Towards Real-Time LPD on Resource-constrained CPUs: An Empirical Evaluation of Pruning and Quantization for YOLOv8

Safa AMEUR*, Amir ISMAIL*†, Xavier CLADY ‡, Anis SAHBANI †‡, Najoua ESSOUKRI BEN AMARA *
* Université de Sousse, Ecole Nationale d'Ingénieurs de Sousse,
LATIS-Laboratory of Advanced Technology and Intelligent Systems, 4023, Sousse, Tunisie;
†Enova Robotics S.A., Sousse 4000, Tunisia;
‡Sorbonne Université, 75005 Paris, France;
Email: *safa.ameur@eniso.u-sousse.tn

***Abstract*—The growing incorporation of deep learning into autonomous robotic systems has highlighted the importance of efficient inference on resource-constrained edge platforms. Moreover, License Plate Detection (LPD) is a key component of intelligent transportation systems, requiring high accuracy and real-time performance for practical deployment. However, deep learning detectors often remain computationally demanding for edge devices with limited resources. This paper proposes an edge-oriented optimization framework for YOLOv8-based LPD, combining structured pruning, INT8 quantization. Extensive experiments evaluate the trade-off between accuracy, computational complexity, and inference latency. Results show that combining pruning and quantization significantly reduce model size and computational cost while preserving detection performance. The optimized model achieves real-time inference exceeding 100 FPS on an Intel CPU, demonstrating its suitability for resource-constrained edge environments. These findings confirm that hybrid compression strategies enable efficient deployment of deep learning–based LPD systems in real-world applications.**

***Index Terms*—edge AI, inference optimization, model compression, license plate detection, structured pruning, quantization, knowledge distillation, Real-Time Robotics**

I. Introduction

Recent advances in Artificial Intelligence (AI) have made robots much more autonomous, allowing them to work in many different fields, such as healthcare [12], logistics, agriculture [11], [13], surveillance [3], and service robotics. Edge intelligence has become a key technology for real-time robotic autonomy in this larger AI ecosystem. It lets AI models run on embedded robotic hardware platforms without needing cloud infrastructure. Therefore, it has intensified the need for efficient inference on resource-constrained edge platforms [7].

Moreover, deep learning–based object detection models, particularly those derived from the YOLO family, have demonstrated remarkable performance in visual detection tasks, including License Plate Detection (LPD). However, their computational requirements often remain a challenge for edge deployment scenarios, where limited processing power, memory capacity, and energy consumption constraints must be considered [3]. Beyond traditional tolling scenarios, this work targets embedded deployment in mobile robotic platforms (e.g., surveillance robots or patrol vehicles), where computational constraints, energy efficiency, and real-time processing are critical. This context motivates the need for a systematic optimization pipeline tailored to CPU-based edge systems in order to be implemented on the mobile security robot PGuard[1]. In this study, we propose an edge-aware optimization pipeline for YOLOv8-based LPD, combining structured model compression and deployment-oriented optimization techniques. The approach leverages the OpenMMLab ecosystem[2], including MMYOLO for model training, MMRazor for pruning and quantization, and MMDeploy for efficient model export and hardware-specific acceleration. Through this unified pipeline, the model is progressively optimized from the training stage to deployment, enabling efficient inference on resource-constrained edge hardware. Comprehensive experiments demonstrate that the proposed optimization strategy significantly improves inference efficiency while preserving high detection accuracy. By combining structured pruning, quantization, and static graph deployment, the optimized model achieves real-time performance on CPU-based edge platforms while maintaining strong robustness in challenging LPD scenarios. In summary, the main contributions of this work are as follows:

- We review state-of-the-art techniques to optimize AI models for inference on resource-constrained hardware, with emphasis on model compression, architecture design, and system-level acceleration relevant to robotic deployments.
- We propose a complete optimization workflow for YOLOv8-based LPD that integrates training, structured pruning, quantization, and hardware-aware deployment.
- We analyze the trade-offs between compression ratio, inference latency, and detection accuracy on two benchmark datasets.

