Edge-Cloud Offloading for Real-Time Perception of Resource-Constrained AMR in IIoT Environments

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

Matteo De Marchi
Industrial and Energy Engineering
Free University of Bozen - Bolzano
Bolzano, Italy
matteo.demarchi@unibz.it

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

***Abstract*—Autonomous mobile robot (AMR) platforms have emerged as practical solutions for logistics operations in dense industrial IoT environments. This paper presents a remote data-processing architecture for legacy mobile robots that offloads compute-intensive perception while preserving existing control software. The system uses ROS and ROSbridge over a secure Tailscale VPN to connect a mobile robot based on an NVIDIA Jetson Orin Nano 4GB to a remote NVIDIA Jetson Orin NX 16GB server across heterogeneous networks. In the studied use case, image topics published by the robot are processed remotely using Ultralytics YOLO, and the resulting information is returned to the robot and external monitoring clients. The work evaluates the feasibility of this architecture for IoT robotics in terms of communication stability, end-to-end latency, resource usage, and system bottlenecks. The results indicate stable connectivity and latency suitable for near-real-time operation, while reducing the computational burden on the robot. These findings suggest that the proposed approach provides a practical basis for upgrading legacy robotic platforms and supporting future multi-robot orchestration through containerized services.**

***Index Terms*—autonomous robots, computer vision, iiot, industry 5.0, yolo**

I. Introduction

The digital transformation of industrial environments is accelerating the adoption of connected robotic systems that combine autonomy, perception, and secure data exchange across distributed infrastructures [1], [2]. Within the Industry 5.0 paradigm, autonomous mobile robots (AMRs) are expected not only to automate repetitive tasks, but also to collaborate safely with human operators, interact with cyber–physical production systems, and adapt to changing operational contexts [3]. These requirements place strong emphasis on interoperability, cybersecurity, and the integration of advanced artificial intelligence services into existing robotic assets without disrupting proven industrial workflows [4], [5]. Robot Operating System (ROS) has become a widely used middleware for mobile robotics, providing modular communication, reusable software components, and practical integration of sensors, control logic, and higher-level applications [6]. However, many industrial and research platforms still rely on legacy ROS Noetic deployments and embedded hardware designed before the current generation of high-throughput edge AI accelerators and compact vision models. In such situations, enabling modern perception directly on the robot often requires replacing the main computing board, migrating the software stack, and adapting validated robot code to a new execution environment.

This paper addresses this limitation through a remote-processing solution that preserves the legacy robot architecture while enabling the integration of recent YOLO versions and AI capabilities. The target platform is ASTIbot, a mobile robot developed in our previous work for smart-manufacturing and logistics scenarios, which is currently based on ROS Noetic and legacy code [7]. Figure 1 illustrates the proposed communication architecture, which connects the robot, the perception and middleware stack, and user-facing services through a secure overlay network. In the proposed configuration, the robot publishes telemetry, image, and control data through ROS and ROSbridge while connected via Wi-Fi to a 4G network, while a remote server based on NVIDIA Jetson Orin NX 16GB performs the compute-intensive perception tasks on a separate enterprise network.

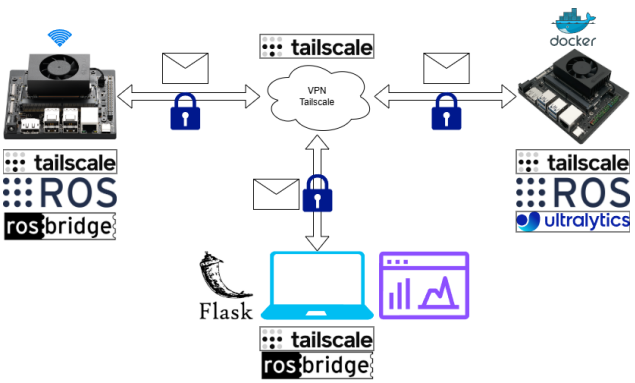

Fig. 1. System communication architecture

This architecture enables remote monitoring, command exchange, dashboard access, and perception offloading without directly exposing internal middleware endpoints, thereby

This research was partially funded by the "Building Opportunities for SMEs Sustainable Entrepreneurship with Artificial Intelligence in Industry 5.0" (BOOST-AI), project number 2024-1-RO01-KA220-HED-000246238, co-funded by the European Union within the Erasmus+ program. The support of the Transdisciplinary Association for Cutting Edge Research and Information (TRACE) is gratefully acknowledged.

improving modularity, simplifying deployment across heterogeneous devices, and supporting scalable supervision and interaction in Industry 5.0 robotic scenarios. It is motivated by a practical Industry 5.0 use case in which mobile robots act as flexible add-ons to smart manufacturing lines and intralogistics workflows, where interoperability among autonomous vehicles, industrial networks, cloud-edge services, and human operators is essential. Our previous work [7] showed that a local AI server can be connected to the robot to support advanced interaction, including natural-language task requests. Here, we extend that direction by studying a remote inference architecture that supports secure offloading of visual perception over heterogeneous networks and can serve as a foundation for future multi-robot or swarm-based systems. The use of containerized server-side services further improves portability, maintainability, and scalability when deploying new perception pipelines.

