Sustainable Manufacturing Application of Embedded Learning Algorithms for Vision-based Defect Detection under the Industry 5.0 Paradigm

Mihai-Daniel Pavel
Automation and Industrial Informatics
University Politehnica of Bucharest
Bucharest, Romania
mihai_daniel.pavel@upb.ro

Grigore Stamatescu
Automation and Industrial Informatics
University Politehnica of Bucharest
Bucharest, Romania
grigore.stamatescu@upb.ro

***Abstract*—The improvement of flexible manufacturing systems towards sustainable use of raw materials and increased resource efficiency represents a core tenant of Industry 5.0 competitiveness. This can be currently achieved through the adoption and accelerated implementation of state-of-the-art artificial intelligence models in forecasting, anomaly detection and classification applications. Human-centric approaches balance the deployment and implementation models for control and cognition with socially relevant goals for increased resilience. The paper presents and embedded learning application for vision-based defect detection on a five-station connected laboratory flexible manufacturing line. Quantitative results are illustrated and discussed that comparatively benchmark multiple generations of the YOLO real-time object detection model family along with implementation considerations and integration aspects with industrial automation technology.**

***Index Terms*—sustainable manufacturing, embedded learning, ai, defect classification, industry 5.0.**

I. Introduction

The concept of connected Industry 4.0 has been extended to include three pillars of sustainability, human centricity and resilience under the Industry 5.0 paradigm [1]. Focusing on sustainability, the development of intelligent systems in manufacturing can bring a significant contribution to reduce raw material and process energy waste by early detection and classification of product defects. Vision-based inspection solutions leverage advanced AI models for identifying, classifying and labeling defect product parts in real time. The main requirements for a successful implementation of such models imply a suitable understanding of the manufacturing process and domain-specific adaptation of the AI models for robust performance under closed-loop control constraints.

In this context we introduce our work that considers the application of various YOLO-family models and benchmarking their performance on three different hardware platform for a flexible assembly line experimental system. Unlike prior work that requires extensive dataset collection and model retraining for industrial applications, this study demonstrates the feasibility of using pre-trained YOLO models to achieve robust defect detection in a five-station manufacturing line, minimizing setup time and data requirements. The system illustrated in Figure 1 represents a five-stage process that includes the following operations: base plate supply, bottom piece assembly, collaborative robot handling and visual inspection station, top part assembly and storage station.

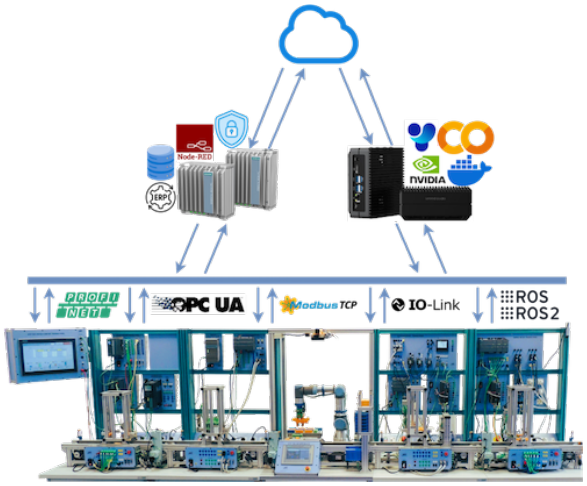


Fig. 1: Advanced flexible assembly line architecture [2]

The main contributions of this paper are argued to be as follows:

- Benchmarking various YOLO-family object detection models on multiple hardware platforms in a manufacturing context;
- Integration and deployment of the solution with industry-grade components and systems.

The rest of the paper is structured as follows. Section II frames our work into the scientific context of embedded industrial AI applications. Section III describes the methodology used for the experimental research, including the performance metrics for benchmarking and collected image datasets. Section IV extensively presents the achieved results on three hardware platforms with multiple generations of the YOLO deep learning models for object detection and classification.

Section V concludes the paper with outlook on deployment of the current solution.

