Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.088224

ARTICLE

A 5G-MEC-Enabled, Digital-Twin-Trained Framework for Autonomous Mobile Robots on the ROSMASTER R2 Platform

Daniel Šolc[1,*], René Ivančák[1], Juraj Gazda[1], Eva Chovancová[1], Eugen Šlapák[1] and Gabriel Bugár[2]

[1]Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Košice, Slovakia

[2]Department of Computer Networks, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Košice, Slovakia

*Corresponding Author: Daniel Šolc. Email: daniel.solc@tuke.sk

Received: 30 June 2026; Accepted: 10 August 2026; Published: 15 September 2026

ABSTRACT: Autonomous mobile robots increasingly rely on three tightly coupled capabilities: low-latency wireless connectivity, edge-side compute acceleration, and simulation-based pre-training of perception and control models. Each has been studied extensively in isolation, but their joint deployment on a single platform remains rare. This paper presents an integrated framework combining a private 5G Stand-Alone (5G SA) access network, a Multi-Access Edge Computing (MEC) layer with adaptive offloading, and a digital-twin training pipeline in NVIDIA Omniverse Isaac Sim. It is realised on the Yahboom ROSMASTER R2 with an NVIDIA Jetson Orin NX, a Quectel RM530N-GL 5G modem in band n78, and a Raspberry Pi 5 load-balancing orchestrator. Perception models are trained entirely in simulation: a one-dimensional residual network with Squeeze-and-Excitation blocks (ResNet-1D-SE) for Light Detection and Ranging (LiDAR) localization, and a Soft Actor-Critic (SAC) agent for vision-guided lane following. At deployment time, a utility-based node selector on the Raspberry Pi 5 selects in real time which heterogeneous MEC node runs the You Only Look Once version 26 (YOLO26) detector from central processing unit (CPU), random-access memory (RAM), graphics processing unit (GPU) and round-trip-time (RTT) telemetry, and the robot streams directly to it. We evaluate end-to-end latency and stability under one-to-four concurrent streams, adaptive MEC switching, onboard-vs.-offloaded energy, and sim-to-real localization accuracy. The private 5G SA link delivers seven times lower per-packet latency jitter than Wi-Fi (2.1 vs. 14.8 ms standard deviation) at the cost of a 5 ms higher single-stream end-to-end latency (67 vs. 62 ms); this overhead is already erased at three concurrent streams and becomes a 14 ms advantage for 5G SA at four. Offloading YOLO to a GPU MEC node cuts the robot's compute-related power draw by 5 W (40% of the compute power budget, propulsion excluded); and the sim-trained localizer transfers to the physical robot with a 7.8 cm mean error and no real-world fine-tuning. The framework provides a reproducible reference design and a quantitative deployment guideline for connected autonomous robots.

KEYWORDS: Autonomous mobile robots; 5G Stand-Alone; multi-access edge computing; digital twin; NVIDIA Isaac Sim; sim-to-real transfer; reinforcement learning; LiDAR localization; YOLO; ROSMASTER R2

1 Introduction

Autonomous mobile robots and small-scale autonomous vehicles have moved from laboratory demonstrators to applied platforms in logistics, surveillance, last-mile delivery, agriculture and education. Their technical foundation rests on three pillars usually developed by separate communities: the communication

stack that connects the vehicle to the infrastructure, the compute stack that runs perception, planning and control, and the learning stack that produces the neural-network models [1,2]. Each pillar is mature in isolation, yet one question receives little attention: how should these three pillars be co-designed and jointly evaluated on a single physical platform so that the resulting system is both deployable in the field and reproducible in the laboratory?

Three enabling technologies make such a system feasible. Fifth-generation networks in their Stand-Alone (5G SA) configuration provide the bandwidth, latency and reliability required for high-rate sensor streaming and tele-operation [3]; Multi-Access Edge Computing (MEC) brings cloud-class compute within a single radio hop, offloading perception that would otherwise saturate the onboard central-processing-unit (CPU) and graphics-processing-unit (GPU) budget [4]; and the digital-twin paradigm, accelerated by photorealistic simulators such as NVIDIA Isaac Sim, lets perception and control networks be trained entirely in simulation and deployed with minimal or zero fine-tuning under careful domain randomization [5,6].

The difficulty lies in bringing them together. On small, educationally accessible platforms (e.g., Yahboom ROSMASTER R2, DuckieTown, F1TENTH), integration has been only partial: each pillar is typically addressed in depth while the other two are treated as fixed boundary conditions. Tele-operation work assumes the perception stack is already in place [7]; MEC offloading studies use canned video rather than full autonomy stacks [8,9]; and sim-to-real papers seldom discuss the wireless backhaul on which the trained model will run in production [10].

This paper closes that gap with an integrated three-layer framework, realised on a single commodity platform—the Yahboom ROSMASTER R2 with an NVIDIA Jetson Orin NX—and evaluated end-to-end in one experimental campaign. The individual components are deliberately off-the-shelf; the contribution is a reproducible reference design and an end-to-end integration study showing how the three planes co-operate on one platform. Two couplings make this concrete: the node selector's round-trip-time (RTT) term ties the edge-compute decision to the live connectivity state, and the energy freed by offloading the detector is what leaves onboard headroom for the sim-trained localizer (Sections 7.2 and 7.3). Concretely:

- A layered reference architecture for connected autonomous mobile robots that separates connectivity, edge computation and digital-twin training into independent but interoperable planes (Section 3).
- A complete realisation on a commodity educational robot: private 5G SA bring-up over an Ericsson Private 5G infrastructure (Quectel RM530N-GL, band n78), OpenVPN-tunnelled MEC connectivity, an Ethernet Control Model (ECM) mode modem integration and a hardware-accelerated GStreamer video pipeline (Sections 4 and 5).
- A digital-twin training pipeline on NVIDIA Isaac Sim using a one-dimensional residual network with Squeeze-and-Excitation blocks (ResNet-1D-SE) for two-dimensional Light Detection and Ranging (LiDAR) localization and a Soft Actor-Critic (SAC) agent for vision-based driving, with LiDAR-specific domain randomization (Section 6).
- A utility-based MEC node selector that chooses, in real time, which heterogeneous edge node runs You Only Look Once version 26 (YOLO26) inference from a normalized weighted average of CPU, random-access memory (RAM), GPU and RTT load; the Raspberry Pi 5 orchestrator only signals the decision while the video flows directly from the robot to the chosen node (Section 5).
- A unified evaluation contrasting onboard, Wi-Fi-MEC and 5G-MEC modes along latency, throughput, jitter, energy and accuracy—including multi-stream experiments with up to four concurrent streams—distilled into deployment guidelines (Sections 7 and 8).

2 Related Work

The proposed system unifies three research streams that have advanced largely in isolation—5G connectivity, MEC offloading and digital-twin training—which we review below, emphasising small, educationally accessible autonomous mobile robots and the seams at which each stream connects to the other two.

2.1 5G and V2X Connectivity for Autonomous Mobility

Fifth-generation networks are a recognised enabler for connected and autonomous vehicles. The Third Generation Partnership Project (3GPP) roadmap distinguishes enhanced mobile broadband (eMBB), ultra-reliable low-latency communication (URLLC) and massive machine-type communication (mMTC) [2,3]; URLLC targets end-to-end latencies below 10 ms at reliability $1–10^{-5}$, underpinning safety-critical vehicle-to-everything (V2X) communication such as cooperative collision avoidance and remote driving. A less-studied, practical aspect is the integration of commercial 5G modules into small mobile robots and the choice of onboard edge computer, which we address for the Quectel RM530N-GL on a private 5G SA network in Section 4. Remote Experience Centre (REC) systems [7] combine digital twins with 5G and show that bandwidth alone is not the relevant metric: coding-scheme changes and jitter degrade fine-motor teleoperation even when average throughput is sufficient, which is why our evaluation measures jitter explicitly (Section 7). More recent work builds a network digital twin of the 5G segment traversed by a mobile robot, predicting the radio quality a robot will encounter along its path [11]; this reinforces the same lesson—that the connectivity state must be treated as a first-class input to the compute and control layers rather than an afterthought.