The remainder of the paper is organized as follows. Section II reviews related work on secure robotic connectivity, ROS-based distributed systems, and AI-enabled perception. Section III presents the methodology based on the embedded ROS architecture and the YOLO family of deep learning object detection models. Section IV discusses the obtained results. Section V summarizes the conclusions, and Section VI outlines future extensions toward multi-robot operation scenarios.

II. Related work

Prior work on distributed robotic systems has highlighted the importance of extending ROS communication beyond a single local network. An early cloud-oriented ROS deployment over a virtual private network (VPN) showed that robotic services and sensor streams can be securely exposed to remote computing resources, establishing a practical baseline for cloud robotics and off-board processing [8]. More recent studies have examined this problem in the broader context of the Internet of Robotic Things (IRT) and the edge–cloud continuum. In particular, ROS 2 communication strategies based on middleware optimization and VPN-supported interconnection have been evaluated for scalable robotic networking [9], while broader surveys have summarized the main challenges and recent advances in ROS 2 software, libraries, and applications [10]. Complementing these efforts, ROS Gateway has been proposed as a middleware approach for improving ROS availability across heterogeneous network environments, with experimental evidence showing improved topic delivery and delay behavior for data-intensive robotic applications [11]. From a software-engineering perspective, systematic mapping studies further show that ROS remains central to robotics development workflows, reinforcing the importance of maintainable and interoperable distributed deployments [6]. Together, these works confirm that multi-network ROS connectivity is both feasible and increasingly relevant for distributed perception and control.

A second group of studies is directly related to the industrial and robotic context addressed in this paper. Our previous work on flexible manufacturing systems for Industry 4.0 and Industry 5.0 applications established the broader production environment in which mobile robotic platforms can act as adaptable cyber–physical assets [12]. We also investigated embedded learning algorithms for vision-based defect detection on the NVIDIA Jetson Orin NX 16GB platform, showing the relevance of edge artificial intelligence for sustainable manufacturing applications and providing an initial benchmark of YOLO-family models under embedded constraints [13]. In addition, we previously studied deep reinforcement learning for controlling an open mobile robotic platform [14] and later explored large-language-model-enhanced control for smart-industry robotics, including natural-language task interaction and higher-level decision support [7]. An experimental robotic design for indoor mapping tasks is presented in [15]. Broader reviews on AI-enabled robotics and connected autonomous systems emphasize the growing need to couple perception, autonomy, and networked infrastructure in operational environments [1], [2]. These studies motivate the present work from two directions: first, the need to preserve and extend legacy robotic platforms already integrated into industrial workflows; and second, the need to connect perception, autonomy, and human-facing services through secure distributed infrastructures. From a cybersecurity perspective, the use of Tailscale and related protected networking mechanisms in Internet of Things (IoT) environments further supports the adoption of lightweight secure overlays for industrial robotic communication [16]. This motivation is consistent with Industrial IoT (IIoT) security surveys and recent analyses of edge-computing-integrated industrial cybersecurity, which underline the need for secure communication overlays, layered protection, and resilient distributed architectures [4], [5], [17].

The perception component of the present study builds on the rapid evolution of the Ultralytics YOLO family and recent comparative analyses of object-detection models for robotics. The official Ultralytics release documentation for YOLO26 and subsequent architectural analyses emphasize improvements aimed at efficient deployment, refined feature fusion, and end-to-end detection behavior suitable for edge inference [18], [19]. Comparative evaluations of YOLO variants for autonomous robotic applications have likewise shown that model selection requires balancing latency, model size, and detection performance according to deployment constraints [20]. In parallel, the availability of open-source integration artifacts, such as our accompanying project repository, reinforces the importance of reproducible tooling and deployable software stacks when transferring AI-enabled robotics solutions from experimentation to operational settings [21]. In contrast to the cited literature, the present paper combines secure ROS and ROSbridge communication over a VPN, remote offloading from a legacy ROS Noetic mobile robot, and an experimental evaluation of multiple YOLO deployment formats on a Jetson Orin NX 16GB server, with emphasis on latency, stability, and practical suitability for future multi-robot scenarios.

