerical precision during activation and weight representation.

A deeper inspection of model sensitivity highlights that smaller models exhibit superior robustness to INT8 quantisation. The YOLOv10-N variant shows only a 1.8 percentage-point F1 loss ($0.832 \rightarrow 0.814$), whereas the large variant loses more than 3 points. This observation aligns with quantisation theory: lightweight architectures possess fewer convolutional layers and a narrower dynamic range of activations, resulting in lower quantisation error propagation. Larger networks, with deeper hierarchies and wider feature channels, amplify rounding and clipping effects across multiple layers—explaining their steeper accuracy decline when compressed to 8-bit precision. Empirically, INT8 impacts recall more strongly than precision (e.g., YOLOv10-L recall $0.825 \rightarrow 0.771$, precision $0.865 \rightarrow 0.856$), indicating that quantised models tend to miss small or ambiguous detections rather than misclassify them.

Nonetheless, the overall degradation remains modest, and FP16 retains nearly identical accuracy to FP32 while achieving

TABLE III. COMPREHENSIVE COMPARISON OF YOLOV9–YOLOV11 UNDER TENSORRT PRECISION MODES (PT, FP32, FP16, INT8)

Family	Size	Precision	Recall	Precision	F1	mAP@0.5:0.95	mAP@0.5
YOLOv10							
YOLOv10	N	native	0.813	0.863	0.837	0.668	0.873
YOLOv10	N	fp32	0.808	0.859	0.833	0.654	0.865
YOLOv10	N	fp16	0.807	0.858	0.832	0.654	0.865
YOLOv10	N	int8	0.776	0.855	0.814	0.629	0.843
YOLOv10	M	native	0.819	0.879	0.848	0.694	0.890
YOLOv10	M	fp32	0.813	0.869	0.840	0.679	0.876
YOLOv10	M	fp16	0.812	0.868	0.839	0.679	0.876
YOLOv10	M	int8	0.778	0.835	0.806	0.640	0.837
YOLOv10	L	native	0.825	0.865	0.845	0.692	0.878
YOLOv10	L	fp32	0.815	0.871	0.842	0.678	0.875
YOLOv10	L	fp16	0.815	0.870	0.842	0.678	0.875
YOLOv10	L	int8	0.771	0.856	0.811	0.653	0.848
YOLOv11							
YOLOv11	N	native	0.823	0.868	0.845	0.671	0.879
YOLOv11	N	fp32	0.804	0.867	0.835	0.651	0.865
YOLOv11	N	fp16	0.805	0.868	0.835	0.651	0.865
YOLOv11	N	int8	0.786	0.837	0.811	0.617	0.837
YOLOv11	M	native	0.840	0.870	0.855	0.696	0.891
YOLOv11	M	fp32	0.819	0.868	0.843	0.679	0.882
YOLOv11	M	fp16	0.819	0.867	0.843	0.679	0.882
YOLOv11	M	int8	0.774	0.870	0.819	0.638	0.843
YOLOv11	L	native	0.844	0.877	0.860	0.699	0.896
YOLOv11	L	fp32	0.849	0.849	0.849	0.680	0.883
YOLOv11	L	fp16	0.849	0.849	0.849	0.680	0.883
YOLOv11	L	int8	0.780	0.839	0.809	0.628	0.840
YOLOv9							
YOLOv9	T	native	0.820	0.858	0.839	0.671	0.879
YOLOv9	T	fp32	0.807	0.850	0.828	0.655	0.867
YOLOv9	T	fp16	0.808	0.850	0.828	0.655	0.867
YOLOv9	T	int8	0.797	0.835	0.815	0.641	0.851
YOLOv9	M	native	0.823	0.877	0.849	0.695	0.887
YOLOv9	M	fp32	0.825	0.859	0.842	0.680	0.875
YOLOv9	M	fp16	0.826	0.858	0.842	0.680	0.875
YOLOv9	M	int8	0.768	0.846	0.805	0.631	0.832
YOLOv9	C	native	0.831	0.873	0.852	0.692	0.885
YOLOv9	C	fp32	0.835	0.853	0.844	0.674	0.872
YOLOv9	C	fp16	0.836	0.854	0.845	0.674	0.872
YOLOv9	C	int8	0.815	0.807	0.811	0.619	0.836

significantly faster inference. Therefore, for real-time agricultural produce detection on resource-limited hardware, FP16-based TensorRT models or Nano-INT8 variants offer the best trade-off, balancing accuracy, efficiency, and deployment feasibility.