2.2 Multi-Access Edge Computing for Robotic Perception

The MEC paradigm [4] relocates compute from central data centres to the edge of the radio access network, reducing both round-trip latency and the energy cost of offloading; the reference architecture of the European Telecommunications Standards Institute (ETSI) [12] is now widely adopted, and recent work integrates the MEC data plane directly into an open 5G core to shorten the path between radio and compute [13]. For vision-based perception, full offloading ships raw video to a server that performs the entire inference [9], whereas partial offloading splits the inference graph between device and server, sometimes with early-exit classifiers that terminate locally on easy inputs [14]; multi-hop relaying [8] forwards and partially processes the workload at the cost of a more complex control plane, and more recent schemes learn the offloading decision itself through deep reinforcement learning [15]. All require a node-selection policy aware of server load and network state; for autonomous mobile robots specifically, edge-cloud placement has been framed as a 5G planning problem [16], and its convergence with digital twins for 5G-and-beyond offloading has been surveyed in [17]. A complementary line of work keeps the detector on the device and instead shrinks it to fit: lightweight YOLO variants trade a little accuracy for a footprint that embedded hardware can sustain in real time [18], while dedicated accelerators such as field-programmable gate arrays (FPGAs) push detection throughput up at the edge node itself [19]. These directions are orthogonal to ours—they lower the cost of local inference, whereas offloading removes it from the robot altogether—and the energy and thermal measurements of Sections 7.3 and quantify why, on our platform, the latter is the more effective lever. Our edge-compute layer adopts full offloading of a YOLO26 detector driven by a utility-based node selector whose RTT term ties the offloading decision directly to the connectivity plane (Section 5); the two are therefore evaluated jointly (Section 7).

2.3 Digital Twins and Sim-to-Real Transfer in Robotics

The digital twin—a live, bidirectionally synchronised replica of a physical system—originates in manufacturing [20] and has been embraced by robotics for training, validation and safety analysis [21]. Photorealistic simulators such as NVIDIA Isaac Sim [22] provide the physics, rendering and sensor models needed to train deep-learning policies entirely in simulation. Transfer to reality is not automatic: the reality gap [5] arises because simulators only approximate sensor noise, contact dynamics, illumination and material properties. Domain randomization (DR), which samples simulation parameters randomly during training, is the dominant counter-measure [5,6]: with broad enough training-time variability, reality is just another realisation of the same distribution. Digital-twin-based pipelines pair this idea with a high-fidelity replica of the target workcell, further narrowing the reality gap for tasks such as industrial grasping [23]. More recent work refines the randomization itself rather than widening it blindly: offline domain randomization infers the simulator's parameter distribution from a small set of previously collected real trajectories instead of hand-tuning the ranges [24], and pairing randomization with an explicit domain adaptation stage narrows the residual gap further [25]; a recent review surveys the resulting design space and its open problems [10]. For LiDAR, DR must respect the geometry of the sensor—Gaussian range noise, random ray dropout, occlusion masks and cyclic shifts—the strategies adopted in Section 6. For policy learning, off-policy actor-critic methods, in particular Soft Actor-Critic (SAC) [26], are the de-facto standard for continuous control; SAC's sample efficiency, twin-critic stability, automatic temperature tuning and maximum-entropy exploration motivate its use here for vision-guided lane following.

2.4 Positioning of the Present Work

Compared to this literature, our contribution is a reproducible cross-layer reference design that treats connectivity, edge computation and digital-twin training as one engineering problem rather than three separately optimised subsystems. What is distinctive is how the planes drive one another: the node-selection decision is coupled, through an explicit RTT term, to the live state of the 5G plane, and the digital-twin layer is calibrated against the latency, jitter and energy operating points the other planes actually exhibit on commodity hardware. We study a single robot deliberately: precise per-node characterisation of latency, jitter, energy and sim-to-real accuracy is a prerequisite for—not a competitor to— cooperative multi-robot deployment, whose contention models need exactly these single-node operating points as inputs.

3 System Architecture

This section gives the bird's-eye view of the framework: the common physical platform shared by all three layers (Section 3.1), the three logical planes (Section 3.2) and the end-to-end data and control flow that ties them together (Section 3.3).

3.1 Reference Hardware Platform

The reference platform is the Yahboom ROSMASTER R2, a small (≈30 cm) four-wheeled vehicle with Ackermann steering kinematics that is widely used in academia. Its Ackermann geometry—unlike differential-drive platforms—reproduces the steering behaviour of real automobiles, enabling validation of path-following and dynamic-model algorithms intended for full-size vehicles. A modular sensor suite (2D LiDAR YDLIDAR TG30, red-green-blue (RGB) camera, inertial measurement unit (IMU), depth camera) exposes data via standard ROS 2 topics, and compute is likewise modular: the onboard NVIDIA Jetson Orin NX runs perception and control, while a separate Raspberry Pi 5 in its 16 GB configuration (Arm Cortex-A76 at 2.4 GHz) is dedicated to node-selection decisions only and never relays the video stream.

Because both figures matter for reproducing the results, the two compute modules are specified exactly. The Jetson module is the Orin NX 8 GB variant, which NVIDIA rates at 70 TOPS (sparse INT8) under its 10 to 20 W power profiles and at 117 TOPS under the 40 W "Super" profile introduced with JetPack 6.2 [27]; all experiments in this paper run the JetPack 6.2.1 image, and the onboard full-stack test of Section 7.4 uses the MAXN profile, so the 117 TOPS and 40 W figures are the applicable ones. The 8 GB of shared CPU/GPU memory on this variant is also what makes the memory ceiling of Section 7.3 binding. The orchestrator is the 16 GB model of the Raspberry Pi 5, which has been available alongside the 4 and 8 GB models since 2025 [28]; the orchestrator is a control-plane element only, so the smaller models would also suffice.

While the full autonomy stack is characterised on a single ROSMASTER R2, the testbed contains up to three identically equipped units; because each carries more than one camera, two to three units suffice to generate the one-to-four simultaneous video streams used in Section 7, placing the shared infrastructure under realistic concurrent load.

The infrastructure side comprises the campus 5G SA network of the Technical University of Košice (TUKE)—an Ericsson Private 5G core on band n78—and two heterogeneous MEC servers: Virtual Server 1, a GPU server (Intel Core i7-8700K, NVIDIA RTX A4500 20 GB, 64 GB RAM) and the primary candidate for accelerated convolutional-neural-network (CNN) inference; and Virtual Server 2, a CPU-only virtual machine (VM) (Intel Xeon Gold 5218 B, 16 GB RAM) used as a fallback and to study the cost of CPU-only inference.

3.2 The Three Logical Planes

On top of the physical platform the framework introduces three logical planes (Fig. 1), each detailed below. The connectivity plane (Section 4) exposes two interchangeable bearers (Wi-Fi 5 and private 5G SA) unified by an OpenVPN tunnel that hides the active bearer from the application layer. The edge-compute plane (Section 5) comprises two MEC servers and a Raspberry Pi 5 orchestrator whose utility-based node selector collects per-node load and directs the robot to the lowest-index node, while the video flows directly from robot to node. The training and digital-twin plane (Section 6) is logically offline: on a USD reconstruction of the laboratory in Isaac Sim it trains a ResNet-1D-SE LiDAR localizer and a SAC driving policy, both exported to TensorRT and deployed on the Jetson.

3.3 End-to-End Data and Control Flow

At run-time, the Jetson captures RGB and LiDAR via ROS 2 topics; the GStreamer pipeline hardware-encodes the video (≈5 Mbps per stream) and streams it directly to the MEC node chosen by the Raspberry Pi 5 (over Wi-Fi or the Quectel 5G modem inside the OpenVPN tunnel), where YOLO26 runs and returns detections as JavaScript Object Notation (JSON) messages over the User Datagram Protocol (UDP), fused onboard with the LiDAR localization estimate. The orchestrator is never on the data path, and the digital twin can mirror the robot's state offline to test alternative policies without risking the physical platform.