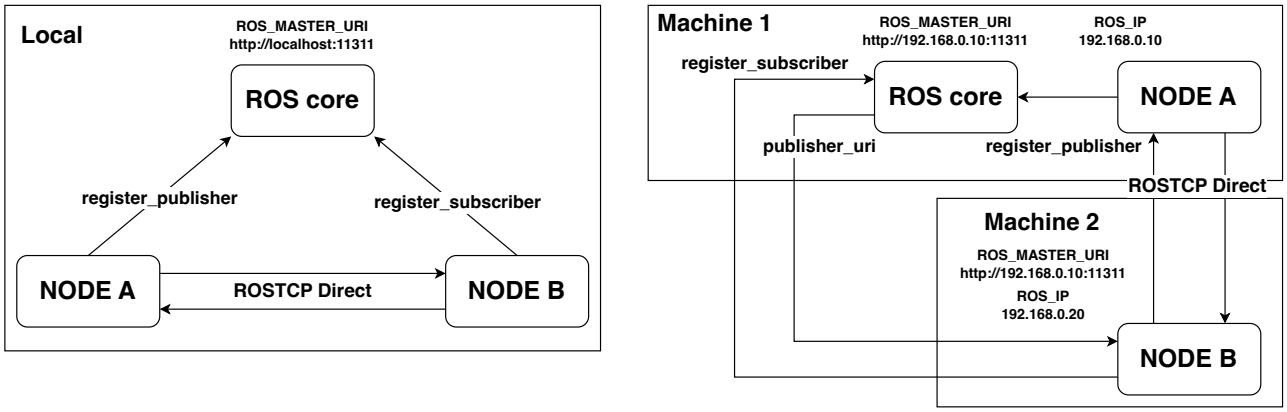

Fig. 2. ROS communication layouts for single and multiple platforms

III. Methodology

A. ROS 2 architecture considerations

ROS provides modular middleware in which robot functionality is decomposed into nodes that exchange typed messages through well-defined interfaces, thereby supporting the flexible integration of sensing, control, and supervisory functions. Figure 2 compares the ROS communication layouts considered for local and distributed deployment. The first view shows the single-platform ROS organization, while the second highlights the extended communication arrangement used to integrate remote services and external clients through the proposed architecture. Communication is primarily organized around topics for asynchronous publish/subscribe streams and services for synchronous request/response interactions, while node discovery, name resolution, and parameter management are coordinated through the central ROS Master.

In this architecture, the Master belongs to the control plane, whereas application data typically flows directly between peer nodes over transport protocols such as TCPROS and, in selected cases, UDPROS. This separation enabled rapid prototyping, supported strong ecosystem growth, and simplified the integration of modular software components, but it also introduced a central dependency for discovery and multi-machine coordination.

By contrast, ROS 2 preserves the node/topic/service abstraction and extends it with actions for long-running, feedback-oriented tasks, while replacing master-centric coordination with Data Distribution Service (DDS)/Real-Time Publish-Subscribe (RTPS)-based distributed discovery and configurable Quality of Service (QoS) policies that better support latency, reliability, and scalability tuning in distributed robotic systems [10]. As illustrated in Figure 3, ROS 2 no longer relies on a `ROS_MASTER_URI`-style discovery endpoint. Instead, it depends on middleware backends, decentralized participant discovery, and network conditions compatible with the selected DDS implementation; setting a common `ROS_DOMAIN_ID` is sufficient to allow participants to join the ROS network [9].

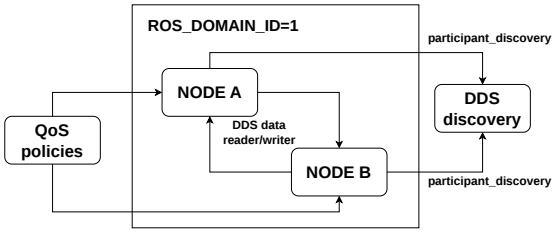

Fig. 3. ROS 2 communication model based on DDS-driven distributed discovery and peer data exchange

In practical multi-machine ROS 1 deployments, all hosts typically point to a common `ROS_MASTER_URI`, while each machine advertises a reachable address through `ROS_IP` or `ROS_HOSTNAME`; once registration is complete, nodes communicate directly across the network. In the proposed solution, this mechanism is mapped onto the Tailscale VPN by assigning Tailscale client addresses to both `ROS_IP` and `ROS_MASTER_URI`. This ensures that ROS 1 name resolution and peer-to-peer topic transport operate through the secure VPN address space rather than through public or site-local network interfaces. This choice provides deterministic addressing across heterogeneous networks and simplifies deployment when the robot and server are attached to different access domains. For ROS 2, a Zenoh/DDS-based solution for distributed communication has already been described in the literature [9]; in this case, the communication stack should rely on Zenoh [22] rather than the default Fast DDS, while the Tailscale client addresses of the participating hosts support remote node discovery and transport across the VPN. Under this configuration, the secure Tailscale overlay provides the addressing substrate for distributed ROS 2 participants, and the Zenoh-based layer extends discovery and data exchange across remote sites while preserving protected end-to-end connectivity.

The server-side implementation is designed with scalability in mind, especially for future robot-swarm scenarios.

Because the processing environment is containerized, separate Docker containers can be instantiated with their own ROS 1 environment-variable definitions. In particular, each container can use dedicated values for `ROS_MASTER_URI` and `ROS_IP`, allowing it to act as an isolated communication context associated with a specific robot or task flow. This approach allows multiple logically independent ROS environments to coexist on the same NVIDIA Jetson Orin NX 16GB server while sharing compute resources for perception, monitoring, and orchestration services. In this way, containerization not only simplifies deployment and maintenance, but also provides a practical technical path toward multi-robot integration, secure network segregation, and service replication within the same off-board processing infrastructure [11].

B. YOLO deep learning object detection

YOLO26 is a recent single-stage object detector from Ultralytics, developed for efficient visual perception on edge-oriented platforms. Figure 4 illustrates the YOLO26 processing architecture considered in this work. The model follows the general design direction of recent Ultralytics detectors while refining the three main stages of the perception pipeline: feature extraction, feature fusion, and detection.

