

A Testbed for Enabling Intelligence Offloading in Edge Computing for Object Detection

Reginaldo Luna¹, Nicole Correa Ramos¹, Ronan Pereira¹, Hermes Senger²

¹ Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – CEP: 13565-905 – São Carlos – SP – Brazil

{reginaldojunior,nicole.ramos,ronanpereira}@estudante.ufscar.br

²Instituto de Ciências Matemáticas e de Computação – ICMC/USP

hsenger@icmc.usp.br

Abstract. *The use of Machine Learning models and IoT devices for low-latency and energy-efficient decision-making continues to grow. This paper presents and evaluates a ROS 2-based edge computing framework for distributed You Only Look Once (YOLO) object detection on heterogeneous Single Board Computers for autonomous mobile robotics. Experimental results show that performance depends on hardware roles and scheduling decisions rather than computational power alone. Offloading inference to a GPU server reduced inference latency by approximately 30% and increased throughput by more than one order of magnitude, while the proposed Random Weighted scheduler improved load balancing and increased successful object detections in heterogeneous environments.*

1. Introduction

The increasing use of robots and Internet of Things (IoT) devices worldwide has driven research focused on improving distributed computing systems. Some of these studies explore how data can be distributed in Edge Computing to reduce delays in latency-sensitive applications while improving energy efficiency and reducing carbon emissions in vehicular Edge Computing (EC) environments [Zhou et al. 2018].

Additionally, in many scenarios, there is a rising adoption of machine learning techniques to perform tasks including data classification, anomaly detection, network intrusion detection for cybersecurity, real-time image classification and segmentation, driver assistance systems, autonomous driving, among others. Although Edge Computing (EC) provides several advantages [Shi et al. 2016], it also introduces important challenges, particularly the high heterogeneity of edge devices.

Thus, robotic systems are becoming increasingly connected, networked, and distributed, driven by the adoption of modern robotics middleware architectures [Zhang et al. 2022]. In this context, Edge Computing (EC) has emerged as a promising approach for robotic systems, offering several advantages, including reduced energy consumption, lower response latency, and improved computational performance for real-time applications.

In this work, we presented the design and implementation of such a testbed for distributed robotic systems, describing its architecture, implementation process, deployment challenges, and initial experimental results. The evaluation demonstrates the feasibility of computation offloading across heterogeneous edge devices, while also highlighting

practical challenges and trade-offs. Finally, we discuss the lessons learned, current limitations, and future research directions toward enabling efficient intelligence offloading in heterogeneous Edge Computing environments.

2. Edge Computing and Mobile Robotics

AI-driven workloads have become common in edge deployments. Object detection models such as YOLO and image classifiers based on convolutional networks often run directly on edge nodes to meet latency constraints that cloud offloading struggles to satisfy. Autonomous mobile robots, surveillance systems, and industrial inspection pipelines share this requirement: it is important for inference to occur close to the data source, within tight time and energy budgets. However, the heterogeneity of edge hardware — ranging from microcontrollers to GPU-equipped single-board computers — affects how these workloads are partitioned and scheduled across the infrastructure [Shi et al. 2016].

Robot Operating System 2 (ROS-2) has become the dominant middleware for coordinating perception, planning, and actuation in modern robotic platforms. Its publish-subscribe model, built on DDS (Data Distribution Service), exposes configurable Quality of Service (QoS) policies governing reliability, history depth, and message delivery guarantees, parameters that directly shape inference latency and throughput in distributed pipelines [Zhang et al. 2022]. Native support for heterogeneous node topologies and containerized deployment makes ROS-2 a natural integration layer between robotic applications and edge computing infrastructure.

3. A Testbed for Mobile Robotics on the Edge

In this paper, we propose a testbed that supports the experimental environment by interconnecting distributed machines and enabling intelligent inference allocation. The architecture leverages ROS 2 topics for inter-node communication and Docker containerization to integrate the server (*process-video*) and the client (*inference*) in a reproducible manner, while frame offloading is managed by the *dispatcher*.

The architecture workflow starts with the *process-video* module, responsible for preprocessing the incoming video stream before object detection is performed. This stage converts the stream into individual frames, records timestamps, and applies compression (if specified) to prepare the data for inference. The application also monitors performance metrics, including **latency**, **throughput**, and the **number of returned frames**.

Once preprocessed, frames are forwarded to the *dispatcher*, which is responsible for assigning each frame to the most suitable available machine for inference. To this end, different scheduling strategies can be employed, such as **Random Selection**, **Round-Robin** and **Weighted Choice**. Our analysis focused on weighted scheduling, in which machine selection is based on predefined or adaptive weights. Static weights are derived from machine behaviors and network metrics, following [Kashyap and Vidyarthi 2022]. In contrast, dynamic weights are continuously adjusted during system operation to reflect current conditions and resource availability [Kaur et al. 2021]. Following [Michaelis et al. 2024], we implemented a static Inverse Latency Weighting strategy based on historical profiling. The selection weight W_i for a known node i is defined as $W_i = 10 \cdot (L_i^{-1} / \sum_{j \in K} L_j^{-1})$, where L_i is its historical mean latency and K is the set of profiled nodes. Unprofiled nodes receive a default weight of 5.

After receiving the preprocessed frames from the *dispatcher*, a machine with the *inference* application executes the *YOLOv5n* algorithm to detect the objects present in the analyzed video stream. The results are subsequently returned to the *process-video* application, enabling the measurement of **inference latency** and the verification of potential **frame losses** throughout the processing pipeline.