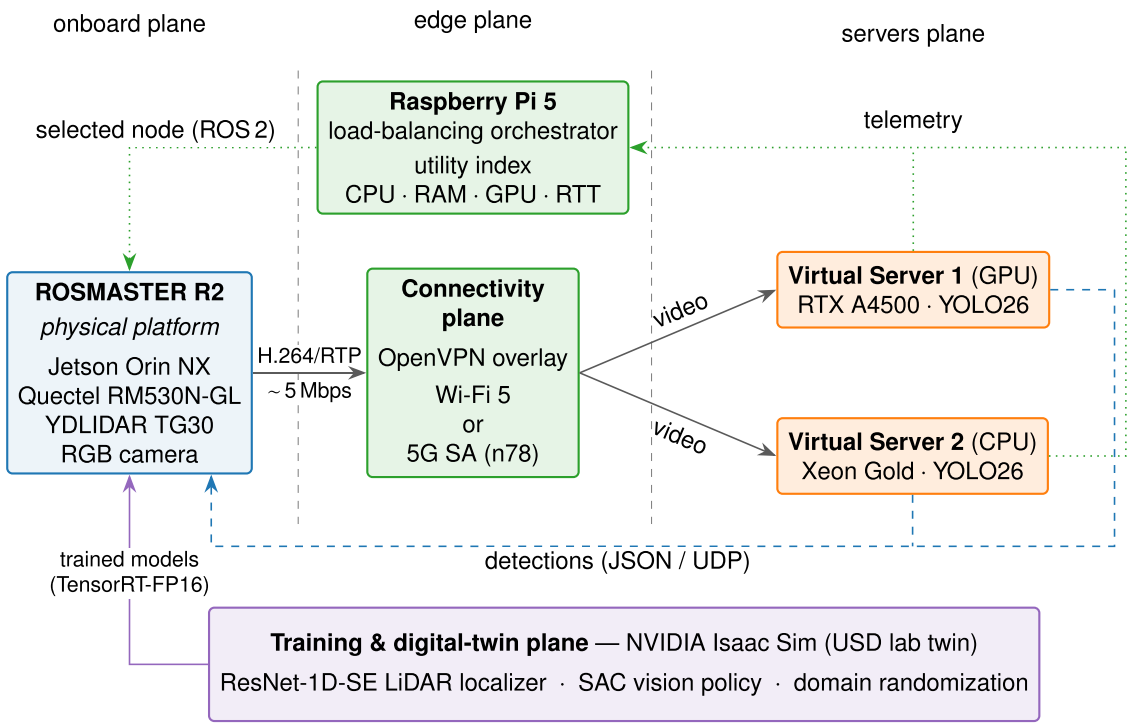

Figure 1: High-level architecture of the integrated 5G-MEC framework. The robot streams hardware-encoded H.264 video through the connectivity plane (Wi-Fi 5 or private 5G SA, unified by an OpenVPN overlay) directly to the MEC node selected by the Raspberry Pi 5 orchestrator. The orchestrator is a control-plane element only: it collects per-node telemetry (CPU, RAM, GPU, RTT) and signals the robot over ROS 2 which node to use, but never carries the video. The selected node runs YOLO26 inference and returns detections over UDP. The digital-twin plane trains the LiDAR localizer and SAC driving policy offline in NVIDIA Isaac Sim and deploys them on the Jetson Orin NX.

4 Connectivity Plane: Private 5G SA Integration

The connectivity plane delivers two services to the rest of the framework: (i) a wireless bearer between the robot and the edge infrastructure, and (ii) a secure, address-stable overlay that hides the choice of bearer from the application layer. This section describes the physical bearer (Section 4.1), the modem integration on the Jetson side (Section 4.2), and the OpenVPN-based overlay (Section 4.3). All design choices are backed by measurements reported in Section 7.

4.1 Private 5G Stand-Alone Infrastructure

The radio infrastructure used in this work is the campus Private 5G SA network of the Technical University of Košice (TUKE), based on the Ericsson Private 5G platform. The network operates in 3GPP band n78 (3.5 GHz time-division duplex), which is the most widely deployed mid-band 5G spectrum in Europe. Two radio access network (RAN) cells cover the autonomous-vehicle laboratory and the surrounding corridor, which makes it possible to study intra-cell as well as inter-cell mobility of the robot.

A Stand-Alone configuration (5G SA, as opposed to non-stand-alone 5G NSA which still relies on a 4G LTE anchor) was chosen for three reasons. First, 5G SA exposes the full URLLC capability of the protocol stack, which is a prerequisite for low-jitter teleoperation. Second, network slicing and quality-of-service differentiation are native features of 5G SA, which allows the operator to reserve a dedicated slice for the

robot fleet. Third, SA-only deployment isolates the experimental traffic from the legacy 4G network and simplifies the analysis of measurement results.

4.2 Onboard 5G Modem Integration

Connecting the robot to the radio infrastructure described above required a chain of decisions that build on one another, beginning with the modem itself and ending with the low-level tuning that finally made the link stable. The first of these was the choice of module. We evaluated the Quectel RM500Q-GL and the Quectel RM530N-GL modules against the requirements of an autonomous mobile robot: worldwide band support, downlink throughput, power draw and availability of Linux drivers. Table 1 summarises the comparison. The RM530N-GL was selected because of its higher 5G NR SA downlink (2.4 Gbps vs. 2.1 Gbps), wider band support (n26, n70, n75, n76 added) and the availability of Linux drivers up to kernel 6.7. The modest extra power consumption (approximately 300 mW above the RM500Q-GL) was accepted as a deliberate trade-off.

Table 1: Comparison of the two candidate 5G NR SA modules.

Parameter	Quectel RM500Q-GL	Quectel RM530N-GL
Coverage	Worldwide except Americas	Worldwide
NR SA downlink	2.1 Gbps	2.4 Gbps
NR SA uplink	900 Mbps	900 Mbps
USB 3.0 idle	54 mA	69 mA
Bands (selected)	n1, n3, n7, n28, n77, n78	+n26, n70, n75, n76

Once the module was chosen, it had to be physically integrated into the robot. The selected module is hosted on the Waveshare USB 3.2 Gen1 5G DONGLE expansion board, fed by four external antennas and connected to the Jetson Orin NX via a USB 3.0 type-A port. The expansion board provides an auxiliary 5 to 12 V jack that is wired to the robot battery so that the modem does not draw its current through the Jetson's power-management integrated circuit (PMIC).

With the hardware in place, the next question was how the modem should present its data interface to the operating system. A 5G modem can expose it in several modes—the Ethernet Control Model (ECM), the Remote Network Driver Interface Specification (RNDIS), the Mobile Broadband Interface Model (MBIM) and the Qualcomm MSM Interface (QMI)—and ECM was selected for two reasons. First, in Linux Ubuntu 22.04 (the version shipped with the JetPack 6.2.1 SDK used in this work) ECM appears as a regular network interface with the prefix `enx`, which makes it transparent to ROS 2 and to the OpenVPN client. Second, the modem vendor explicitly recommends ECM for ROS-based payloads through their support channel.

Selecting ECM is only the first step of bringing up the link; that choice, together with the rest of the radio behaviour, is then committed to the modem through its AT port using the `minicom` terminal program. The configuration sequence used in this work is shown in Listing 1. The most important parameters are the network preferred mode (NR5G), the n78 band restriction, the QMAP multiplex/passthrough rule (which keeps the modem in non-pass-through mode so that the LAN range `192.168.224.0/22` provided by the 5G core is forwarded to the Jetson), and the explicit enable of IPv4 DHCP. The settings are saved to non-volatile memory with `AT&W` so that the modem boots into the correct state.

Listing 1: Key AT commands used to bring up the 5G SA link on the Quectel RM530N-GL.

```
AT+QCFG="usbnet",1                    # ECM mode
AT+CGDCONT=1,"IP","data"              # PDP context, IPv4 APN "data"
AT+QNWPREFCFG="mode_pref",NR5G        # 5G-only network search
AT+QNWPREFCFG="nr5g_band",78          # restrict to band n78
AT+QMAP="MPDN_rule",0,1,0,0,1         # rule 0, profile 1, no IPPT
AT+QMAP="auto_connect",0,1,1          # auto-connect rule 0
AT+QMAP="DHCPV4DNS","enable"          # enable IPv4 DHCP and DNS
AT+CGATT=1                            # attach to packet services
AT&W                                  # persist settings
```

Each command's effect is annotated inline in Listing 1; two groups deserve emphasis. The `QNWPREFCFG` pair pins the modem to 5G-only operation on band n78, which prevents the silent fallback to LTE that would otherwise defeat the Stand-Alone measurements of Section 7. The `QMAP` rules keep the modem out of IP-passthrough mode, so that the `192.168.224.0/22` subnet handed out by the 5G core is routed to—and terminated on—the Jetson rather than hidden behind the modem; this is what makes the ECM interface behave as an ordinary Linux network device for ROS 2 and OpenVPN. The closing `AT&W` writes the whole configuration to non-volatile memory so that the modem returns to this state after a power cycle.

Even with the modem attached and configured, the link did not yet behave reliably, and the final piece of the integration proved the most subtle one: the interaction between IP fragmentation and the OpenVPN overlay. By default the 5G network interface is assigned a maximum transmission unit (MTU) of 1500 bytes, but the private 5G SA bearer does not sustain frames that large. Throughput testing with `iperf3` showed that 1460 B is the largest MTU at which the link transfers data reliably, and that raising it to even 1461 bytes halts the transfer altogether. We therefore lower the MTU to 1460 B on the physical 5G interface and set the OpenVPN `tun0` interface to the same value, so that tunnelled traffic never exceeds the bearer's limit. Listing 2 shows the commands used to configure the MTU on both interfaces.