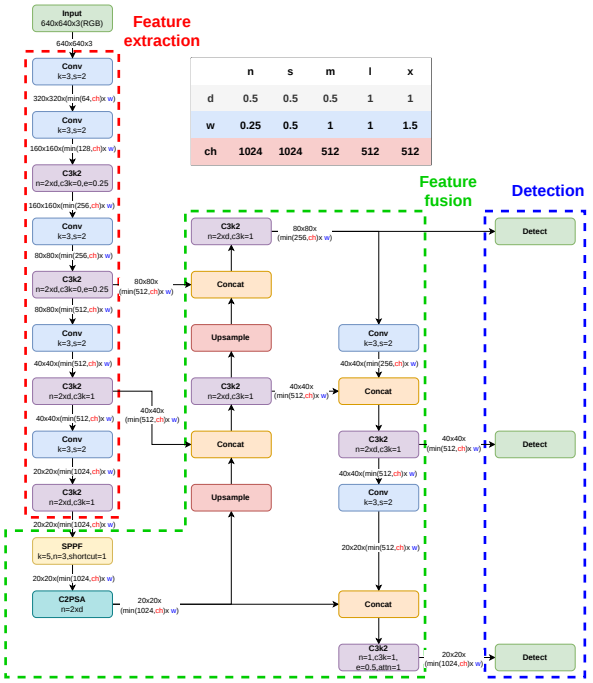

Fig. 4. YOLO26 architecture (based on [19])

In the feature-extraction stage, YOLO26 preserves an efficient hierarchical backbone structure while improving the propagation of high-level visual representations. In the feature-fusion stage, it enhances multi-scale aggregation through updated Spatial Pyramid Pooling Fast (SPPF) components and attention-supported fusion blocks, with the goal of preserving contextual information more effectively, especially for small objects. In the detection stage, the architecture moves away from Distribution Focal Loss (DFL)-dependent regression toward direct bounding-box regression in an end-to-end, Non-Maximum Suppression (NMS)-free formulation, supported by dual assignment during training and one-to-one prediction during inference. Relative to YOLO11, these changes are presented primarily as an optimization-oriented refinement of the fusion and detection pipeline rather than a complete redesign. For embedded deployment, this distinction is relevant because it directly affects the trade-off between latency and detection performance on resource-constrained systems, complementing our previous benchmarking analysis of multiple YOLO families on the NVIDIA Jetson Orin NX 16GB platform [13].

IV. Results

Figure 5 summarizes the mean inference time measured for the evaluated models and deployment configurations. The reported mean values were calculated as:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1}$$

where x_i is the inference time measured for frame i, and N is the total number of processed frames. The observed trend is consistent with the expected scaling behavior of the YOLO family: larger models generally require longer processing times because they contain more parameters and perform more computation per frame.

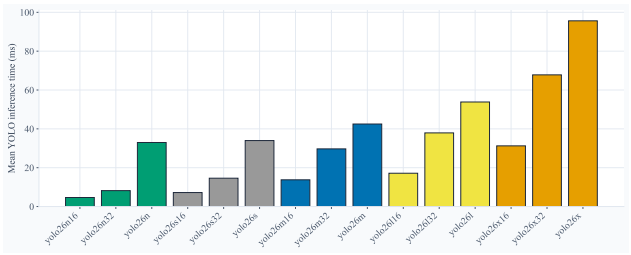

Fig. 5. Mean model inference time across evaluated configurations

At the same time, model size alone does not determine runtime. The execution format is equally important, as TensorRT-based NVIDIA engine deployments are better aligned with the Jetson Orin NX 16GB platform than baseline PyTorch execution. In practice, the FP32 and especially FP16 TensorRT variants provide a more favorable latency–accuracy trade-off, showing that optimized deployment formats are essential when the target environment is an NVIDIA edge device.

As a continuation of the inference-time analysis, Figure 6 reports the mean GPU utilization observed for the same configurations. The average accelerator load was calculated using the mean formula in (1), with x_i representing the GPU utilization percentage measured at sample i. This result helps identify which deployment formats use the accelerator more

effectively when perception is offloaded from the robot to the remote Jetson Orin NX server. The optimized TensorRT variants achieve better timing while avoiding unnecessary computational pressure, whereas less optimized formats can consume comparable or higher accelerator resources for the same workload while still delivering slower inference. From a system-design perspective, this is important because the goal is not only to execute the model remotely, but also to allocate GPU resources efficiently so that the server can preserve headroom for communication, monitoring, and future multi-robot extensions.

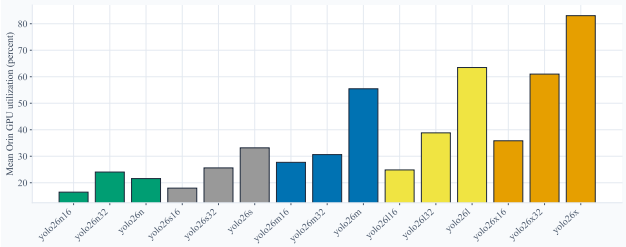

Fig. 6. Mean GPU utilization across evaluated configurations

Figure 7 complements the previous observations by showing the mean power consumption measured across the evaluated inference configurations.