4. Performance Evaluation

We conducted 280 experiments (10 repetitions per configuration) across 7 pairings of four devices: Jetson Nano, Raspberry Pi 5, a GPU Server (Xeon + 4xRTX 2080), and a Khadas VIM4 which was designated as the dummy robot in our experiments. We varied QoS policy Keep All and frame compression. Connected via a 1 Gbps Ethernet network to isolate computational latency, the testbed tracked total/inference/transmission latency, frame returns, and throughput. Since the VIM4 acted solely as the video source, it is excluded from inference metrics.

The device assigned to each role in the pipeline had asymmetric effects. The choice of inference server was the dominant factor: offloading to the GPU Server reduced inference latency by roughly 30% compared to the Raspberry Pi 5, and improved throughput by over 10 $\times$ in the Raspberry Pi $\rightarrow$ GPU Server pair. By contrast, using the GPU Server as the *process-video* client increased mean total latency by 98.7%, revealing that network transmission overhead introduced by high-throughput clients can outweigh their computational advantage.

Fixing the policy at Keep All isolates algorithm behavior from policy effects, letting us compare Random Weighted and Round Robin directly on each machine (Figure 1). Round Robin improved throughput on the GPU Server (14.4 vs. 13.1 IPS) and Nano (7.4 vs. 3.2 IPS) but exhibited extreme latency variance on the GPU (1.5s–50s, n=40). Conversely, edge devices showed high stability: the Nano maintained an IPS standard deviation of just 1.05 (n=543). On the Pi 5, Random Weighted proved superior (5.4 vs. 4.1 IPS). Despite latency swings (43s–141s), the Pi 5 under Random Weighted (n=599) sustained stable throughput (std=1.01), securing roughly 5,800 detections. It is worth mentioning that high latencies are primarily driven by network communication overhead and configurations like the *Keep All* QoS policy, which prioritizes complete frame delivery over response time.

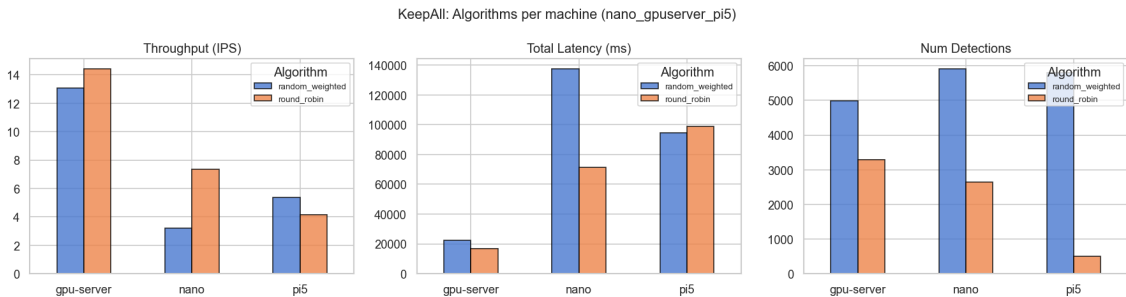


Figure 1. Algorithm comparison (Random Weighted vs. Round Robin)

The difference between algorithms is larger when looking at total detections, shown in the third panel of Figure 1. Random Weighted returned more detections than Round Robin on every machine — roughly 5,000 vs. 3,300 on the GPU Server and 5,900

vs. 2,700 on the Nano — but the gap is widest on the Pi 5, where Random Weighted returned roughly 5,800 detections against only 500 for Round Robin. Round Robin’s even split assigns the Pi 5 the same frame share as the GPU Server, overloading the weaker device and dropping most of its frames before a result returns. Random Weighted’s load-aware assignment shifts fewer frames to the Pi 5, keeping it productive at a modest cost to peak throughput on the stronger machines.

5. Conclusion

The experimental results demonstrate that the efficiency of object detection offloading in edge computing depends not only on hardware capabilities but also on scheduling decisions and device roles. While GPU servers significantly reduced inference latency and increased throughput, assigning them as video sources increased end-to-end latency due to communication overhead. Furthermore, the Random Weighted scheduler proved more suitable for heterogeneous environments by improving load distribution and increasing successful object detections. Future work will investigate adaptive scheduling algorithms driven by runtime telemetry, as well as profiling the actual energy consumption trade-offs of intelligent offloading across these resource-constrained devices, while also analyzing other metrics for offloading policies in the edge continuum by incorporating a broader range of edge devices.

Acknowledgements

H. S. thanks CNPq (grant #302296/2023-9). The authors thank FAPESP grants 2019/26702-8, 2023/00566-6, and 2025/21092-8.

References

- Kashyap, A. and Vidyarthi, A. S. (2022). A load balancing strategy for scheduling of jobs in fog computing environment. In *Proceedings of the 2nd International Conference on Advances in Computing, Communication, Embedded and Secure Systems (ICACITE)*, pages 1152–1157, Greater Noida, India.
- Kaur, K., Walia, G. S., and Gill, N. S. (2021). A systematic study of load balancing approaches in the fog computing environment. In *Proceedings of the 10th International Conference on System Modeling and Advancement in Research Trends (SMART)*, pages 111–115, Moradabad, India.
- Michaelis, O., Schmid, S., and Mostafaei, H. (2024). L3: Latency-aware load balancing in multi-cluster service mesh. In *Proceedings of the 25th International Middleware Conference (Middleware '24)*, pages 49–61, New York, NY, USA. Association for Computing Machinery.
- Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646.
- Zhang, J., Keramat, F., Yu, X., Hernández, D. M., Queralta, J. P., and Westerlund, T. (2022). Distributed robotic systems in the edge-cloud continuum with ros 2: A review on novel architectures and technology readiness. In *2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 1–8. IEEE.
- Zhou, Z., Yu, H., Xu, C., Chang, Z., Mumtaz, S., and Rodriguez, J. (2018). Begin: Big data enabled energy-efficient vehicular edge computing. *IEEE Communications Magazine*, 56(12):82–89.