[1]https://enovarobotics.eu/pguard/

[2]OpenMMLab Contributors, "OpenMMLab: An open source algorithm platform for computer vision," 2020. [Online]. Available: https://github.com/open-mmlab

Through these contributions, our work aims to evaluate key model-level optimization strategies that enable efficient edge deployment without significant accuracy degradation. The remainder of the paper is organized as follows: Section II details the background and motivation of this work. Section III presents the experimental setup, results, and analysis. Finally, Section IV concludes the paper and outlines directions for future work.

II. Background and Motivation

Several strategies for inference optimization have been proposed in the literature to deal with the computational constraints of embedded robots. Most strategies aim to maintain a certain level of accuracy on real-time perception tasks such as LPD, while reducing model complexity, memory requirements, and inference latency. Optimization strategies can be grouped into a certain number of levels or classes, including system-level optimization, architecture-level optimization, and model-level optimization [1]. Model-level optimization strategies aim to reduce the size of a pretrained network by using essentially knowledge distillation, quantization, and pruning. Though system-level methods utilize hardware-aware frameworks, optimized inference engines, and accelerators, architecture-level methods design neural networks that are naturally efficient and well-suited for embedded [2]. An overview of the major categories and their relationships, as they apply to edge AI deployment, is presented in Figure 1, which provides a high-level taxonomy of inference optimization techniques found in the state of the art.

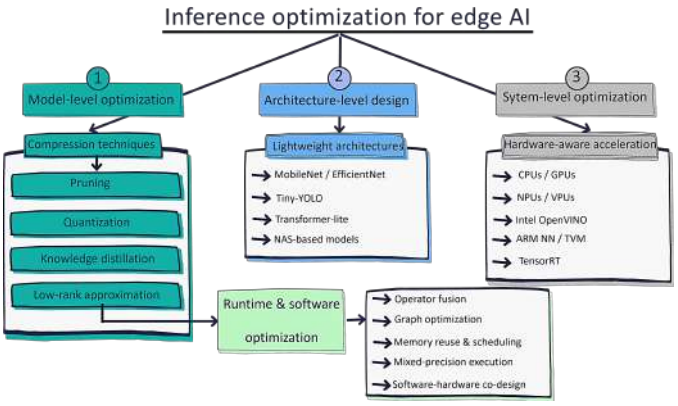

Fig. 1. Taxonomy of inference optimization methods for deploying AI models on edge robotic platforms. Optimization strategies span model-level compression, architecture-level design, and system-level acceleration, jointly enabling real-time and energy-efficient inference under hardware constraints.

A. Model-Level Optimization

Model compression reduces memory and computational requirements while preserving most of the model accuracy. It is therefore essential for edge deployment and typically relies on four main strategies: pruning, quantization, knowledge distillation, and low-rank approximation [4].

*1) **Pruning**:* Pruning is a model compression technique that improves inference efficiency by removing the less important parameters from the neural network. Pruning removes redundant weights or neurons in a neural network that contribute less to predictions. Removing these redundant parameters makes neural networks computationally less expensive. There are three types of neural network pruning techniques, unstructured, structured, and dynamic or semi-organized pruning [2]. Dynamic pruning adjusts the level of pruning during training or inference to accommodate different resource constraints.
In unstructured pruning, individual weights with small magnitudes are set to zero, typically guided by a threshold τ:

$$\hat{W}_{i,j} = \begin{cases} 0, & \text{if } |W_{i,j}| < \tau \\ W_{i,j}, & \text{otherwise} \end{cases} \tag{1}$$

Structured pruning removes entire neurons, channels, or filters based on an importance metric such as the L1-norm $\|x\|_1$:

$$\|f_k\|_1 = \sum_{i,j} |f_k(i,j)|, \quad f_k \text{ is pruned if } \|f_k\|_1 < \tau \tag{2}$$

This approach preserves hardware-friendly tensor shapes, enabling efficient parallel execution on embedded devices like mobile GPUs. The lottery ticket hypothesis also suggests that, in the absence of further training, sub-models discovered by pruning can have accuracy comparable to the original model. While memory can be reduced by up to 82%, according to current research, for models pruned aggressively, there are compromises to be made in terms of accuracy [4]. Thus, structured pruning allows for parallel hardware execution, making it particularly suitable for embedded and edge platforms, and thus it is an important technique for optimizing deep learning models, including real-time LPD for resource-constrained systems.