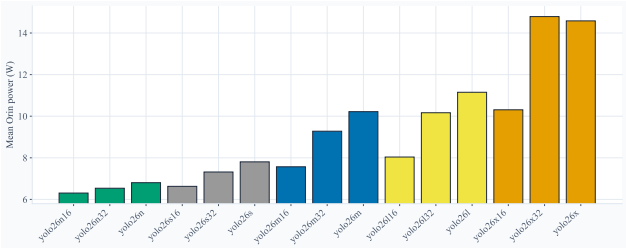

Fig. 7. Mean power consumption across evaluated configurations

In this study, we used (1) to compute the reported mean power in watts, where x_i denotes the instantaneous power measured at sample i. The consumed energy was then estimated from the timestamped power samples as follows:

$$E_{\mathrm{Wh}} \approx \frac{1}{3600} \sum_{k=1}^{M-1} P_k \left(t_{k+1} - t_k\right) \tag{2}$$

where P_k is the measured power and $(t_{k+1} - t_k)$ is the time interval between consecutive samples. In general, larger models draw more power because they impose a heavier load on the GPU, require more memory activity, and increase the thermal-management effort needed to sustain execution. This effect is especially relevant for embedded AI servers, where increased model complexity affects not only latency but also resource demand and cooling requirements. In our experiments, the Jetson Orin NX 16GB platform operated in the 40 W maximum-performance mode; therefore, the reported values correspond to an aggressive, performance-oriented configuration intended to reveal the practical cost of each model and deployment format.

Figure 8 combines the previous two perspectives into a joint latency–power view. This representation highlights that the best operating point is determined not by latency or power alone, but by the balance between them.

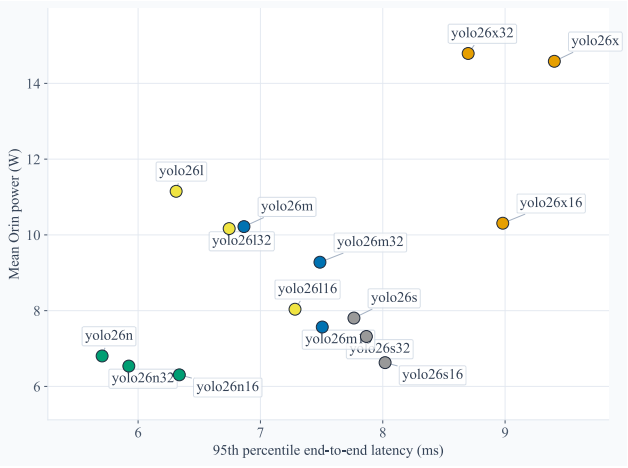

Fig. 8. Latency–power trade-off across evaluated configurations

Configurations located near the lower-left region of the plot are preferable because they reduce response time without incurring a disproportionate energy cost. In contrast, slower and less optimized models tend to move toward a less favorable region, where both latency and power increase simultaneously. This result confirms that deployment-format optimization is a central part of the offloading strategy, rather than merely an implementation detail. Figure 9 presents the 95th-percentile end-to-end communication latency measured across the evaluated configurations. In this study, we used:

$$t_{\mathrm{e2e}}^{(i)} = T_{\mathrm{proc}}^{(i)} - T_{\mathrm{cam}}^{(i)} \tag{3}$$

to compute the end-to-end latency for frame i, where $T_{\mathrm{cam}}^{(i)}$ is the timestamp of camera-image publication and $T_{\mathrm{proc}}^{(i)}$ is the timestamp of processed-image publication. The 95th percentile was then computed from the resulting samples as follows:

$$r = \lceil 0.95\, N \rceil, \qquad p_{95} = x_{(r)} \tag{4}$$

after sorting the observations in ascending order. Thus, 95% of the observations are less than or equal to the reported value.

Although the VPN and ROSbridge communication remain stable, the total end-to-end latency still increases for larger models because image processing extends beyond raw inference alone. When a model detects more objects, the post-processing stage becomes more demanding; similarly, when the model itself is slower, incoming images begin to queue before processing is completed. This behavior is critical in the present setup because the camera publishes at 30 Hz, corresponding to a frame period of approximately 33.3 ms. Once

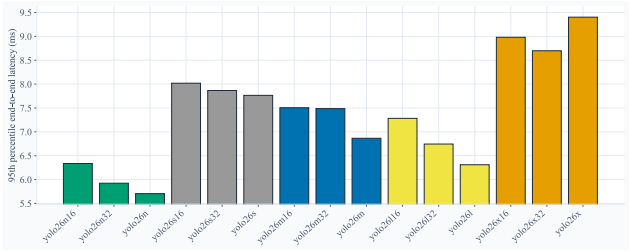

Fig. 9. 95th-percentile end-to-end communication latency across evaluated configurations

the effective processing time exceeds this pacing, buffering occurs and the system begins to lose real-time behavior, even if the network transport itself remains functional.

The same interpretation is reinforced by Figure 10, which summarizes the mean total processing time measured at the node level. In this study, the YOLO node time was computed as follows:

$$t_{\text{yolo}}^{(i)} = t_{\text{decode}}^{(i)} + t_{\text{pre}}^{(i)} + t_{\text{inf}}^{(i)} + t_{\text{post}}^{(i)} \tag{5}$$

where each term corresponds to the decoding, preprocessing, inference, and post-processing time for frame i, respectively. The mean value shown in the figure was then computed using (1).