Listing 2: Setting the MTU on the 5G interface and on the OpenVPN tunnel interface.

```
sudo ip link set enx824421974bb9 mtu 1460   # 5G physical interface
sudo ip link set tun0            mtu 1460 # OpenVPN tunnel
```

The two commands in Listing 2 must be applied together: the first line lowers the MTU of the physical 5G interface (whose name `enx824421974bb9` is the ECM device from Section 4.2), and the second applies the same limit to the OpenVPN `tun0` tunnel that rides on top of it. If only the physical interface were adjusted, OpenVPN would continue to emit 1500 B frames on `tun0` that are then fragmented and dropped by the bearer—precisely the stall observed at 1461 B—so matching both values is what keeps tunnelled traffic within the bearer's usable frame size.

4.3 OpenVPN Overlay

The OpenVPN overlay turns the heterogeneous physical bearers (Wi-Fi inside the campus, public Internet from elsewhere, private 5G SA) into a single logical network with the address range `10.20.30.0/24`. This is necessary for two reasons. First, the MEC servers and the robot belong to different virtual local area networks (VLANs); an overlay network is the only way to give them a stable end-to-end address. Second,

the overlay terminates inside the campus on a fixed virtual private network (VPN) server, which decouples the robot's reachability from the choice of bearer. This matters because of how the private 5G SA bearer assigns addresses: on the 5G path the robot receives a dynamically assigned private IPv4 address out of the `192.168.224.0/22` pool of the private core (Section 4.2). That pool is terminated behind the core's User Plane Function (UPF) and is not routed into the campus VLANs that host the MEC servers, so the robot has no stable, inbound-reachable address on the 5G bearer, and its address changes when the bearer changes. The overlay supplies exactly what the bearer does not: one fixed address that is reachable in both directions regardless of the path in use. No carrier-grade address translation is involved—the network is a private, campus-operated 5G SA deployment—and the constraint is one of private addressing, routing and inbound accessibility rather than of operator policy.

The overlay is built on a self-signed 2048-bit RSA root certificate authority (CA) generated for this testbed and valid until 2028. Each client (the robot, the MEC servers, the orchestrator) holds its own certificate and key pair. The server listens on UDP port 1194; LZO compression and a `keepalive 10 120` watchdog are enabled to reduce overhead and to recover automatically from short link outages. The OpenVPN client on the Jetson is configured as a systemd service so that the tunnel is restored automatically after a reboot.

The overall result is that, from the perspective of the application layer (ROS 2 nodes, GStreamer pipelines, the MEC node selector), the robot has a single fixed IP address regardless of whether it is connected via Wi-Fi or via the private 5G SA network. The choice of bearer becomes a network-layer concern that can be evaluated and switched independently, which is exactly the property exploited in the experimental evaluation of Section 7.

5 Edge-Compute Plane: Adaptive MEC Offloading

The edge-compute plane allows the robot to delegate computationally heavy perception workloads to a nearby MEC infrastructure. A decision block is needed because a MEC node is not an infinite resource: each server saturates beyond a handful of concurrent video streams (Section 7.2 shows the GPU node reaching 82% utilisation at four streams), and once a node is pushed past this concurrent-stream ceiling its inference latency climbs back toward—or beyond—the unacceptable onboard figures of Section 7.4. With several robots streaming at once, the task of this block is to select, for each stream, the MEC node that will run its detector, so as to distribute the streams across the available nodes and keep none of them past its ceiling. We therefore call it a node selector rather than a scheduler: it does not time-multiplex tasks on a shared processor but repeatedly re-evaluates which node should host each stream from live telemetry.

Two design choices distinguish our implementation from the standard "ship-everything-to-the-cloud" approach. First, the node selector runs on a dedicated Raspberry Pi 5 orchestrator rather than on the robot itself, which keeps the Jetson's CPU/GPU entirely available for safety-critical local tasks. Second, the selector is utility-based: it combines four heterogeneous metrics into a single, normalised score that ranks all candidate MEC nodes.

This section describes the data pipeline (Section 5.1), the detection workload (Section 5.2), the node-selection algorithm (Section 5.3) and the integration with the ROS 2 control loop on the robot (Section 5.4).

5.1 Data Pipeline

A schematic of the data pipeline is shown in Fig. 2. RGB frames are captured by the camera connected to the Jetson Orin NX, encoded in H.264 on the hardware encoder using the GStreamer pipeline shown in Listing 3, and forwarded over UDP and the Real-time Transport Protocol (RTP) through the OpenVPN

overlay directly to the MEC node currently selected by the Raspberry Pi 5 orchestrator. The orchestrator itself never receives the video: it only decides which node should be the destination (Section 5.3) and updates the `host` field of the robot's `udpsink` accordingly—in Listing 3 this is the address of the currently selected MEC node.

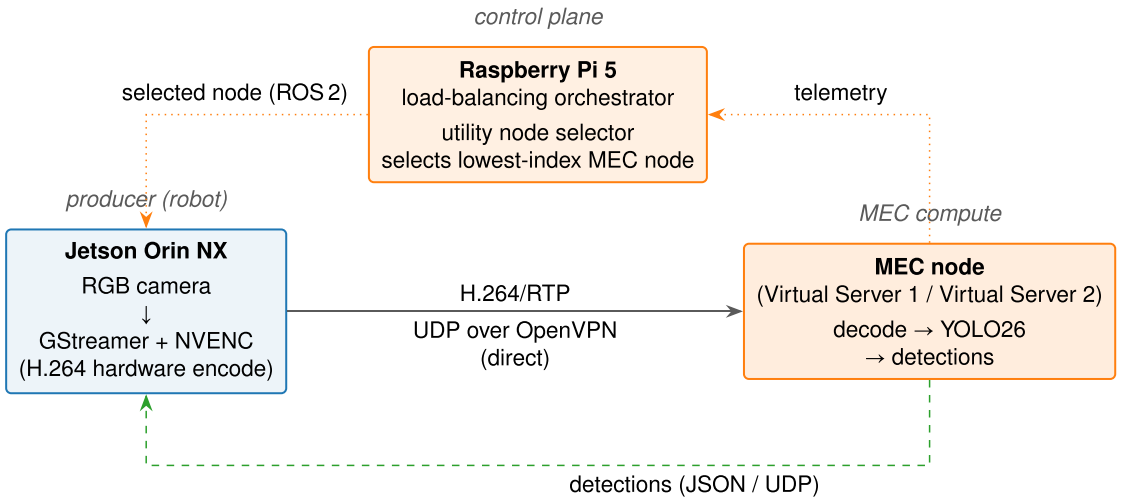

Figure 2: Data and control pipeline of the MEC plane. The JetsonOrin NX hardware-encodes the RGB camera into an H.264/RTP stream that is sent over the OpenVPN overlay directly to the MEC node currently selected by the Raspberry Pi 5 orchestrator. The orchestrator stays on the control plane only: it collects per-node telemetry and signals the robot, over ROS 2, which node to stream to, but never relays the video. The selected node decodes the stream, runs YOLO26 inference and returns detections in JSON over UDP back to the robot.

Listing 3 reads as a left-to-right chain of GStreamer elements separated by the `!` operator: camera capture (`v4l2src`), MJPEG decode, a single-buffer leaky `queue`, NVMM format conversion (`nvvidconv`), hardware H.264 encode (`nvv4l2h264enc`), RTP payloading (`rtph264pay`) and UDP egress (`udpsink`) to the currently selected MEC node. Several of these choices are load-bearing rather than incidental. First, the `nvv4l2h264enc` element delegates the most expensive compression operations to the Jetson's dedicated NVIDIA hardware video encoder (NVENC) unit, freeing the GPU for other workloads. Second, the option `aggregate-mode=zero-latency` disables RTP packet aggregation that would otherwise introduce buffering at the producer side. Third, the `queue` element with `leaky = downstream` discards old frames if the network is temporarily congested, which prevents a cumulative latency build-up. Finally, the RTP payloader's `mtu` is set to 1400 B, below the 1460 B path MTU of the bearer (Section 4.2), so that the IP, UDP and RTP headers fit within a single packet and no fragmentation occurs along the tunnel. We measure the internal pipeline latency in Section 7 and find that it stabilises around 11.5 ms end-to-end through the GStreamer chain.