*2) **Quantization**:* The numerical accuracy of weights and activations is impacted by quantization, which is a model-level optimization technique that typically involves converting 32-bit (FP32) data types into 16-bit (FP16), 8-bit (INT8), and even 4-bit (INT4) data types. This has been seen to provide a 3× to 8× speedup on edge devices that support low-precision arithmetic, such as mobile devices, TPUs, and NPUs [5], [8], [9].
Post-Training Quantization (PTQ) is a quantization technique that can be employed to reduce the accuracy of pre-trained models. This technique is easy to apply and does not require access to the original training data. Instead of employing PTQ, Quantization-Aware Training (QAT) is another technique that attempts to simulate quantization impacts throughout training, which has been seen to be less accurate but still shows small accuracy loss [2]. In order to achieve performance and efficiency, it is possible to apply quantization to all layers apart from fully linked layers. The quantization process converts floating-point values b to discrete integer values d, using a scale factor f and zero-point p as follows:

$$d = \text{round}\left(\frac{b}{f}\right) + p \tag{3}$$

Here, f maps the real value range to a quantized range (e.g., 0–255), and p anchors the representation of zero. To prevent out-of-bound values, clamping is applied to fit within the data type's range. This reduces the amount of memory and computation needed since it reduces the number of bits required to hold each parameter. This is therefore an important technique for implementing deep learning on robotic systems with limited resources, such as LPD.

3) ***Knowledge distillation****:* It is a model compression technique that transfers knowledge from a large, complex instructor model to a smaller, efficient student model. The key is to retain high predictive performance while significantly reducing the size and computation cost of the model, making it suitable for deployment on resource-constrained devices [14]. The student model is trained to mimic the probability distribution of the instructor model's outputs, known as soft targets, which provide more information about relationships between classes and model confidence, instead of learning from ground truth. The popularity of knowledge distillation in the development of efficient deep learning models that can achieve almost teacher-student accuracy under severe compression constraints is undeniable [6].

4) ***Low-rank approximation****:* It decreases the complexity of the computations in the deep neural network by decomposing the large weight matrices or the convolution kernels, which are represented as the product of the low-rank matrices. In this technique, the weight matrices are not directly computed; instead, the weight matrices are represented as the product of the low-rank matrices, given by $W \approx UV$, where the matrices U and V are low-rank matrices [2]. Using this technique, the complexity is reduced, and the number of computations is minimized, allowing the deployment of the deep learning models on the edge devices.

B. Architecture-Level design

Architecture-level approaches focus on the development of lightweight neural networks, particularly for resource-constrained systems, rather than applying compression methods for existing resource-intensive networks. Existing edge-oriented architectures, including MobileNet, EfficientNet, MobileViT, and EdgeFormer, have utilized lightweight attention mechanisms with low computational cost, inverted residual connections, and depth-wise separable convolution as powerful building blocks. Real-time inference on robotics platforms using the ARM and NVIDIA Jetson platforms is now possible due to the introduction of the above architecture concepts, which have reduced the network parameters and FLOPs.

The other significant technique for architecture-level optimization is called Neural Architecture Search (NAS), which helps discover efficient neural network architectures under certain hardware constraints. Here, the search for efficient architectures is performed over a wide design space of candidate architectures using search techniques such as reinforcement learning, evolutionary search, or gradient-based optimization techniques. The main aim of NAS is to search for architectures with good trade-offs between accuracy, latency, memory, and energy consumption.