E. Inference Speed and Accuracy Tradeoff

The mAP–latency plot zs shown in Fig. 15 clearly reveals a Pareto frontier where only a handful of model–precision combinations achieve both high accuracy and real-time inference speeds. At one end of the spectrum, the smallest Nano configurations (e.g., YOLOv10-N FP16/FP32 and YOLOv11-N FP16) produce mAP@0.5 values around 0.865 while completing the full pre-processing, inference, and post-processing pipeline in approximately 53–58 ms. Their INT8 counterparts reduce the mean inference time by a few milliseconds but at the cost of a small drop in recall and overall mAP, typically slipping to ≈ 0.84. These Nano models form the left-most portion of the Pareto curve and thus represent the optimal choice when deploying to constrained edge devices that demand sub-60 ms response times.

Moving up the frontier, the Medium FP16 configuration of YOLOv11 stands out as a compelling compromise. It boosts accuracy to ≈ 0.88 mAP@0.5 yet keeps total latency below about 70 ms, only 15–20 ms slower than the Nano models but delivering a notable increase in detection fidelity. By contrast, the Large FP16/FP32 variants of YOLOv10 and YOLOv11 push accuracy marginally higher (to ≈ 0.896 mAP@0.5) but at the cost of doubling the response time to 90–120 ms, pushing them off the Pareto frontier and making them impractical for real-time field use. In summary, the trade-off analysis underscores that half-precision and medium-sized YOLOv11

offer the best balance when accuracy is paramount, while Nano FP16/INT8 models remain the top recommendation for latency-critical deployments.

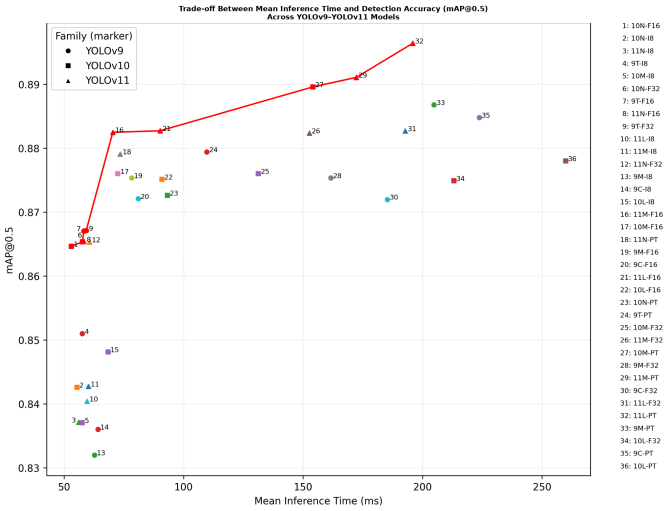

Fig. 15. Trade-off between mean inference time and detection accuracy (mAP@0.5) across YOLOv9–YOLOv11 models. The plot shows each model's mean inference time (ms) on the x-axis and its detection accuracy (mAP@0.5) on the y-axis, with distinct markers for the YOLOv9 (circle), YOLOv10 (square), and YOLOv11 (triangle) families. The red line traces the Pareto front of the speed–accuracy trade-off.

F. Qualitative Analysis

To test real-world robustness, the INT8-quantized YOLOv11-Nano model was deployed on unseen footage and custom images as shown in Fig. 16 and Fig. 17. We downloaded market scenes from YouTube and also purchased several types of produce and recorded them under natural indoor light. The model ran at real-time speed on the Jetson Xavier NX and able to detect the item with accurate bounding boxes and labels. These qualitative observations confirm that the model generalises well beyond the training dataset and complements the quantitative results, demonstrating readiness for deployment in practical retail environments.