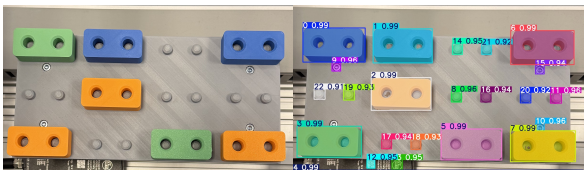
II. Related Work

Both industrial revolutions that started the concepts of Industry 4.0 and Industry 5.0 were deeply documented in [3] and [4]. [5] developed a deep Convolutional Neural Network (CNN) for defect detection for a reliable industrial inspection system. Another research in this field is [6] where a deep learning model was presented for surface anomaly detection using mixed supervision. The advanced architectures of YOLO are available as open-source software [7] for further development and new research in this field. In his work [8] researched a variant that uses programmable gradient information (PGI) to resolve the data loss of other state-of-the-art models. Advanced models like Segment Anything [9] for image segmentation applications can work seamlessly with object detection algorithms as it helps with initial preparation of data sets. The main objective of developing a new model is the ability to work as fast as possible without losing accuracy. [10] presented in his work a model that is 50 times faster than its predecessor, with even better accuracy. Other models focus on open-vocabulary detection, [11], exploring prompt-then-detect paradigms. This high volume of models and methods brings about the need to explore the differences between them, which are the strengths and weaknesses of one model compared to another. An example of a paper exploring this aspect is [12] which provides an overview of the most relevant evaluation methods used in object detection competitions. Previous work [13] demonstrated the effectiveness of one such approach in conjunction with reference performance metrics validated through experiments on a real-world flexible assembly system with state-of-the-art components and tools. In [14] an application of reinforcement learning with vision-based object detection for mobile robot control is presented.

III. Methodology

The application developed for the flexible assembly line system is divided into several parts: raw material monitoring and identification in local warehouses to inform the user of the production capacity (Figure 2), error detection in the robotic cell station (Figure 3), quality control of the final product (Figure 4). The system employs pre-trained YOLO-based object detection models to perform tasks such as object detection, instance segmentation, and image classification. These tasks are critical in industrial automation for inspection stations, intelligent monitoring, and error correction. The application detects five classes of objects: raw material pieces (by type and color), empty warehouse spaces, assembled components, defective parts, and final product configurations.

YOLOv8 employs a CSPDarknet53 backbone with approximately 20–53 convolutional layers, processing 640x640 RGB input images to output bounding boxes, objectness scores, and class probabilities via an anchor-free head [15]; compared to YOLOv9's Generalized Efficient Layer Aggregation (GELAN) backbone with Programmable Gradient Information (PGI) for enhanced gradient flow, YOLOv10's NMS-free dual-assignment strategy with lightweight classification heads, and YOLOv11's C3K2 and C2PSA blocks for improved feature extraction, YOLOv8 is less computationally efficient but simpler. Standard training hyperparameters include a batch size of 16, initial learning rate of 0.01 (SGD) or 0.001 (Adam), momentum of 0.9, and 100–300 epochs with cosine learning rate scheduling.



(a) Raw image of the warehouse (b) Detected warehouse

Fig. 2: Identification of raw materials and free spaces in the warehouse

Figure 2a shows the raw image captured by a camera positioned above the warehouse. Using video feeds and a pre-trained YOLOv8 model with its default architecture, including a backbone of convolutional layers as described in [7], the system identifies available pieces and empty spaces, as shown in Figure 2b. Each detected element is assigned a unique identification number, a bounding box, and object contours, with color recognition for assembly purposes. This enables the industrial robot to select the correct piece based on order specifications and supports warehouse management by identifying empty spaces for carrier robots.