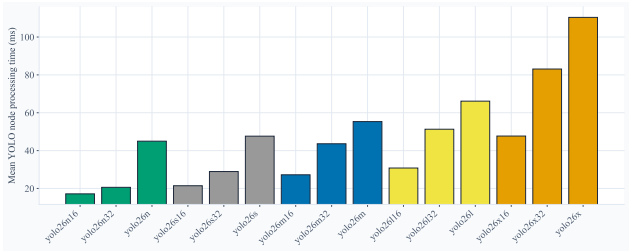

Fig. 10. Mean total node processing time across evaluated configurations

Because this metric includes preprocessing, inference, and post-processing, it better reflects the true time budget of the complete remote-perception pipeline. The practical conclusion is that the optimal operating point is not necessarily the fastest model or the largest one. Instead, the preferred configuration is a middle ground that can keep pace with the 30 Hz camera publisher while using hardware resources efficiently. A model that is much faster than the sensing rate may sacrifice useful accuracy, whereas a model that is slower than the frame budget causes queueing, delay accumulation, and degraded responsiveness.

Finally, Figure 11 shows the histogram of the measured round-trip time (RTT) values. The average RTT was computed using (1), while the upper-tail delay was computed using the 95th-percentile formula in (4). The distribution indicates that most operating points remain compatible with the task, with typical update rates of approximately 50 Hz for most tested configurations and about 20 Hz in less favorable cases associated with the larger models. These rates correspond to RTT values of roughly 20 ms in the more responsive cases and around 50 ms in the slower ones.

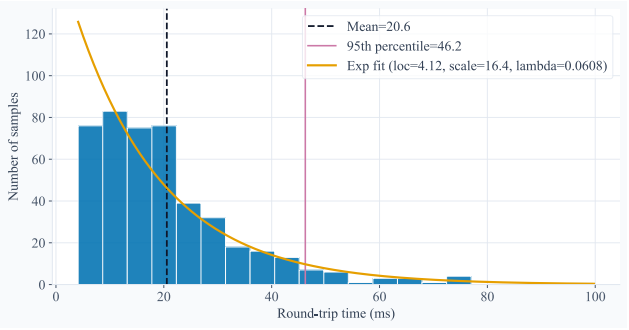

Fig. 11. Histogram of measured RTT values

Both the mean and p95 remain relevant in practice: the mean characterizes typical operation, whereas p95 indicates whether occasional latency excursions may affect interactive robot supervision. The measured RTT values can also be represented by an exponential distribution with the identified location and scale parameters:

$$\begin{aligned} f(x) &= \begin{cases} \lambda \exp[-\lambda(x-\text{loc})], & x \geq \text{loc}, \\ 0, & x < \text{loc}, \end{cases} \\ \text{scale} &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \text{loc}), \qquad \lambda = \frac{1}{\text{scale}} \end{aligned} \tag{6}$$

This formulation provides a compact probabilistic description of the observed delay behavior. In the present experiments, the communication layer remains sufficiently stable for remote perception, while the dominant limitation shifts toward model-processing time for the heavier networks.

Table I reports the model characteristics and inference times obtained during the experimental campaign on the NVIDIA Jetson Orin NX 16GB platform. The comparison includes PyTorch execution as well as TensorRT deployments in FP32 and FP16 precision, enabling a direct assessment of the trade-off between model size, validation accuracy, and runtime performance.

As expected, the TensorRT variants provide substantially lower inference times than the original PyTorch models, while FP16 achieves the most favorable latency figures across all tested network sizes. These values provide the quantitative basis for selecting the most appropriate deployment format for real-time remote perception and confirm that the Orin NX 16GB board can sustain demanding YOLO workloads with headroom for future extensions. Figure 12 presents the monitoring dashboard used to supervise the robotic system during operation. The interface is part of our software stack and is implemented in Python and Flask to provide an accessible client solution for users and monitoring stations that are not Linux-based. It collects and displays data from topics published by both the robot and the remote server

TABLE I
YOLO26 MODELS COMPARISON

Format	Model	Size (MB)	$\text{mAP}^{\text{val}}_{50\text{-}95}$	Inference (ms)
PT	YOLO26n	5.5	48.0	32.97
	YOLO26s	20.4	56.6	33.96
	YOLO26m	44.3	62.3	42.49
	YOLO26l	53.2	62.5	53.83
	YOLO26x	118.8	65.6	95.62
TRT FP32	YOLO26n	11.9	47.7	8.19
	YOLO26s	40.3	56.6	14.59
	YOLO26m	83.9	62.1	29.68
	YOLO26l	101.9	62.5	37.91
	YOLO26x	225.8	65.9	67.80
TRT FP16	YOLO26n	8.4	47.9	4.65
	YOLO26s	22.7	56.5	7.19
	YOLO26m	44.8	62.2	13.75
	YOLO26l	53.5	62.2	17.14
	YOLO26x	115.3	66.3	31.24

through ROSbridge, consolidating live perception outputs, system status indicators, and performance trends into a single user-facing environment. By combining real-time visualization with accessible control and diagnostic views, the dashboard supports rapid situational awareness and facilitates informed decision-making during remote supervision and experimental evaluation, while avoiding reliance on proprietary software applications.