Listing 3: GStreamer pipeline used on the Jetson Orin NX to capture, encode and stream the RGB camera at 640 × 480 resolution, 30 fps, 5 Mbps.

```
gst-launch-1.0 v4l2src device=/dev/video0 \
   ! image/jpeg,width=640,height=480,framerate=30/1 \
   ! jpegdec \
   ! queue max-size-buffers=1 leaky=downstream \
   ! nvvidconv \
```

```
! 'video/x-raw(memory:NVMM),format=NV12,framerate=30/1' \
! nvv4l2h264enc insert-sps-pps=1 control-rate=1 \
                bitrate=5000000 iframeinterval=0 \
                idrinterval=5 preset-level=1 \
                maxperf-enable=1 \
! h264parse config-interval=-1 \
! rtph264pay pt=96 config-interval=-1 mtu=1400 \
             aggregate-mode=zero-latency \
! udpsink host=10.20.30.10 port=5000 sync=false async=false
```

5.2 Detection Workload

The detection workload is based on the YOLO26-M [29] "signs" model trained for the autonomous-vehicle laboratory of TUKE. We emphasise that object-detection accuracy is not a contribution of this paper: YOLO26-M is employed as a representative, interchangeable off-the-shelf detector whose purpose is to generate a realistic perception workload for the offloading pipeline, and any comparable detector could be substituted without affecting the framework.

The detection task is a closed-set problem confined to a single, well-controlled laboratory environment, for which strong in-domain accuracy is expected. The training dataset, prepared with Roboflow [30], contains 10,847 labelled images and 12 classes covering directional signage (right, ahead), regulatory and warning elements (stop, horn, cross_sign, 5 km, red, people) and dynamic objects (car, human), partitioned into disjoint training and held-out validation splits.

Rather than training from scratch, we adopt standard transfer learning: starting from the publicly available pretrained YOLO26-M backbone, we fine-tune it on the laboratory traffic-sign dataset using the `ultralytics` Python framework (input size 640×640, batch size 32, single-GPU training on the Virtual Server 1 MEC node) with early stopping (patience 3), which terminated the run once the held-out validation metrics had plateaued. On the held-out split the model attains a mean average precision $\mathrm{mAP}_{50} \approx 0.99$ with $\mathrm{mAP}_{50\text{-}95}$ still increasing at the stopping point.

Such rapid convergence to high in-domain accuracy is the expected behaviour of fine-tuning a strong pretrained detector on a closed-set task confined to a single controlled environment, and should not be read as a claim of detection novelty.

The trained weights (`best.pt`) are exported as PyTorch state dictionaries and distributed to both MEC servers. On the GPU server (Virtual Server 1) the model runs in FP16 mode on the RTX A4500; on the CPU-only Virtual Server 2 it runs in FP32 mode using `torch.compile` for partial graph fusion. The MEC server reads the H.264/RTP stream through OpenCV with the GStreamer backend enabled, runs inference once per frame and emits JSON results (bounding boxes, classes, confidences and source-camera IP) back to the robot over UDP.

5.3 Utility-Based Node Selector

The core of the edge-compute plane is the node selector running on the Raspberry Pi 5 orchestrator. Every MEC server periodically (every 2 s in our deployment) emits a UDP telemetry packet that carries four normalised load metrics:

- $C \in [0,1]$—CPU utilisation,
- $R \in [0,1]$—RAM utilisation,

- $G \in [0,1]$—GPU 3D-engine utilisation (or zero if the node has no GPU),
- $L \in \mathbb{R}_{\geq 0}$—round-trip time measured from the Raspberry Pi to the server in milliseconds.

For each metric $m \in \{C, R, G, L\}$ the node selector computes a pressure term that grows hyperbolically as the metric approaches saturation:

$$v_m = \frac{1}{1.01 - \tilde{m}}, \tag{1}$$

where $\tilde{m} \in [0,1]$ is the metric normalised to the unit interval. The CPU, RAM and GPU metrics are already fractions in $[0,1]$ and are used directly ($\tilde{C} = C$, $\tilde{R} = R$, $\tilde{G} = G$); the RTT is normalised as

$$\tilde{L} = \min\left(L/L_{\text{ref}}, 1\right), \qquad L_{\text{ref}} = 100\,\text{ms}, \tag{2}$$

which is the definition later used in Algorithm 1 and in the utility index (4). Because every normalised metric satisfies $\tilde{m} \in [0,1]$, the denominator $1.01 - \tilde{m}$ is bounded below by 0.01—attained only in the limit $\tilde{m} \to 1$—so the pressure is always positive and finite and no explicit flooring of the denominator is required. The reference $L_{\text{ref}} = 100$ ms is a conservative upper bound on the round-trip time the closed-loop controller can tolerate: the complete networked path stays near or below this figure (Section 7.1), so an RTT approaching L_{ref} signals a link too degraded for the node to remain preferred. In normal operation the LAN/5G RTTs to both MEC nodes are a few milliseconds, so $\tilde{L}$ stays well below one and only a pathological link drives the RTT pressure toward saturation; the $\min(\cdot, 1)$ clamp caps $\tilde{L}$ so that a single very large RTT cannot dominate the index. The four pressures are then converted into weights by normalising their sum:

$$t_v = v_C + v_R + v_G + v_L, \quad w_m = \frac{v_m}{t_v} \quad \forall m \in \{C, R, G, L\}, \tag{3}$$

which guarantees $\sum_m w_m = 1$. The final utility index of a server is the weighted average of its (normalised) metrics:

$$\text{idx} = w_C\, C + w_R\, R + w_G\, G + w_L\, \tilde{L}. \tag{4}$$

The intuition behind (1)–(4) is that the weight of a metric grows much faster than the metric itself once the resource approaches saturation: a server with a CPU at 0.95 contributes far more to its own index than a server with a CPU at 0.5, even when the RTT is identical. This is exactly the behaviour we want from a node selector that needs to avoid any bottleneck rather than minimising a single resource. The pseudocode of the node selector is summarised in Algorithm 1. One detail of the decision rule is worth emphasising: the selector does not blindly follow the instantaneous arg min every cycle but applies a strict-improvement guard—it retargets the stream only when some node is strictly better (lower index) than the node currently serving it. Because the current target is retained on an exact or near-tie, this guard suppresses the rapid back-and-forth switching (thrashing) that a plain per-cycle arg min would exhibit when two nodes report almost equal indices, while still migrating the stream promptly once a genuinely less-loaded node appears.

In the deployed system, the orchestrator periodically reconfigures the GStreamer `udpsink` on the robot via a ROS 2 control message; the hand-over takes one telemetry cycle (2 s) on average and does not interrupt the video stream because OpenVPN preserves connectivity throughout. In Section 7 we report a representative trace in which the node selector is exposed to three synthetic background workloads and a final cool-down phase.

5.4 Integration with the Robot

The integration with the robot is intentionally minimal. From the ROS 2 graph point of view, the MEC plane appears as a single node `/mec_perception` that publishes detection messages on the topic `/perception/detections`. The robot's behaviour layer is unaware of whether the inference was performed on Virtual Server 1, on the Virtual Server 2 or, in a fall-back configuration, on the Jetson itself; only the per-detection latency reported in the message header differs. This makes it possible to perform A/B experiments simply by masking the MEC nodes one at a time and to combine the MEC-side detections with the onboard LiDAR localization estimates produced by the digital-twin pipeline described in the next section.

Algorithm 1: Utility-based MEC node selector running on the Raspberry Pi 5 orchestrator

Inputs: set of MEC nodes $\mathcal{N}$, telemetry period T, RTT reference L_{ref}
Initialise current target $n^{\star} \leftarrow$ any node in $\mathcal{N}$
loop
  for each node $n \in \mathcal{N}$ **do**
    Receive (C_n, R_n, G_n, L_n) via UDP
    $\tilde{L}_n \leftarrow \min(L_n / L_{\text{ref}}, 1)$
    Compute $v_{C_n}, v_{R_n}, v_{G_n}, v_{L_n}$ from (1)
    Compute weights $w_{\cdot,n}$ from (3)
    Compute idx_n from (4)
  end for
  $n^{\star\star} \leftarrow \arg\min_n \text{idx}_n$
  if $\text{idx}_{n^{\star\star}} < \text{idx}_{n^{\star}}$ **then** ▷ strict-improvement guard: retarget only if some node is strictly better than the current one
    Signal the robot over ROS 2 to retarget its GStreamer `udpsink` to $n^{\star\star}$
    $n^{\star} \leftarrow n^{\star\star}$
  end if
  Wait T
end loop

6 Digital-Twin Plane: Isaac Sim Pipeline and Sim-to-Real Transfer

The third plane of the proposed framework is the digital-twin plane. Its role is to generate, train and validate the perception and control models that are deployed on the robot, without requiring real-world data collection at scale. In this work it produces two models that complement the MEC-side YOLO26 detector: a LiDAR localizer that runs locally on the Jetson Orin NX, and an SAC vision-based driving policy used as a fall-back when MEC connectivity is degraded. This section describes the simulation environment (Section 6.1), the LiDAR localization pipeline (Section 6.2) and the SAC driving pipeline (Section 6.3).