Model performance is evaluated using standard metrics, suggested in [16]: Intersection over Union (IoU), Precision, Recall, and Mean Average Precision (mAP) at IoU thresholds of 0.50 (mAP50) and 0.50–0.95 (mAP50–95). These metrics ensure accurate detection and robustness under varying lighting conditions. Enhancing the box alignment of the detected objects in the image, YOLOv8 uses Complete Intersection over Union (CIoU) [17] calculated with the following formula:

$$L_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v \quad (1)$$

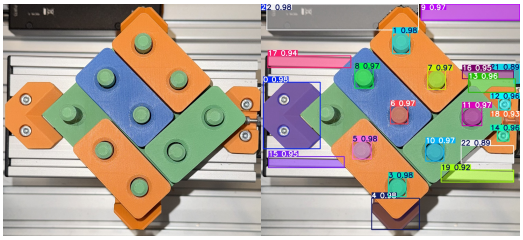
where IoU is the Intersection over Union, $\rho^2(b, b^{gt})$ is the squared Euclidian distance between center points, v is the aspect ratio of the image, and α is the weighting factor emphazing aspect ratio for low IoU.

Distribution Focal Loss (DFL) function enhances regression accuracy, by treating box boundaries as probability distributions:

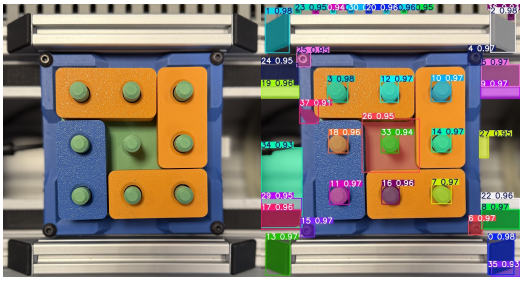
$$L_{DFL} = -\sum_{i \in \{l,r,t,b\}} (y_i log(p_i) + (1 - y_i) log(1 - p_i)) \quad (2)$$

where y_i is the ground-truth offset for the box boundary (left, right, top, bottom) relative to grid cell, p_i is the predicted probability distribution over discrete boundary offsets.

Figure 3 illustrates raw images captured above the assembly carrier in Figure 3a and above the conveyor belt in Figure 3c, with processed images (Figure 3b, 3d) showing tracked components during assembly using object detection models. The dataset comprises 100 images collected at 1920x1080 resolution from industrial cameras, cropped to focus on relevant regions, and annotated for five object classes. The small dataset size reflects the proof-of-concept nature of this study, leveraging pre-trained YOLO models to generalize from the COCO dataset.



(a) Raw image of reconfigured product (b) Detected pieces



(c) Raw image of assembled product (d) Checked product

Fig. 3: Applications during the assembly process

The general output size formula for the convolution layer to extract features for the object detection in all YOLO model, based on [18] article, is:

$$Dim_o = \frac{Dim_i + 2P - K}{S} + 1 \tag{3}$$

$$Channels_o = N_{filters} \tag{4}$$

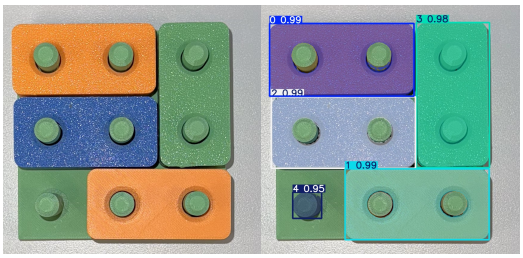
where Dim is used for either width or height of the image, P is the padding, K is size of the kernel, S is the stride, and $N_{filters}$ is the number of filters applied on the input image to determine the output feature channels $Channels_o$. YOLOv8 convolutional layers are using Sigmoid Linear Unit (SiLU) activation function, shown with the following formula:

$$SiLU(x) = \frac{x}{1 + e^{-x}} \tag{5}$$

This activation function is used to introduce non-linearity in the model, increasing the ability to learn complex features for object detection.

A specialised object detection model ensures accurate component placement, while an image classification model verifies that the assembled product matches the customer's specifications. Errors, such as a piece detected as "green" in the warehouse but "blue" in the final product, are flagged for correction. Faulty pieces damaged by pneumatic actuators are also detected, enabling rapid error logging and correction to minimize downtime.

In Figure 4, the final product is shown before entering the storage station. The detection algorithm serves as the basis for a final product inspection. The pre-trained YOLOv8 model, using default hyperparameters and weights trained on the COCO dataset, achieves reliable detection in this industrial context. Figure 4b is the processed image of the raw image, Figure 4a, where all parts are checked for damages or incorrect configuration.