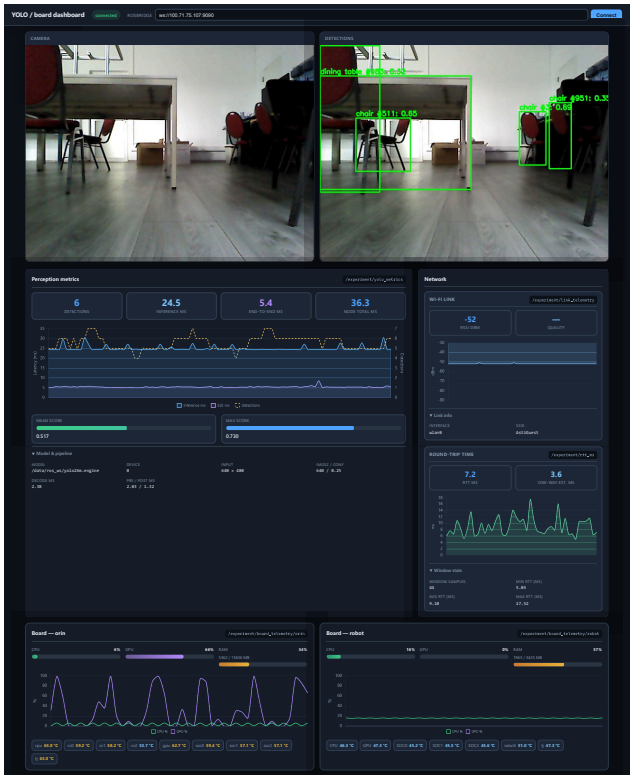

Fig. 12. Monitoring dashboard interface

Figure 13 shows the robot control and telemetry dashboard developed as part of our software stack and deployed locally on the robot board using Node-RED. The dashboard connects to ROSbridge to exchange messages with the ROS stack, enabling browser-based access to command interfaces, state feedback, and live telemetry without requiring a Linux-based environment or a native ROS installation on the operator side.

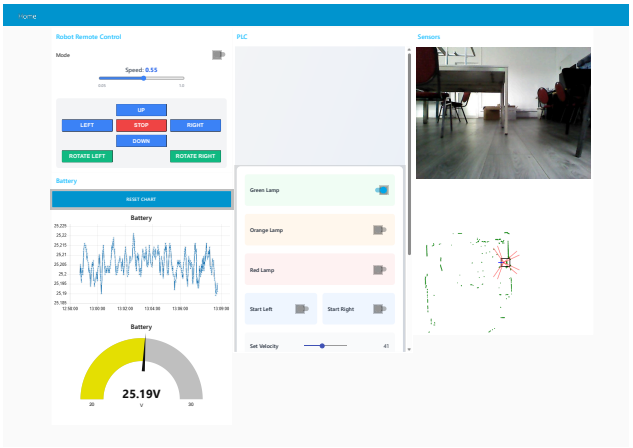

Fig. 13. Robot control and telemetry dashboard

In practice, this architecture provides a platform-independent human–machine interface that can be accessed from any modern web browser on desktop systems, tablets, or smartphones, while still exposing direct robot-control functions and topic-level observability. By integrating control widgets with real-time telemetry visualization in a lightweight web application, the dashboard supports low-friction supervision, shortens operator response time, and improves the technical accessibility of the robotic platform during deployment and testing.

V. CONCLUSION

The proposed ROS- and VPN-based architecture demonstrates that legacy mobile robots can be extended with modern AI perception services without replacing the onboard computing platform or refactoring validated control software. By offloading YOLO inference to a remote NVIDIA Jetson Orin NX 16GB server, the system preserves secure interoperability across heterogeneous networks while maintaining latency compatible with real-time robotic supervision. The work also shows that new software packages can be integrated with legacy robotic components through Docker containers, enabling ROS-based communication services and modern vision pipelines, such as Ultralytics YOLO, to coexist in a structured and reproducible deployment on NVIDIA-based embedded systems. In this sense, the proposed solution helps bridge the gap between legacy robotic software stacks and current edge-AI processing capabilities.

From an experimental perspective, the study confirms the feasibility of secure VPN communication and end-to-end latency analysis for a robotic application centered on YOLO-based object detection. The resulting client–server architecture

combines ROS, ROSbridge, and Tailscale VPN to provide a secure connection model for remote perception, command exchange, and system observability across heterogeneous networks. In addition, the developed IoT dashboards extend access to monitoring and control functions beyond ROS-native environments, making the platform more accessible to non-ROS systems and user devices. Taken together, these contributions support the feasibility of the proposed framework for Industry 5.0 scenarios in which cybersecurity, modularity, observability, and cost-effective modernization are key deployment requirements.

VI. Future Work

Future work will extend the present single-robot validation toward the coordinated operation of multiple robotic platforms sharing the same remote inference and middleware infrastructure, with particular emphasis on robotic-swarm scenarios in which multiple ROS machines and mobile robots must be supervised, synchronized, and orchestrated within a common secure architecture. In this direction, an important next step is to develop orchestration mechanisms for task assignment, communication management, and service coordination across distributed robot fleets connected through the proposed client–server framework.

A second direction concerns the integration of higher-level intelligent interaction services on top of the validated perception pipeline. Building on our previous work on large-language-model-assisted robotic control, future extensions may combine LLM-based user interaction with YOLO object detection to provide richer context-aware supervision, natural-language command interfaces, and more intuitive access to robot status and scene understanding. Such a combination could improve operator usability while preserving the modular separation between perception, control, and user-facing services.

Another relevant objective is to bridge the proposed architecture with ROS 2 platforms and heterogeneous industrial robotic systems, including fixed manipulators and robotic arms used in production cells. Extending interoperability across ROS 1, ROS 2, and industrial robot controllers would broaden the applicability of the framework beyond mobile robots and support mixed deployments in which perception, manipulation, and supervisory services interact across different middleware generations and device classes.