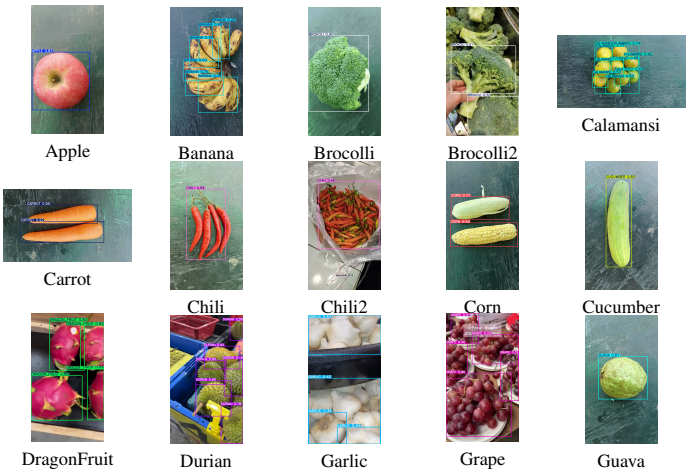

Fig. 16. YOLOv11n INT8 qualitative results — Panel (A), samples 1–15.

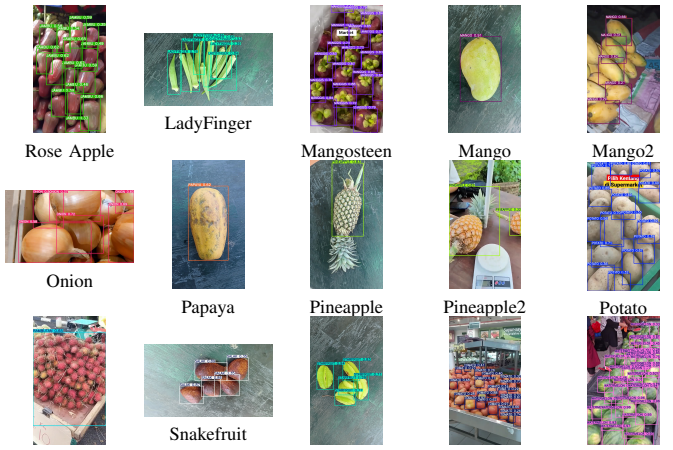

Fig. 17. YOLOv11n INT8 qualitative results — Panel (B), samples 16–30.

V. Conclusion

This study demonstrates that real-time visual recognition of loose fresh produce can be effectively integrated into intelligent retail weighing systems to reduce reliance on manual PLU-based item selection. By enabling accurate produce identification within a single weighing operation, the proposed Edge-AI approach directly addresses a key bottleneck in retail checkout workflows, namely slow and error-prone manual product lookup. To this end, an Edge-AI–based fresh-produce weighing system was developed and evaluated by integrating YOLOv9, YOLOv10, and YOLOv11 detectors with transfer learning on a 26-class Malaysian produce dataset comprising 8,450 annotated images.All models converged smoothly under identical training regimes; YOLOv11-Large achieved the highest mAP@0.5:0.95 of 0.699, YOLOv10 exhibited the strongest per-class discrimination, and YOLOv9 showed more misclassifications among visually similar produce. Across the dataset, each model maintained a mean Average Precision above 0.8, with medium or large configurations providing the most consistent accuracy. Pre- and post-processing overheads remain small relative to inference for large models (4–12 ms), but become proportionally significant as models shrink; moreover, YOLOv11's reliance on a Non-Maximum Suppression (NMS) head results in noticeably longer post-processing than YOLOv10's NMS-free design. TensorRT optimization with FP16/INT8 quantization reduced YOLOv10-Large inference latency from approximately 216 ms to 49 ms (FP16) and 28 ms (INT8). Among the tested configurations, YOLOv10-Medium and YOLOv10-Nano offer the best balance between accuracy and real-time performance, achieving roughly 14–15 ms per image (≈ 18–19 FPS) while maintaining high precision.

Inference-speed analysis demonstrated that pre- and post-processing overheads are negligible relative to the actual network inference time. With TensorRT optimization and FP16/INT8 quantization, the YOLOv10-Nano configuration achieved real-time performance at roughly 14–15 ms per image (≈ 18–19 FPS). The YOLOv10-Large model, once converted from unoptimized PyTorch to TensorRT, saw its inference latency shrink from ≈ 216ms to ≈ 49ms with FP16 and ≈ 28ms with INT8. Although YOLOv11 models offered

slightly higher precision, their reliance on a Non-Maximum Suppression head introduced additional post-processing delay compared with the NMS-free dual-assignment strategy of YOLOv10. These results show that YOLOv10-Medium and YOLOv10-Nano strike the best balance between detection accuracy and real-time inference, making them suitable for deployment on resource-constrained retail scales.