The techniques for NAS optimization can be broadly categorized into two categories, namely, proxy-based NAS and proxyless NAS. In proxy-based NAS, the candidate architectures are evaluated using surrogate models or reduced training settings, reducing the overall search time, whereas proxyless NAS directly evaluates the architectures on the hardware platform. Though proxy-based NAS reduces the overall search time, proxyless NAS is more effective in learning the actual hardware characteristics of embedded devices. Thus, NAS optimization techniques are effective for hardware-aware lightweight model optimization for robotic perception inference at the edge [7].

C. System-Level Optimization

This relates to optimizing at the system level in which hardware and software frameworks are used to enhance inference performance on edge devices. Specialized accelerators and associated runtime engines take advantage of heterogeneous compute units (CPU, GPU, NPU) to provide modern robotic platforms with appropriate interfaces [10].

1) ***Hardware Acceleration****:* Various edge platforms offer specialized hardware support and optimize libraries:

- Intel OpenVINO: Intel's optimization library for models targeting tightly-coupled CPUs and VPUs & Pruning and concreting aware kernels supported.
- ARM NN, TVM, TensorFlow Lite: Optimizes models for ARM CPUs and NPUs commonly residing in robotics SoCs.
- TensorRT (NVIDIA Jetson): To allow these highly-optimized, low-latency GPU inferences.

These frameworks employ various techniques such as the fusion of operations, memory planning, and graph rewriting to lower the execution overhead while maximizing the system throughput.

2) ***Deployment Engines and Graph Optimization****:* In the optimized inference engines, graph-level optimization is used to optimize the execution efficiency:

- Operator fusion: This is the combination of two or more operations in the graph that are frequently used together, such as Conv + Batch Normalization + Relu activation. This reduces the execution time.
- Memory reuse: This minimizes memory bandwidth usage, which is particularly important for embedded ARM devices and mobile GPUs that have limited memory.

The use of hardware-based acceleration in combination with system-level techniques allows for real-time execution with low latency in resource-constrained robotic systems.

D. Model optimization for Edge-Based LPD

Edge deployment of LPD models, such as mobile robots, smart traffic cameras, or parking systems, demands models that operate under severe constraints in terms of memory, latency, and power consumption [2]. Model-level optimization techniques, such as pruning, quantization, and compression, can significantly reduce the computation and memory demands

while ensuring the accuracy of the detection, even under severe environmental conditions, including motion blur, low illumination, and partial occlusion. Such optimizations are essential for real-time LPD using lightweight CNN-based detectors, including YOLOv8, running on ARM-based SoCs or Intel-based embedded systems. Moreover, LPD is an extremely challenging problem due to the object's small size, high intra-class variability (across various license plate formats), noise, and the need for real-time processing. Thus, compression is not just an efficiency improvement but a fundamental enabler for practical large-scale edge-based LPD systems.

In order to overcome these challenges, we suggest an edge-aware optimization pipeline for YOLOv8, as shown in Figure 2, which applies structured pruning, fine-tuning, and quantization in that order. To achieve high initial accuracy, the pipeline starts with baseline training of YOLOv8 on the benchmark datasets. In order to reduce model size and computation while preserving hardware-friendly tensor shapes, structured pruning eliminates unnecessary neurons and filters. Any accuracy loss is then recovered through fine-tuning. Quantization, either through QAT or PTQ, reduces memory consumption and speeds up inference on edge CPUs or embedded GPUs with little effect on performance by converting the pruned model to lower-precision integers (such as INT8). Figure 2 illustrates the integration of the proposed LPD optimization pipeline into YOLOv8, showing the transition from dynamic development to static deployment to ensure real-time performance on resource-constrained CPUs.