6.1 Simulation Environment

The simulation environment is built on NVIDIA Isaac Sim, a physics- and rendering-accurate robotics simulator that uses Universal Scene Description (USD) [31] as its native scene format. USD's layering and referencing mechanism turns out to be very well suited for digital twins: the geometric reconstruction of the laboratory can be kept in one immutable base layer, while experimental variants (different obstacles, different lighting, additional reference landmarks) are applied as non-destructive overlays. The scene used throughout this work consists of three layers:

1. A geometric layer with the walls, doors and structural landmarks of the laboratory, reconstructed from LiDAR-based scanning of the physical space and simplified for efficient collision detection.
2. A semantic layer containing the painted "street" that the robot drives on, with the lane markings, intersections and traffic signs that match the YOLO26 training classes introduced in Section 5.2.
3. A randomization layer that injects domain randomization (lighting, materials, object placement) at training time.

A USD model of the ROSMASTER R2 itself is added to the stage with the correct Ackermann steering joints, mass, inertia and wheel radii calibrated to the physical platform. Two simulated sensors are attached to it: a 2D LiDAR configured to match the parameters of the real YDLIDAR TG30 (8 Hz sweep rate, 720 samples per scan, field-of-view 360°), and an RGB camera that mirrors the intrinsic parameters of the camera mounted on the physical robot. Ranges are saturated at 5.0 m—the extent of the navigable area, beyond which returns carry no information for localization—and returns closer than 0.30 m are discarded as reflections from the robot's own body. The scan that reaches the localizer is therefore a 720-element vector of ranges normalized by the 5.0 m saturation distance to the interval $[0, 1]$.

6.2 LiDAR Localization Pipeline

The first model produced by the digital-twin plane is a LiDAR localizer. Describing it means following it through four stages that build on one another: how the localization problem is posed, the network that solves it, the regime under which it is trained, and the way it is finally deployed onboard the robot. The starting point is the formulation of the task itself. The localization task is formulated as a mapping from a single 2D LiDAR scan $\mathbf{x} \in \mathbb{R}^{720}$ (after Cartesian pre-processing into polar range readings) to a 2D pose $(\hat{x}, \hat{y}) \in \Omega \subset \mathbb{R}^2$, where Ω covers the navigable area of the laboratory. We replace the naive regression formulation with a hybrid classification–regression formulation: the navigable area is discretised into a uniform $G \times G$ grid (with $G = 40$ in the final configuration, corresponding to roughly 12 cm cells), the network predicts a probability distribution over the cells, and a Soft-Argmax operator [32]

$$\hat{\mathbf{p}} = \sum_{i=1}^{G^2} \sigma(\mathbf{z})_i \, \mathbf{c}_i \tag{5}$$

recovers the continuous pose as a weighted sum over cell centres. This formulation provides a stable cross-entropy training signal together with sub-cell metric accuracy.

The way the problem is posed in turn dictates the network that has to produce this cell distribution. The network is a 1D residual CNN [33] augmented with Squeeze-and-Excitation (SE) attention blocks. Table 2 summarises the architecture. Each residual block contains two 1D convolutions, batch normalisation, ReLU activations and a skip connection; an SE module [34] sits between the second convolution and the residual sum, recalibrating the channel-wise activations. The choice of 1D rather than 2D convolutions is dictated by the structure of the LiDAR scan: adjacent samples correspond to adjacent angles, so 1D weight sharing exploits the natural translational invariance of the signal while keeping the parameter count manageable. The final classification head outputs logits over the G^2 cells; an auxiliary regression head outputs $(\hat{x}, \hat{y})$ for direct supervision of the Soft-Argmax output.

With the architecture fixed, the remaining question is how to train it so that it survives the transition to the real sensor. The network is trained entirely on synthetic data generated in Isaac Sim. For each training sample, the simulator is reset to a random pose inside the navigable area, the LiDAR is sampled once, and the ground-truth pose is recorded. To bridge the sim-to-real gap, the following domain-randomization techniques are applied during data generation and as on-the-fly augmentation:

- Additive Gaussian noise $\mathcal{N}(0, \sigma^2)$ with $\sigma = 0.007$. The perturbation is applied to the normalized range vector of Section 6.1—ranges divided by the 5.0 m saturation distance—and the result is clipped back to $[0,1]$, so σ is dimensionless and corresponds to $0.007 \times 5.0\,\text{m} = 35\,\text{mm}$ in metric terms. This is the right order of magnitude for the sensor: the TG30 datasheet publishes no noise distribution, but it specifies a mean range error of $\leq \pm 60\,\text{mm}$ over the 0.05 to 5 m band that covers the whole workspace [35], so the injected noise sits inside the manufacturer's accuracy envelope without saturating it.
- Random ray dropout: each ray is independently set to its maximum value with probability 1.5%, simulating non-reflective surfaces.
- Random occlusion: contiguous segments of 5−50 rays are zeroed, simulating dynamic obstacles such as people standing in the field of view.
- Cyclic shift: the entire range vector is shifted by a random offset, enforcing rotational equivariance.

Table 2: ResNet-1D-SE architecture used for LiDAR localization.

Stage	Operation	Output Shape
Input	720-dim range vector	720 × 1
Stem	Conv1D-7, stride 2, BN	360 × 32
Stage 1	2 × ResBlock-SE	360 × 64
Stage 2	2 × ResBlock-SE	180 × 128
Stage 3	2 × ResBlock-SE	90 × 256
Stage 4	2 × ResBlock-SE	45 × 256
Pool	Global Average Pool	256
Head (cls)	Linear → G^2	1600
Head (reg)	Linear → 2	2

The loss function is a weighted sum of cross-entropy on the cell classification head and mean-squared-error on the Soft-Argmax-derived pose. The model is trained with AdamW [36] for 200 epochs on a single RTX A4500 hosted on Virtual Server 1 and converges in approximately 2 h.

Once trained, the model still has to earn its place on the robot. After training, the network is converted to TensorRT-FP16 [37] and deployed on the Jetson Orin NX. The localizer subscribes to the `/scan` ROS 2 topic and publishes pose estimates on `/loc/pose` at the LiDAR's native 8 Hz rate. We measure its end-to-end inference latency in Section 7.

6.3 Vision-Based Driving via Soft Actor-Critic

The second sim-trained model is a vision-based driving policy. It is designed as a redundant fall-back that takes over when the MEC link is degraded (for example because the robot has left the radio coverage area). The policy takes a 480 × 480 greyscale view of the front RGB camera as its visual input and outputs a continuous two-dimensional action (throttle and steering).

The mapping from this visual input to the action is learned with Soft Actor-Critic (SAC) [26]. SAC's maximum-entropy objective

$$\pi^{\star} = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t} \gamma^{t}\left(r(s_t, a_t) + \alpha\,\mathcal{H}(\pi(\cdot \mid s_t))\right)\right] \tag{6}$$

balances reward maximisation with policy entropy and therefore encourages exploration without the need for external action noise. The temperature α is tuned automatically following the standard Stable-Baselines3 [38] implementation.

What the agent actually optimises is fixed by its state, action and reward. The observation is a dictionary with two entries: the 480 × 480 greyscale camera image and a three-dimensional telemetry vector $\mathbf{t} = (x, y, \psi)$ giving the robot's metric world-space position and yaw, which supplies the agent with explicit spatial context alongside the visual input. The action is a two-dimensional vector in $[-1, 1]^2$ mapping to throttle and steering, which matches SAC's design for continuous control. The reward is the dense signal

$$r = (1 - d_{\min}) + 0.5\,|\text{throttle}|, \tag{7}$$

where $d_{\min}$ is the Euclidean distance from the robot to the nearest reference waypoint: the first term rewards staying on the reference path, while the second rewards forward motion so that the agent does not learn to idle near a waypoint. An episode terminates when $d_{\min} > 1.0$ m, which acts as an implicit penalty by forfeiting all remaining reward in the episode.

With this objective in place, the agent is trained for 1×10^5 environment steps using a single Isaac Sim instance accelerated by the same RTX A4500. Domain randomization at training time covers lighting, materials and non-ideal motion effects. The trained actor network is exported to TensorRT-FP16, packaged into a ROS 2 node and deployed onboard. The policy runs at the camera's native 30 Hz rate on the Jetson.