Although this system demonstrates that high-accuracy object detection is feasible on embedded GPUs, several challenges remain.

- Intraclass and interclass similarity: Some classes (e.g., grapes, mangosteen, dragon fruit) exhibit low per-class mAP due to high intra-class variation and inter-class similarity. Future work should augment the dataset and explore model-level innovations to improve discrimination among similar-looking items.
- Species and cultivar detection: The current dataset treats each fruit or vegetable as a single class and does not include species- or cultivar-level distinctions (e.g., different varieties of oranges). Even if such data were added, it remains uncertain whether the current YOLO architectures could fully capture the subtle visual variations between closely related cultivars. Future work should therefore combine dataset expansion with model adaptations or fine-grained classification strategies to assess whether detection at this granularity is feasible.
- NMS-free architectures: YOLOv11's NMS-based post-processing introduces significant latency; integrating next-generation NMS-free models such as the forthcoming YOLO26—designed as an end-to-end detector without a non-maximum suppression step [26]—could further reduce inference time and simplify deployment.
- Data and modality expansion: Additional future work includes expanding the dataset to cover occlusions, mixed produce and seasonal variations, incorporating multi-modal sensors (weight, depth, spectral) to distinguish visually similar items, applying model pruning and dynamic quantization, developing incremental learning for new classes, and evaluating deployment on alternative edge devices.

Overall, the findings confirm that high-accuracy object detection on embedded GPUs is not only technically feasible but also practically deployable in real retail environments. By combining NMS-free architectures with TensorRT optimization, the proposed system supports low-latency inference suitable for unattended or semi-automated checkout scenarios, reducing operator workload and minimizing customer waiting time. These results provide a concrete pathway toward scalable, PLU-independent intelligent weighing systems in modern retail settings.

ACKNOWLEDGMENT

This paper is supported by grant ZG-2025-018 from the Institute of Visual Informatics (IVI), Universiti Kebangsaan Malaysia (UKM).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author.

REFERENCES

[1] S. Greenwood and C. Reynolds, "Fresh produce on the loose: examining the coherence between plastic packaging and food waste policy using the case study of fruit and vegetables in the UK," *Environmental Research: Food Systems*, vol. 1, no. 2, Art. no. 025008, Nov. 2024.

[2] J.-K. Chaw and M. Mokji, "Analysis of produce recognition system with taxonomist's knowledge using computer vision and different classifiers," *IET Image Processing*, vol. 11, no. 3, pp. 173–182, 2017.

[3] R. Singh and S. S. Gill, "Edge AI: A survey," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023.

[4] C. Silvano *et al.*, "A survey on deep learning hardware accelerators for heterogeneous HPC platforms," *ACM Computing Surveys*, vol. 57, no. 11, Art. no. 286, Nov. 2025.

[5] S. Y. Mohammed, "Architecture review: Two-stage and one-stage object detection," *Franklin Open*, vol. 12, Art. no. 100322, 2025.

[6] M. L. Ali and Z. Zhang, "The YOLO framework: A comprehensive review of evolution, applications, and benchmarks in object detection," *Computers*, vol. 13, no. 12, Art. no. 336, 2024.

[7] T. Diwan, G. Anirudh, and J. V. Tembhurne, "Object detection using YOLO: challenges, architectural successors, datasets and applications," *Multimedia Tools and Applications*, vol. 82, pp. 9243–9275, Mar. 2023.

[8] E. Pandilova *et al.*, "Transfer learning with YOLO for object detection in remote sensing," in *ICT Innovations 2024. TechConvergence: AI, Business, and Startup Synergy*, B. Risteska Stojkoska and S. Janeska Sarkanjac, Eds. Cham, Switzerland: Springer Nature, 2025, pp. 121–135.

[9] H. Zhai, J. Du, Y. Ai, and T. Hu, "Edge deployment of deep networks for visual detection: A review," *IEEE Sensors Journal*, vol. 25, no. 11, pp. 18662–18683, 2025.