(a) Raw image of the final product (b) Detected product

Fig. 4: Identification of final product configuration

One of the main features that such a model must check is the noise tolerance. This feature is mandatory because even one false detection can compromise the production process. This implies that the model must not confuse shadows with anomalies and must not be influenced by the light conditions in the room, as most of the traditional algorithms lose accuracy exponentially due to lighting conditions. The robustness of the model under different conditions is one of the key metrics that must be taken into account when developing vision-based applications. This study demonstrates the practical application of pre-trained YOLO models in a five-station manufacturing line, achieving robust defect detection without requiring extensive dataset collection or model retraining, unlike prior studies that focus on custom-trained models for specific industrial tasks.

IV. Results

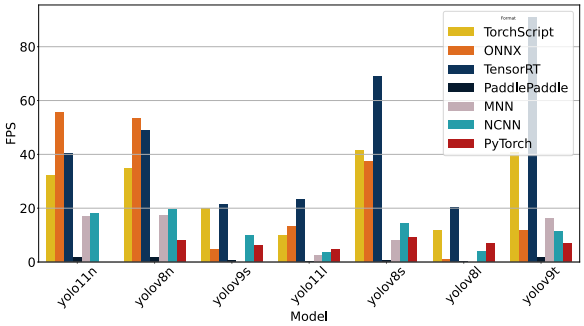
The development platform chosen to perform the benchmark test as core reference is the embedded system edge AI computer with NVIDIA Jetson Orin NX 16GB as it demonstrates substantial performance in ML applications. The GPU board delivers up to 100 trillions of operations per second (TOPS), making it well-suitable for high-demand workloads such as real-time object detection, tracking and instance segmentation. When paired with Ultralytics YOLO11, the Orin NX achieves considerable results in metrics such as inference time, having

good compatibility with ML frameworks. It is a well-designed and popular development embedded system even for industrial applications, as seen in its AGX variants. Figure 5 presents the results of the test of different open-source models and frameworks available with Ultralytics packages. Across all models, TensorRT offers the best speed in image processing and in terms of compatibility since TensorRT is an advanced software development kit (SDK) developed by NVIDIA and designed for high-speed deep learning inference. It is well-suited for real-time applications like object detection. This toolkit optimizes deep learning models for NVIDIA GPUs and results in faster and more efficient operations. This statement is proved as seen in Figure 5a. The Frames per Second (FPS) rate has the highest average score, both on light-weighted models (yolo11n, yolov8n and yolov9n) and high-weighted models (yolo11l, yolo8l, and yolo9l). In terms of precision, Figure 5b shows that different formats do not necessarily introduce computational errors, as the scores are almost the same. The only difference is when referencing larger models like yolo11l and yolov8l, which have 10 times more parameters in the structure than the lighter analog models yolo11n and yolov8n. This conclusion is expected when taking into account the features of TensorRT models as provided by the developer:

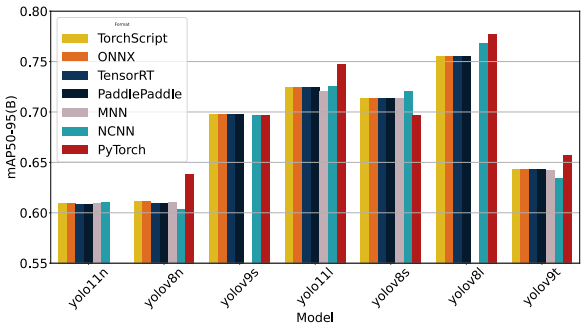
1) The first major feature of those models is the layer fusion, which is a process of optimization in which multiple layers of the neural network are combined into single operations. This reduces computational workload and reduces inference speed.
2) Another feature of TensorRT is the precision calibration that allows models to be fine-tuned for specific accuracy requirements.
3) This fine-tuning process is further improved by the automatic tuning mechanism that selects the most optimized GPU kernel for each layer of the model.
4) The last feature is efficient memory management that optimizes memory allocation and reduces memory overhead during inference.