This deployment does not, however, make the policy the centrepiece of the system. It should be emphasised that the SAC policy is not the primary decision-maker of the production robot: at full system speed, the fused output of the MEC-side YOLO26 detector and the onboard LiDAR localizer is used for behaviour planning. The SAC policy provides a self-contained, network-independent fall-back that can keep the robot inside its lane even when both MEC nodes are unreachable—an important safeguard for an autonomous system that relies on external infrastructure for the bulk of its perception.

7 Experimental Evaluation

This section reports the experimental evaluation of the integrated framework. The goal is not to optimise any single component, but to quantify how the three planes interact under realistic deployment conditions. We organise the experiments along five axes: Section 7.1 characterises the connectivity plane; Section 7.2 analyses the MEC node selector; Section 7.3 reports energy and memory measurements on the robot; Section 7.4 tests whether the entire stack could instead run onboard; and Section 7.5 evaluates the sim-to-real transfer of the LiDAR localizer. Section 7.6 summarises the trade-offs between the three deployment modes that the framework supports (on-board only, Wi-Fi + MEC, 5G + MEC).

All measurements were obtained in the autonomous-vehicle laboratory of the Faculty of Electrical Engineering and Informatics, TUKE. The radio environment, the MEC servers, the software versions and the operating system (Ubuntu 22.04.5 LTS, JetPack 6.2.1 + b38) are kept identical across experiments to ensure reproducibility. The full autonomy stack (LiDAR localization and vision-based driving) is characterised on a single robot, while the connectivity and MEC planes are exercised with up to three ROSMASTER R2 units streaming concurrently, so that the node selector and the radio link are evaluated under genuine multi-robot load.

7.1 Connectivity Plane: Latency, Throughput and Stability

We first measure the latency introduced by the local video pipeline, independently of the network. Using the `gstshark` profiling tool, we record the time spent in every GStreamer element between the `v4l2src`

capture point and the `udpsink` send point under nominal load (640 × 480 resolution, 30 fps, 5 Mbps). The overall pipeline latency stabilises at approximately 11.5 ms end-to-end. The dominant contributors are the JPEG decoder (which absorbs the bulk of the early processing time) and the H.264 hardware encoder (approximately 5 ms). All other elements (`capsfilter`, `queue`, `h264parse`) contribute negligible latency, confirming that the buffer settings are correctly tuned.

We then measure the end-to-end (E2E) latency from the moment a frame is captured on the Jetson to the moment the corresponding detection JSON arrives back at the robot, with one to four parallel video streams. The parallel streams are not synthetic: they are produced by physical hardware, using two to three ROSMASTER R2 units—each carrying more than one camera—to generate the one-to-four concurrent streams. The measurements therefore reflect genuine multi-robot load on the shared MEC and radio resources rather than replayed traffic. Two bearers are compared: Wi-Fi 5 inside the campus, and the private 5G SA network. Bitrate is varied through the `bitrate` parameter of `nvv4l2h264enc`.

Table 3 reports the main result. Three observations stand out. First, with a single stream Wi-Fi has a small advantage over 5G (62 vs. 67 ms), which follows from the fact that Wi-Fi access points sit one switch away from the MEC servers, while the 5G core adds two additional hops (gNodeB and UPF). Second, the Wi-Fi latency grows much faster with the number of streams, reaching 93 ms at four streams when the aggregated bitrate exceeds 90 Mbit/s; the 5G latency grows almost linearly and stops at 79 ms at four streams. Third, the Quectel RM530N-GL has a practical sustained-throughput envelope of approximately 60 Mbit/s: when we deliberately stress-tested the link by forcing four streams up to 90 or 120 Mbit/s, the modem could no longer hold the session. This bound lies well above the 20 Mbit/s aggregate of the target four-stream configuration (4 × 5 Mbit/s), so it delimits the stress envelope rather than constraining nominal operation; it nonetheless motivates the adaptive-bitrate controller outlined in Section 9.

Table 3: End-to-end latency in milliseconds for one to four parallel video streams on Wi-Fi 5 vs. private 5G SA, including all on-server processing (YOLO inference on Virtual Server 1 and JSON return through the OpenVPN tunnel). Each entry is averaged over 10 runs of 60 s each, for which per-run variation was small.

Streams	1	2	3	4
Wi-Fi (E2E, ms)	62	71	82	93
5G SA (E2E, ms)	67	71	75	79

A single average latency value tells only half of the story. We measure the jitter—here the standard deviation of the per-packet latency—of a sustained UDP stream over each bearer. The probe was repeated 20 times per bearer (20 runs of 60 s each); the per-run statistics were mutually consistent, and Table 4 reports the values averaged across the 20 runs. The table shows that 5G has a slightly higher average latency but a markedly lower spread; this is consistent with the fact that Wi-Fi shares the medium with all the other devices in the laboratory, while the private 5G SA slice is dedicated. From the perspective of real-time control, low jitter is more valuable than low average latency, because it determines the worst-case end-to-end delay that the control loop must tolerate.

Tables 3 and 4 measure two different quantities and should not be compared directly. Table 4 isolates the network contribution with a bare UDP probe, and there the 5G SA penalty is 1.8 ms of mean latency (11.4 vs. 9.6 ms). Table 3 measures the complete application path—capture, encode, transport, server-side YOLO26 inference and the JSON return—where the single-stream penalty is the 5 ms gap between 62 and 67 ms. The larger end-to-end figure reflects the extra hops and the queueing they add ahead of and behind inference, not a second measurement of the same delay. Both penalties apply only at a single stream: from three streams

onward the ordering reverses and 5G SA is the faster bearer (75 vs. 82 ms at three streams, 79 vs. 93 ms at four).

Table 4: Per-packet latency statistics over 20 repeated 60-s UDP probes per bearer (mean and standard deviation in milliseconds, averaged across the 20 runs, which were mutually consistent).

Bearer	Mean (ms)	Std. dev. (ms)
Wi-Fi 5	9.6	14.8
5G SA	11.4	2.1

7.2 MEC Plane: Adaptive Node Selection

We first measure the maximum sustainable load on the GPU-equipped Virtual Server 1 server when it processes four simultaneous H.264 streams. Table 5 reports the GPU 3D-engine utilisation as a function of resolution and frame rate. At 30 fps, resolutions of 320 × 240, 640 × 480 and 800 × 600 stay within a safe 75% to 85% utilisation; the 720 p resolution only fits at 25 fps. This confirms that the chosen 640 × 480 input resolution is a sensible default for four-stream MEC offloading.

Table 5: GPU 3D-engine utilisation on Virtual Server 1 when processing four simultaneous H.264 streams. Values are steady-state utilisation sampled during sustained four-stream processing.

Resolution	15 fps	25 fps	30 fps
320 × 240	40%	60%	78%
640 × 480	45%	65%	82%
800 × 600	50%	70%	85%
1280 × 720	60%	92%	—(unstable)

We then evaluate the node selector of Algorithm 1 under a controlled workload sequence. Fig. 3 plots, for every 2 s telemetry sample along the horizontal axis over a 230 s run, the weighted utility index idx of (4) computed for each server on the vertical axis. The figure is read as follows: a lower index marks the less-pressured node, so at any instant the selector streams to whichever curve is lower, and a hand-over corresponds to a crossing of the two curves. The dashed vertical lines mark workload events—T1, T2 and T3 (upward arrows) are the activation instants and the unlabelled lines near samples 85, 100 and 102 the corresponding releases. Throughout the run the robot streams a single 640 × 480 video; the three synthetic CPU/RAM workloads are two on the GPU node Virtual Server 1 (T1 at sample 16, T3 at sample 46) and one on the CPU-only Virtual Server 2 (T2 at sample 29).