III. Experimental Results

A. Datasets

We conduct our experiments on two benchmark datasets: **LPD dataset**[3] is designed for license plate detection tasks. It consists of 5561 images. It was split into 4865 training images, 464 for validation, and 232 for test. It contains annotated images of vehicles with bounding boxes labeling license plate regions, enabling supervised training of object detection models. The dataset supports common formats compatible with modern detection frameworks (e.g., YOLO, COCO, JSON), facilitating direct integration into deep learning pipelines.

PGTLP dataset[4] is a Tunisian license plate dataset collected by the mobile security robot Pearl Guard navigating real parking lots and high-risk urban areas. Images were captured under two motion scenarios, stationary robot with moving vehicle, and both in motion simultaneously across three resolutions: 1920×1080, 800×600, and 640×480 pixels. The dataset covers diverse Tunisian license plate templates and includes multi-plate frames with up to three plate instances per image. For our experiments, we use an extended version of this dataset comprising 4,912 images (3,912 train / 1,000 test). To our knowledge, PGTLP remains the only publicly available dataset dedicated to Tunisian license plate detection.

[3]LPD open Source Dataset, Roboflow Universe, 2023. Available: https://universe.roboflow.com/mmm-u89pc/lpd-qvs3z

[4]PGUARD ALPR Datasets, Enova Robotics, 2021. Available: https://emirismail.github.io/

B. Edge implementation Details

All experiments use PyTorch with YOLOv8 (MMYOLO) as the baseline detector, trained on an NVIDIA RTX 3050 6GB / Intel Core i7-12650H workstation. Models are trained for 100 epochs on 640×640 inputs, with batch size 16, AdamW optimizer (lr=0.001, weight decay=0.0005), and evaluated using mAP@0.5:0.95.

The optimization pipeline comprises two stages as illustrated in Figure2. In the development stage, the baseline YOLOv8 is compressed via structured pruning (MMRazor, L1-norm criterion), followed by fine-tuning to recover accuracy, then INT8 quantization via both PTQ and QAT. In the deployment stage, the optimized model is exported through MMDeploy: the PyTorch checkpoint is converted to ONNX, then compiled into OpenVINO IR format for Intel hardware acceleration, enabling low-latency CPU inference suited to real-time edge LPD.

C. Comparative Performance Analysis

To evaluate the impact of the proposed optimization strategy, we conducted a comparative analysis between the baseline YOLOv8 model and its optimized variants after pruning, fine-tuning, quantization and compression. The pruning configuration was designed to significantly reduce computational complexity while preserving detection accuracy. Specifically, the pruning ratio adopted during the structured pruning procedure was set initially at 25%, resulting in an estimate of a reduction of 35% in the total FLOPs. Furthermore, an increased pruning ratio of 50% resulted in a reduction in the computing requirement of 50% in the FLOPs while ensuring the performance of the detection task through fine-tuning. This demonstrates the effectiveness of the structured compression methods in minimizing the computational weights of the model without compromising the predictive capabilities.

Figure 3 illustrates qualitative detection results on the LPD dataset, obtained from the optimized inference pipeline. The figure compares baseline ground-truth annotations with the predictions produced by the final optimized ONNX inference engine. The first row presents the annotated reference images, while the second row displays the detection outputs generated by the compressed model. This comparison demonstrates that the optimized YOLOv8 detector preserves strong visual detection performance despite the applied compression techniques. Furthermore, the bounding box predictions generated by the optimized model show strong alignment with the ground-truth annotations, indicating that the structural pruning performed through the MMRazor framework preserves the spatial regression accuracy of the YOLO detection head. Finally, the optimized deployment pipeline significantly improves inference efficiency.