Table 1 summarises the quantitative benchmarking results for the Jetson Orin use case.

Another test set is shown in Figure 6 to evaluate if testing in the cloud is more efficient for the final application. The same configurations have been made to ensure that the platform can work with multiple models to extract some results and get some conclusions concerning cloud computing and cloud model training, exposing the proprietary data to another server. Looking at Figure 6a, it is obvious that the TensorRT format offers the best performance, both on FPS and $mAP50-95$ in Figure 6b. The same can be said about the TorchScript format, which gives almost half the speed and the same precision score. As the number of parameters increases, the FPS scores fall by almost half of the initial values, indicating a higher toll on the calculation unit. However, accuracy performances increase by more than 10% for each model and format, which means that the platform is not too sensitive to increasing parameters or the model architecture. This solves some prob-



(a) FPS of different models and formats on Jetson Orin hardware



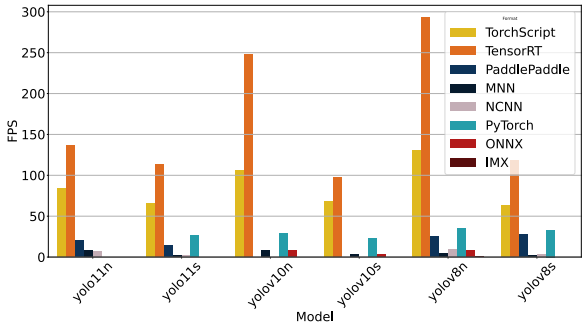
(b) Mean Average Precision of different models and formats on Jetson Orin hardware

Fig. 5: Performances on Jetson Orin hardware

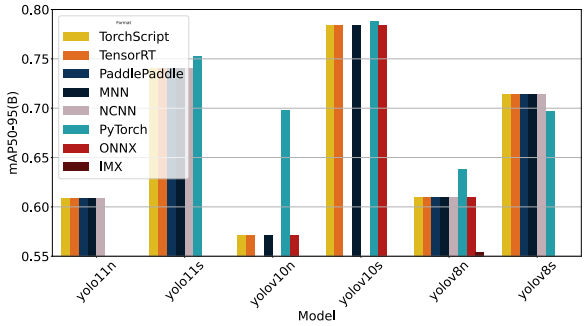
Model	Format	Size (MB)	mAP50-95(B)	FPS
yolo11l.pt	TensorRT	101.7	0.725	23.14
yolo11l.pt	ONNX	97	0.725	13.1
yolo11l.pt	TorchScript	97.6	0.725	9.99
yolo11n.pt	ONNX	10.2	0.61	55.48
yolo11n.pt	TensorRT	15.2	0.6082	40.42
yolo11n.pt	TorchScript	10.5	0.61	32.25
yolo11n.pt	NCNN	10.2	0.6106	18
yolo11n.pt	MNN	10.1	0.6099	16.96
yolo11n.pt	PyTorch	5.4	0.6176	7.59
yolov8l.pt	TensorRT	170.6	0.7554	20.2
yolov8l.pt	TorchScript	167.2	0.7554	11.77
yolov8n.pt	ONNX	12.2	0.6117	53.43
yolov8n.pt	TensorRT	16.1	0.6092	48.77
yolov8n.pt	TorchScript	12.4	0.6117	34.85
yolov8n.pt	NCNN	12.2	0.6034	19.41
yolov8n.pt	MNN	12.2	0.6104	17.23
yolov8n.pt	PyTorch	6.2	0.6381	7.84
yolov8s.pt	TensorRT	46.1	0.7136	68.93
yolov8s.pt	TorchScript	43	0.7136	41.45
yolov8s.pt	ONNX	42.8	0.7136	37.48
yolov8s.pt	NCNN	42.7	0.7204	14.16
yolov8s.pt	PyTorch	21.5	0.6967	9.17
yolov8s.pt	MNN	42.7	0.7141	7.84
yolov9s.pt	TensorRT	33.7	0.6974	21.4
yolov9s.pt	TorchScript	28.4	0.6981	19.83
yolov9s.pt	NCNN	27.7	0.6973	9.72
yolov9t.pt	TensorRT	12.6	0.6428	90.96
yolov9t.pt	TorchScript	9	0.6428	40.63
yolov9t.pt	MNN	8.2	0.6423	16
yolov9t.pt	ONNX	8.3	0.6428	11.86
yolov9t.pt	NCNN	8.2	0.6348	11.44