[10] H. M. Mohan, S. Anitha, R. Chai, and S. H. Ling, "Edge artificial intelligence: Real-time noninvasive technique for vital signs of myocardial infarction recognition using Jetson Nano," *Advances in Human-Computer Interaction*, vol. 2021, no. 1, Art. no. 6483003, 2021.

[11] D. Berardini, L. Migliorelli, A. Galdelli, and M. J. Marín-Jiménez, "Edge artificial intelligence and super-resolution for enhanced weapon detection in video surveillance," *Engineering Applications of Artificial Intelligence*, vol. 140, Art. no. 109684, 2025.

[12] R. A. Amin, M. Hasan, V. Wiese, and R. Obermaisser, "FPGA-based real-time object detection and classification system using YOLO for edge computing," *IEEE Access*, vol. 12, pp. 73268–73278, 2024.

[13] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, "Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application," *IEEE Access*, vol. 8, pp. 9102–9114, 2020.

[14] C. Dhasarathan *et al.*, "TensorRT optimized driver drowsiness detection system using edge device," *Ain Shams Engineering Journal*, vol. 16, no. 10, Art. no. 103620, 2025.

[15] X. Zhang and B. Li, "Tennis ball detection based on YOLOv5 with TensorRT," *Scientific Reports*, vol. 15, no. 1, Art. no. 21011, Jul. 2025.

[16] S. Wen *et al.*, "PcMNet: An efficient lightweight apple detection algorithm in natural orchards," *Smart Agricultural Technology*, vol. 9, Art. no. 100623, 2024.

[17] A. Crespo, C. Moncada, F. Crespo, and M. E. Morocho-Cayamcela, "An efficient strawberry segmentation model based on Mask R-CNN and TensorRT," *Artificial Intelligence in Agriculture*, vol. 15, no. 2, pp. 327–337, 2025.

[18] Z. Lv *et al.*, "Efficient deployment of peanut leaf disease detection models on edge AI devices," *Agriculture*, vol. 15, no. 3, Art. no. 332, 2025.

[19] P. Hidayatullah, N. Syakrani, M. R. Sholahuddin, T. Gelar, and R. Tubagus, "YOLOv8 to YOLO11: A comprehensive architecture in-depth comparative review," *arXiv preprint arXiv:2501.13400*, 2025.

[20] R. Bumbálek *et al.*, "Computer vision in precision livestock farming: benchmarking YOLOv9, YOLOv10, YOLOv11, and YOLOv12 for individual cattle identification," *Smart Agricultural Technology*, vol. 12, Art. no. 101208, 2025.

[21] P. Baglat *et al.*, "Comparative analysis and evaluation of YOLO generations for banana bunch detection," *Smart Agricultural Technology*, vol. 12, Art. no. 101100, 2025.

[22] P. Kamat, S. Gite, H. Chandekar, L. Dlima, and B. Pradhan, "Multi-class fruit ripeness detection using YOLO and SSD object detection models," *Discover Applied Sciences*, vol. 7, no. 9, Art. no. 931, Aug. 2025.

[23] A. Sharma, V. Kumar, and L. Longchamps, "Comparative performance of YOLOv8, YOLOv9, YOLOv10, YOLOv11 and Faster R-CNN models for detection of multiple weed species," *Smart Agricultural Technology*, vol. 9, Art. no. 100648, 2024.

[24] H. He, O. M. Lawal, Y. Tan, and K. Cheng, "An anchor-based YOLO fruit detector developed on YOLOv5," *PLOS ONE*, vol. 20, no. 9, pp. 1–14, Sep. 2025.

[25] O. Shafi, C. Rai, R. Sen, and G. Ananthanarayanan, "Demystifying TensorRT: Characterizing neural network inference engine on Nvidia edge devices," in *Proc. IEEE Int. Symp. on Workload Characterization (IISWC)*, 2021, pp. 226–237.

[26] R. Sapkota and M. Karkee, "Ultralytics YOLO Evolution: An overview of YOLO26, YOLO11, YOLOv8 and YOLOv5 object detectors for computer vision and pattern recognition," arXiv preprint arXiv:2510.09653, 2025.