Tables I and II summarize the trade-off between accuracy and computational efficiency obtained through the different optimization strategies. While structured pruning reduces the number of parameters and FLOPs, it introduces a moderate accuracy degradation proportional to the pruning ratio. Figure4 presents the Pareto frontier illustrating the trade-off between

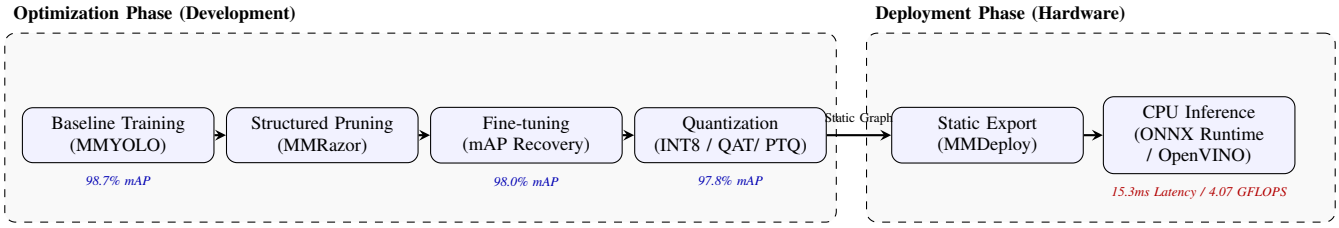

Fig. 2. Integration of the proposed LPD optimization pipeline into YOLOv8 model, including pruning, fine-tuning, and quantization. The transition from dynamic development to static deployment ensures real-time performance on resource-constrained CPUs.

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT OPTIMIZATION METHODS APPLIED TO THE YOLOV8 LPD ON TWO BENCHMARK DATASETS.

	LPD dataset			PGTLP dataset		
Method	**mAP@50**	**mAP@75**	**mAP@50:95**	**mAP@50**	**mAP@75**	**mAP@50:95**
Baseline Model (PyTorch)	98.7%	98.1%	81.3%	97.8%	97.2%	79.6%
ONNX Conversion	98.9%	98.9%	81.4%	97.9%	97.4%	79.7%
OpenVINO IR (FP32)	98.9%	98.8%	81.4%	97.9%	97.3%	79.7%
PTQ INT8	98.4%	97.8%	81.0%	97.1%	96.5%	79.0%
QAT INT8	98.6%	98.2%	81.2%	97.4%	96.9%	79.3%
Pruning 25%	98.0%	97.5%	80.8%	96.8%	96.1%	78.5%
Pruning 50%	95.8%	95.2%	79.5%	94.2%	93.5%	77.8%
Pruning 50% + QAT INT8	97.6%	97.0%	80.3%	96.3%	95.6%	78.8%

Fig. 3. Qualitative results of the optimized LP detector on image samples from two benchmark LPD datasets used in our experiments. From top to down, the first row presents images of baseline annotations and the second row presents inference results from the final optimized ONNX engine, highlighting the robustness of the compressed architecture.

detection accuracy and inference speed for the evaluated optimization strategies. The baseline model achieves the highest detection accuracy but with the lowest inference speed in term of FPS. Structured pruning progressively increases inference speed while slightly reducing the mAP. Quantization significantly improves runtime performance while preserving most of the detection capability. The combined pruning and quantization strategy provides the best trade-off, achieving near-baseline accuracy while reducing latency by almost 5× compared to the original PyTorch model, making it suitable for real-time LPD on edge CPUs. Contrary to the expected behavior, where the application of a combined pruning of 50% and quantization is expected to cause a slight drop in the accuracy of the object detection model, the results showed a slight increase in the mAP metric. This can be attributed to the increased sensitivity of the FP32 pruned model to numerical noise, as well as the effect of the INT8 quantization process acting as a regularization mechanism to numerical noise, without changing the object detector architecture itself.

To assess the robustness of the proposed optimization pipeline, additional experiments were conducted on a secondary PGTLP dataset. The results confirm that the performance trends observed with pruning and quantization remain consistent, demonstrating the generalization capability of the approach across different data distributions.