TABLE I: NVIDIA Jetson Orin NX 16GB benchmarks

lems with compatibility matrices, making it easier to choose a model and a format to develop the application. Of course, the results can be exported from the cloud and processed on local machines to make further improvements, thus reducing the exposure of data and vulnerabilities to third parties. Disadvatages in using third party platforms are also underscored by the constraints that are imposed by the developers. An example discovered during the tests was the safe resource utilization mechanism. These platforms typically impose limits on resource usage, such as execution time restrictions, time-outs, and safeguards that prevent excessive resource consumption beyond allocated quotas. This mechanism did not allow one to benchmark bigger models like yolov8l, yolov10l or yolo11l as it stopped the code from running and no results could be recorded.



(a) FPS of different models and formats on Colab platform



(b) Mean Average Precision of different models and formats on Colab platform

Fig. 6: Performances on Google Colab cloud

Table 2 summarises the quantitative benchmarking results for the Google Colab use case.

Figure 7a shows the results of running multiple YOLO object detection models of different sizes (n,s,l) and architectures (v11, v9, v8) on the local machine. Based on those results, models yolo11n and yolov8n are the fastest, with scores above 60 Frames per Second (FPS), corresponding with inference times below 20 milliseconds per image, making them the best option for real-time applications on the local machine. However, studying Figure 7b, the scores of these 2 models are the lowest, with mAP50-95 below 65% for each format, compared to the other models such as yolo11s and yolov8s which score

Model	Format	Size (MB)	mAP50-95(B)	FPS
yolo11n.pt	TensorRT	13.3	0.6082	135.97
yolo11n.pt	TorchScript	10.5	0.6082	83.35
yolo11n.pt	PyTorch	5.4	0.61	28.28
yolo11n.pt	PaddlePaddle	20.4	0.6082	19.96
yolo11n.pt	MNN	10.1	0.6082	7.79
yolo11n.pt	NCNN	10.2	0.6082	7.3
yolo11s.pt	TensorRT	40.4	0.74	112.72
yolo11s.pt	TorchScript	36.5	0.74	64.98
yolo11s.pt	PyTorch	18.4	0.7526	26.36
yolo11s.pt	PaddlePaddle	72.5	0.74	14.28
yolov10n.pt	TensorRT	12.9	0.5715	248.28
yolov10n.pt	TorchScript	11.1	0.5715	106.08
yolov10n.pt	PyTorch	5.6	0.6981	28.63
yolov10n.pt	ONNX	9	0.5715	8.29
yolov10n.pt	MNN	8.9	0.5715	7.99
yolov10s.pt	TensorRT	34.7	0.784	97.54
yolov10s.pt	TorchScript	31.5	0.784	68.44
yolov10s.pt	PyTorch	15.9	0.7881	22.92
yolov8n.pt	TensorRT	17.7	0.6092	293.28
yolov8n.pt	TorchScript	12.4	0.6092	130.86
yolov8n.pt	PyTorch	6.2	0.6381	34.75
yolov8n.pt	PaddlePaddle	24.4	0.6092	25.62
yolov8n.pt	NCNN	12.2	0.6092	8.97
yolov8n.pt	ONNX	12.2	0.6092	7.96
yolov8s.pt	TensorRT	54	0.7136	118.56
yolov8s.pt	TorchScript	43	0.7136	62.58
yolov8s.pt	PyTorch	21.5	0.6967	33.15
yolov8s.pt	PaddlePaddle	85.5	0.7136	28.02