Finally, future work may investigate the evolution of the platform toward a turnkey embedded supervision solution for production lines, including deployment across multiple remote servers and resource management at the cluster level. Such an extension would make it possible to balance perception workloads, allocate compute resources dynamically, and support resilient multi-server operation in larger industrial environments. Altogether, these directions will help assess the suitability of the proposed framework for swarm robotics, smart intralogistics, adaptive cyber–physical production systems, and scalable industrial supervision.

References

[1] A. Rayhan, "Artificial intelligence in robotics: From automation to autonomous systems," *IEEE Transactions on Robotics*, vol. 39, no. 7, pp. 2241–2253, 2023.

[2] M. M. Rana and K. Hossain, "Connected and autonomous vehicles and infrastructures: A literature review," *International Journal of Pavement Research and Technology*, vol. 16, no. 2, pp. 264–284, 2023.

[3] B. Martini, D. Bellisario, and P. Coletti, "Human-centered and sustainable artificial intelligence in industry 5.0: Challenges and perspectives," *Sustainability*, vol. 16, no. 13, p. 5448, 2024.

[4] A. M. Alnajim, S. Habib, M. Islam, S. M. Thwin, and F. Alotaibi, "A comprehensive survey of cybersecurity threats, attacks, and effective countermeasures in industrial internet of things," *Technologies*, vol. 11, no. 6, 2023.

[5] T. Zhukabayeva, L. Zholshiyeva, N. Karabayev, S. Khan, and N. Alnazzawi, "Cybersecurity solutions for industrial internet of things–edge computing integration: Challenges, threats, and future directions," *Sensors*, vol. 25, no. 1, p. 213, 2025.

[6] M. Albonico, M. Đorđević, E. Hamer, and I. Malavolta, "Software engineering research on the robot operating system: A systematic mapping study," *J. of Systems and Software*, vol. 197, p. 111574, 2023.

[7] M.-D. Pavel, G. Stamatescu, M. Chodnicki, and C. G. Amza, "Llm-enhanced control of a mobile robotic platform for smart industry," *Applied Sciences*, vol. 16, no. 4, 2026. [Online]. Available: https://www.mdpi.com/2076-3417/16/4/1680

[8] J. Z. Lim and D. W.-K. Ng, "Cloud based implementation of ros through vpn," in *2019 7th International Conference on Smart Computing & Communications (ICSCC)*, 2019, pp. 1–5.

[9] L. Fu, G. Kapoor, L. Militano, G. T. Carughi, and T. M. Bohnert, "Iort ros 2 applications: Evaluating zenoh and vpn for robotic networking in the edge-cloud continuum," in *2025 IEEE Symposium on Computers and Communications (ISCC)*, 2025, pp. 1–6.

[10] A. S. Al-Batati, A. Koubaa, and M. Abdelkader, "Ros 2 key challenges and advances: A survey of ros 2 research, libraries, and applications," *preprint*, 2024.

[11] B.-Y. Song and H. Choi, "Ros gateway: Enhancing ros availability across multiple network environments," *Sensors*, vol. 24, no. 19, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/19/6297

[12] M.-D. Pavel and G. Stamatescu, "Flexible manufacturing system for enhanced industry 4.0 and industry 5.0 applications," in *2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, 2024, pp. 483–490.

[13] ——, "Sustainable manufacturing application of embedded learning algorithms for vision-based defect detection under the industry 5.0 paradigm," in *2025 33rd Mediterranean Conference on Control and Automation (MED)*, 2025, pp. 185–190.

[14] M.-D. Pavel, S. Roșioru, N. Arghira, and G. Stamatescu, "Control of open mobile robotic platform using deep reinforcement learning," in *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer, 2022, pp. 368–379.

[15] F.-D. Secuianu and C. Lupu, "Implementation of a home appliance mobile platform based on computer vision: System configuration and calibration," *UPB Scientific Bulletin, Series C: Electrical Engineering and Computer Science*, vol. 82, no. 4, pp. 41–54, 2020.

[16] D.-F. Hrițcan and D. Balan, "Using tailscale and pfsense for security and anonymity of iot environments," in *2024 International Conference on Development and Application Systems (DAS)*, 2024, pp. 91–94.

[17] C. Ni and S. C. Li, "Machine learning enabled industrial iot security: Challenges, trends and solutions," *Journal of Industrial Information Integration*, vol. 38, p. 100549, 2024.

[18] G. Jocher and J. Qiu, "Ultralytics yolo26," 2026, software repository. [Online]. Available: https://github.com/ultralytics/ultralytics

[19] P. Hidayatullah and R. Tubagus, "Yolo26: A comprehensive architecture overview and key improvements," 2026. [Online]. Available: https://arxiv.org/abs/2602.14582

[20] R. Vaghela, D. Vaishnani, J. Sarda, A. Thakkar, Y. Nasit, B. Brahma, and A. K. Bhoi, "Optimizing object detection for autonomous robots: a comparative analysis of yolo models," *Measurement*, p. 118676, 2025.

[21] M. D. Pavel, "Ros yolo vpn communication," 2026, project repository. [Online]. Available: https://github.com/mihaidanielPavel/vpn_ros_communication_with_yolo

[22] "Ros2 rmw zenoh," project repository. [Online]. Available: https://github.com/ros2/rmw_zenoh