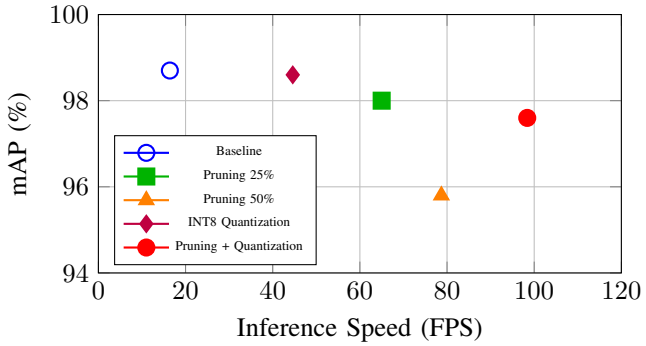

Fig. 4. Speed–Accuracy Pareto curve of the optimized YOLOv8 license plate detector under different optimization strategies. Points closer to the upper-right corner represent better trade-offs between detection accuracy and inference speed.

D. Latency and Throughput Results discussion

In this part, the performance of the improved YOLOv8-based LPD model in real-time execution is investigated, with a focus on the balance of latency and throughput for the Intel CPU Edge platform. Table II highlights the progressive performance improvements obtained through deployment optimization and model compression techniques. The base model has a latency of 61 ms, which translates to a processing rate of 16.4 FPS. When the model is translated to the ONNX Runtime, the performance of the model increases slightly by

TABLE II
YOLOv8 MODELS COMPLEXITY ON INTEL CPU EDGE HARDWARE: LATENCY AND THROUGHPUT COMPARISON.

Method	Latency (ms)	FPS	GFLOPs	#Param (M)	Model Size (MB)
Baseline Model	61.0	16.4	8.07	3.01	38.7
ONNX Runtime	55.3	18.1	8.07	3.01	12.1
OpenVINO IR FP32	50.8	19.7	8.07	3.01	6.3
PTQ INT8	25.6	39.0	4.07	3.01	3.0
QAT INT8	22.4	44.6	4.07	3.01	3.1
Pruning 25%	15.0	65.0	5.35	2.25	10.2
Pruning 50%	12.7	78.7	4.07	1.50	8.4
Pruning 50% + QAT INT8	15.3	98.4	4.07	1.50	2.8

lowering the latency to 55.3 ms. The graph is further optimized using the OpenVINO IR FP32 engine, which decreases latency to 50.8 ms and improves the throughput to 19.7 FPS. Finally, combining pruning with QAT yields the best real-time performance, at 98.4 FPS and an average latency of 15.3 ms. These findings show that hybrid compression techniques can enable high-speed LPD on resource-constrained edge devices while still providing robust detection performance.

Inference latency remains largely consistent between the LPD and PGTLP datasets, with variations of only ±1–2 ms across all optimization configurations. This is expected, as latency reflects the computational cost of executing the model architecture on the target CPU hardware, rather than the visual complexity of the input images.

IV. CONCLUSIONS AND FURTHER WORK

This paper presented an end-to-end optimization pipeline for real-time license plate detection on resource-constrained edge devices, leveraging the YOLOv8 architecture. Through the combination of structured pruning and INT8 quantization, the proposed approach reduces computational cost from 8.7 to 4.07 GFLOPs while preserving detection accuracy, achieving inference speeds exceeding 100 FPS on an Intel CPU. These results demonstrate the viability of deploying high-performance LPD systems on embedded hardware, with direct applicability to intelligent transportation systems, smart parking infrastructure, and robotic surveillance platforms.

Building on these findings, future work will pursue three research directions. First, the pipeline will be validated under real operational conditions through full deployment on the Pearl Guard mobile robot. Second, knowledge distillation will be investigated as a complementary compression strategy, enabling lightweight student models to inherit representational capacity from larger teacher networks, potentially surpassing the accuracy-efficiency trade-off achieved by pruning and quantization alone. Third, the system will be extended into a complete automatic license plate recognition pipeline, incorporating a transformer-based character recognition stage optimized for edge inference, toward a fully integrated and deployable edge AI solution.