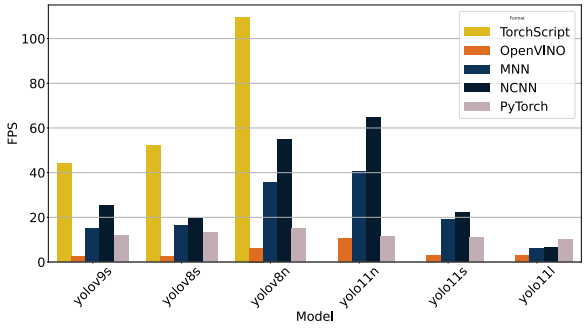
TABLE II: Google Colab Tesla T4 GPU benchmarks

above 70% with almost each format. Those models are the bigger versions of the first two, having more than 3 times more parameters in the architecture. This difference in size implies computational costs; as seen in Figure 7a, these two models achieve under 20 FPS on each format. Moving even further, models yolo11l and yolo8l, the models with almost 3 times and, respectively, 4 times more parameters than the previous models, show limited improvement in any score. The mAP50-95 scores of these two models are comparable with their smaller versions, showing only disadvantages with the FPS scores, which are half the scores of the latest. It is obvious that, by increasing the size of the model, the complexity increases, and the time needed to compute the final prediction (the result) also increases, making the model less likely to perform well on reduced hardware resources. One key point of those models is the compatibility with the packages installed on the machine, some models require more disk space, some require older versions of software packages. This continuous package dependency is time consuming when developing a stable application, and it easily becomes insignificant as the software progresses faster and new compatibility matrices arise each time, some of them solving one issue and rising another.

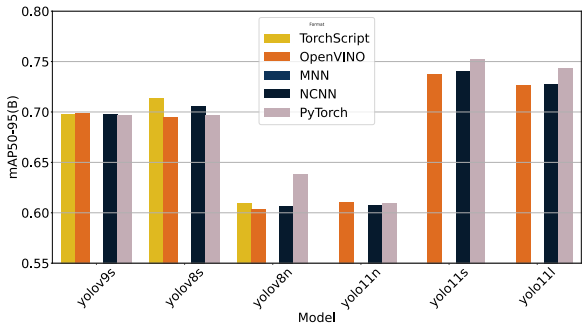
Table 3 summarises the quantitative benchmarking results for the Apple M3 use case.

V. Conclusion

The development of Industry 5.0 applications based on artificial intelligence algorithms to help increase productivity and quality in advanced manufacturing scenarios has been adopted at a growing rate. The use of dedicated hardware and compatible software libraries to solve common applications,



(a) FPS of different models and formats on Apple hardware



(b) Mean Average Precision of different models and formats on Apple hardware

Fig. 7: Performances on Apple machine

Model	Format	Size (MB)	mAP50-95(B)	FPS
yolov9s.pt	PyTorch	14.7	0.6973	9.68
yolov9s.pt	TorchScript	28.5	0.6981	48.74
yolov9s.pt	OpenVINO	28.2	0.6991	2.49
yolov9s.pt	NCNN	27.7	0.698	25.02
yolov8s.pt	PyTorch	21.5	0.6967	13.01
yolov8s.pt	TorchScript	43	0.7136	52.27
yolov8s.pt	OpenVINO	42.9	0.6951	2.6
yolov8s.pt	NCNN	42.7	0.7059	19.49
yolov8n.pt	PyTorch	6.2	0.6381	14.83
yolov8n.pt	TorchScript	12.4	0.6092	109.58
yolov8n.pt	OpenVINO	12.3	0.6034	6.28
yolov8n.pt	NCNN	12.2	0.6062	54.75
yolov9s.pt	PyTorch	14.7	0.6973	11.68
yolov9s.pt	TorchScript	28.5	0.6981	39.44
yolov9s.pt	OpenVINO	28.2	0.6991	2.68
yolov9s.pt	NCNN	27.7	0.698	25.19
yolo11n.pt	PyTorch	5.4	0.61	11.54
yolo11n.pt	OpenVINO	10.4	0.6107	10.73
yolo11n.pt	NCNN	10.2	0.6078	64.66
yolo11s.pt	PyTorch	18.4	0.7526	11
yolo11s.pt	OpenVINO	36.4	0.738	2.78
yolo11s.pt	NCNN	36.2	0.7407	22.3
yolo11l.pt	PyTorch	49	0.743	9.88
yolo11l.pt	OpenVINO	97.3	0.7271	2.93
yolo11l.pt	NCNN	96.9	0.7274	6.56

TABLE III: Apple Macbook M3 CPU/GPU benchmarks

such as quality inspection and process monitoring, is a move known and appreciated by integrators in industrial fields. This paper introduced an embedded learning application designed for vision-based applications within a five-station connected laboratory flexible manufacturing line. The quantitative results presented here provide a comparative evaluation of multiple generations of the YOLO real-time object detection model family. Due to the small dataset size, this study serves as a proof-of-concept, demonstrating the feasibility of applying pre-trained YOLO models to industrial defect detection. In addition, the paper discusses critical implementation considerations and integration aspects with industrial automation technologies, offering insights into the practical deployment of such systems in manufacturing environments. These findings contribute to advancing the field of automated quality inspections. Future work will involve collecting a larger dataset to validate performance across diverse conditions.

REFERENCES

[1] E. Commission, D.-G. for Research, Innovation, M. Breque, L. De Nul, and A. Petridis, *Industry 5.0 – Towards a sustainable, human-centric and resilient European industry*. European Union, 2021.

[2] S. Rosioru, G. Stamatescu, I. Stamatescu, I. Fagarasan, and D. Popescu, "Deep learning based parts classification in a cognitive robotic cell system," in *2022 26th International Conference on System Theory, Control and Computing (ICSTCC)*, 2022, pp. 403–408.

[3] X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang, "Industry 4.0 and industry 5.0—inception, conception and perception," *Journal of Manufacturing Systems*, vol. 61, pp. 530–535, 2021.

[4] J. Leng, W. Sha, B. Wang, P. Zheng, C. Zhuang, Q. Liu, T. Wuest, D. Mourtzis, and L. Wang, "Industry 5.0: Prospect and retrospect," *Journal of Manufacturing Systems*, vol. 65, pp. 279–295, 2022.

[5] D. Weimer, B. Scholz-Reiter, and M. Shpitalni, "Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection," *CIRP Annals*, vol. 65, no. 1, pp. 417–420, 2016.

[6] J. Božič, D. Tabernik, and D. Skočaj, "Mixed supervision for surface-defect detection: From weakly to fully supervised learning," *Computers in Industry*, vol. 129, p. 103459, Aug. 2021.

[7] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023.

[8] C.-Y. Wang and H.-Y. M. Liao, "Yolov9: Learning what you want to learn using programmable gradient information," *arXiv preprint arXiv:2402.13616*, 2024.

[9] A. K. et al., "Segment anything," 2023.

[10] X. Zhao, W. Ding, Y. An, Y. Du, T. Yu, M. Li, M. Tang, and J. Wang, "Fast segment anything," 2023.

[11] T. Cheng, L. Song, Y. Ge, W. Liu, X. Wang, and Y. Shan, "Yolo-world: Real-time open-vocabulary object detection," *arXiv preprint arXiv:2401.17270*, 2024.

[12] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit," *Electronics*, vol. 10, no. 3, 2021.

[13] M.-D. Pavel and G. Stamatescu, "Flexible manufacturing system for enhanced industry 4.0 and industry 5.0 applications," in *Intl Conf on Distributed Computing in Smart Systems and Internet of Things*, 2024.

[14] M.-D. Pavel, S. Roșioru, N. Arghira, and G. Stamatescu, "Control of open mobile robotic platform using deep reinforcement learning," in *SOHOMA Workshop*. Springer, 2023, pp. 368–379.

[15] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023. [Online]. Available: https://www.mdpi.com/2504-4990/5/4/83

[16] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[17] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-iou loss: Faster and better learning for bounding box regression," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 12 993–13 000, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6999

[18] S. Sakib, N. Ahmed, A. J. Kabir, and H. Ahmed, "An overview of convolutional neural network: Its architecture and applications," *Preprints*, February 2019. [Online]. Available: https://doi.org/10.20944/preprints201811.0546.v4