REFERENCES

[1] X. Wang and W. Jia, "Optimizing Edge AI: A Comprehensive Survey on Data, Model, and System Strategies," 2025, doi: 10.32388/IZOHCH.

[2] D. Vaddepally, "Lightweight Deep Learning for Resource-Constrained Devices," *International Journal of Science and Advanced Technology (IJSAT)*, vol. 16, no. 2, Apr.–Jun. 2025, doi: 10.71097/IJSAT.v16.i2.6224.

[3] A. Ismail, M. Mehri, A. Sahbani, and N. E. B. Amara,"End-to-End License Plate Recognition System with Optimized Inference on Mobile Security Robots," in *Robotics and Mechatronics*, L. Romdhane, A. Mlika, S. Zeghloul, A. Chaker, and M. A. Laribi, Eds., Mechanisms and Machine Science, vol. 158. Cham, Switzerland: Springer, 2024, doi: 10.1007/978-3-031-59888-3-34.

[4] D. Maulana and R. K. Ramasamy, "Systematic Review of Quantization-Optimized Lightweight Transformer Architectures for Real-Time Fruit Ripeness Detection on Edge Devices," *Computers*, vol. 15, no. 1, p. 69, 2026, doi: 10.3390/computers15010069.

[5] G. Ferreira, M. H. d. N. Marinho, V. Severo, and F. Madeiro,"Optimization Strategies Applied to Deep Learning Models for Image Steganalysis: Application of Pruning, Quantization and Weight Clustering," *Applied Sciences*, vol. 15, no. 9, p. 4632, 2025, doi: 10.3390/app15094632.

[6] A. Moslemi, A. Briskina, Z. Dang, and J. Li, "A Survey on Knowledge Distillation: Recent Advancements," *Machine Learning with Applications*, vol. 18, p. 100605, 2024. doi: 10.1016/j.mlwa.2024.100605.

[7] R. Zhang, H. Jiang, W. Wang, and J. Liu, "Optimization Methods, Challenges, and Opportunities for Edge Inference: A Comprehensive Survey," *Electronics*, vol. 14, no. 7, p. 1345, 2025, doi: 10.3390/electronics14071345.

[8] AskariHemmat, MohammadHossein, et al. , 2022. "QReg: On Regularization Effects of Quantization". doi:10.48550/arXiv.2206.12372.

[9] AskariHemmat, MohammadHossein, et al. "Qgen: On the ability to generalize in quantization aware training." arXiv preprint arXiv:2404.11769, (2024).

[10] X. Wang, Q. Li, and W. Jia, "Cognitive Edge Computing: A Comprehensive Survey on Optimizing Large Models and AI Agents for Pervasive Deployment," *arXiv preprint arXiv:2501.03265*, 2025. [Online]. Available: https://arxiv.org/abs/2501.03265

[11] H. Li, Y. Mo, J. Chen, J. Chen, and J. Li, "Accurate Orah fruit detection method using lightweight improved YOLOv8n model verified by optimized deployment on edge device," *Artificial Intelligence in Agriculture*, vol. 15, no. 4, pp. 707–723, 2025, doi: 10.1016/j.aiia.2025.05.001.

[12] C. Pulgarín-Ospina, L. Launet, A. Colomer, and V. Naranjo, "Optimizing Deep Learning Models for Edge Computing in Histopathology: Bridging the Gap to Clinical Practice," *Procedia Computer Science*, vol. 246, pp. 2549–2557, 2024, doi: 10.1016/j.procs.2024.09.443.

[13] S. J. Wei, D. F. Al Riza, and H. Nugroho, "Comparative study on the performance of deep learning implementation in edge computing: Case study on plant leaf disease identification," *Journal of Agriculture and Food Research*, vol. 10, p. 100389, 2022, doi: 10.1016/j.jafr.2022.100389.

[14] Z. Li, P. Xu, X. Chang, L. Yang, Y. Zhang, L. Yao, and X. Chen, "When object detection meets knowledge distillation: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 8, pp. 10555–10579, 2023.