Following the strict-improvement guard of Algorithm 1, the stream is served by whichever node holds the lower index and is retargeted only when the other node becomes strictly better. The trace divides into four phases. (i) Baseline (up to sample 16): both indices sit near 0.1 and Virtual Server 1 serves the stream because its index is marginally the lower. (ii) After T1 the Virtual Server 1 index climbs —with the expected one-cycle (2 s) telemetry lag—to ~0.25, rising above the still-idle Virtual Server 2 (~0.13), so the selector migrates the stream to Virtual Server 2. (iii) T2 then lifts the Virtual Server 2 index to ~0.55, well above Virtual Server 1's ~0.25, so the stream is handed back to Virtual Server 1. (iv) The critical window: T3 spikes the Virtual Server 1 index to ~0.72 (sample ≈ 47)—now clearly above Virtual Server 2's ~0.55—so the selector correctly migrates the stream off the overloaded GPU node onto Virtual Server 2. The two indices then sit close together (~0.60 vs. ~0.55) and oscillate as the pressure weights re-balance each cycle; this is the regime

that stresses the selector most, and it is precisely here that the strict-improvement guard earns its place—by retaining Virtual Server 2 unless Virtual Server 1 becomes strictly better, it prevents the per-cycle thrashing that a plain arg min would produce at such a near-tie. As the workloads are released (samples 85–102) both indices relax toward the ~0.1 baseline and the stream returns to Virtual Server 1. Crucially, no video frames are dropped during any hand-over, because the OpenVPN overlay preserves the underlying UDP socket across a change of the robot's `udpsink` target.

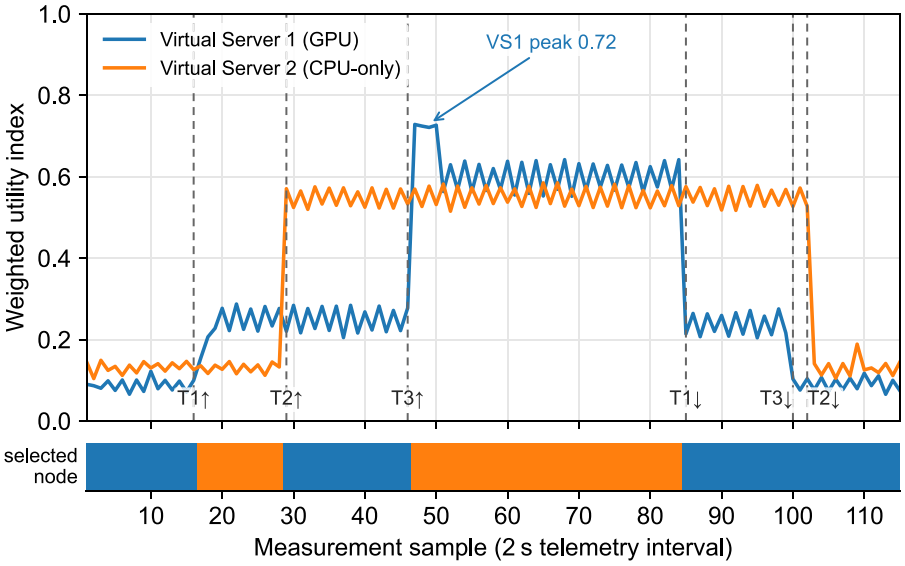

Figure 3: Adaptive MEC node selection under synthetic load. Top: for each 2 s telemetry sample (horizontal axis) the plot shows the weighted utility index idx of (4) (vertical axis) for Virtual Server 1 (GPU) and Virtual Server 2 (CPU-only) over a 230 s run; a lower index denotes the less-pressured node. Dashed vertical lines mark the workload events: CPU/RAM loads are activated on Virtual Server 1 (T1 at sample 16, T3 at sample 46) and on Virtual Server 2 (T2 at sample 29), then released (T1↓ at 85, T3↓ at 100, T2↓ at 102). Bottom: the "selected node" bar shows which node actually serves the stream under the strict-improvement guard of Algorithm 1 (blue = Virtual Server 1, orange = Virtual Server 2). When Virtual Server 1 is loaded its index rises above Virtual Server 2 and the stream migrates off it (samples 17–28 and, most sharply, at the ~0.72 spike near sample 47); the guard then holds the stream on Virtual Server 2 through the near-tie rather than thrashing, and returns it to Virtual Server 1 once the loads are released. No video frames are dropped during any hand-over.

7.3 Energy and Memory Footprint on the Robot

We compare three operating regimes on the robot: (i) idle, with ROS 2 running but no perception workload; (ii) GStreamer, with the streaming pipeline of Listing 3 active but inference off-loaded to MEC; (iii) local YOLO, with the same YOLO26-M model running on the Jetson Orin NX. Power draw is measured at the battery output; memory usage is read from the kernel `free` interface. We then go one step further (Section 7.4) and ask whether the complete autonomy stack could simply run onboard and avoid the network altogether.

Table 6 confirms the central energy argument of the framework: offloading YOLO inference to a MEC node reduces the robot power draw by 5 W and the RAM footprint by more than half. This 5 W is 40% of the platform's compute power budget—measured with the platform stationary, so that propulsion is excluded—not of total vehicle power. Because the propulsion motors dominate the total power budget of a moving robot and their draw is highly load-dependent, we do not attempt to translate this saving into a precise mission-time figure; the relevant point is that lowering the compute power draw extends the achievable operating time

per charge. The memory figures also explain why the two perception tasks are split across planes: onboard YOLO inference alone already occupies 6.0 of the Jetson's 8 GB, leaving insufficient headroom to host the LiDAR localizer concurrently (quantified further in Section 7.4). Switching the bearer from Wi-Fi to 5G adds approximately 300 mW of modem power, which is negligible at the system level but worth noting in long-duration mission planning.

Table 6: Power draw and RAM utilisation of the ROSMASTER R2 in three operating regimes. Values are steady-state readings taken during sustained operation in each regime.

Regime	Power (W)	RAM (GB)
Idle (ROS 2 only)	7.2	2.6
GStreamer + MEC offloading	7.4	2.7
Local YOLO inference	12.3	6.0

7.4 Onboard Full-Stack Feasibility

Table 6 isolates a single workload at a time; it does not yet establish whether the network could be avoided entirely by running everything locally. To test this directly, we executed every perception and control component on the Jetson Orin NX simultaneously—YOLO26 object detection, the SAC vision-based line-keeping policy of Section 6.3, the LiDAR ResNet-1D-SE localizer of Section 6.2 and the H.264 streaming pipeline—with every model exported to TensorRT-FP16 and the optimisations of Section 6 applied.

Even with these optimisations, the platform was driven to its limit: the per-frame detection latency rose to approximately 120 ms (±10 ms), i.e., only 8 to 10 fps, against the camera-native 30 fps that the offloaded path sustains (bounded by the ~67 ms end-to-end latency of Table 3). Power draw exceeded the 12.3 W of the local-YOLO regime, and under the MAXN power profile the junction temperature reached the thermal-throttling threshold after roughly 30 min of continuous operation.

Crucially, beyond the lower mean frame rate, throttling made the per-frame time increasingly variable—individual frames took markedly longer—and, exactly as for the network jitter discussed above, it is this growing variance, more than the lower average rate, that degraded closed-loop driving reliability. We therefore treat full onboard execution not as a steady-state operating mode but as a degraded fall-back: offloading the detector is precisely what frees the compute, thermal and energy budget that the safety-critical onboard tasks—localization and line-keeping—require.

7.5 LiDAR Localization: Sim-to-Real Transfer

We finally evaluate the LiDAR localizer trained in the digital-twin plane. The model is trained exclusively on synthetic data generated in Isaac Sim with the domain randomization schedule of Section 6.2, then deployed on the Jetson Orin NX and evaluated on a held-out simulation set and on a real-world set of LiDAR scans referenced against 50 surveyed ground-truth waypoints that span the navigable area; the reported errors are aggregated over the full set of real scans rather than over the reference points alone.

Table 7 reports the localization error. The sim-to-real gap is moderate: the mean error grows from 4.1 cm on the held-out simulation set to 7.8 cm on the real-world set, with no real-world fine-tuning at all. The P95 error—the level below which 95% of per-scan errors fall—rises proportionally, from 9.3 to 17.5 cm, so the tail of the distribution scales with the mean rather than developing a heavy sim-to-real outlier component. Both numbers are well within the tolerance of the downstream behaviour-planning module, confirming that domain randomization alone—without adversarial domain adaptation or system identification—is sufficient

for this class of robot and environment, in contrast to settings where the residual reality gap warrants an explicit domain-adaptation stage [25]. The full inference pipeline (LiDAR pre-processing, TensorRT-FP16 forward pass, Soft-Argmax, ROS 2 publication) runs in approximately 6 ms on the Jetson Orin NX with a 95th-percentile latency of 8.4 ms, which leaves a substantial margin below the 125 ms budget imposed by the LiDAR's native 8 Hz rate.

Table 7: Mean and 95th-percentile (P95) Euclidean pose error of the ResNet-1D-SE localizer on the held-out simulation set and on the real-world set. For a set of N scans with per-scan Euclidean pose errors $e_1, \ldots, e_N$, the P95 error is the smallest value that at most 5% of the e_i exceed (equivalently, the 0.95 empirical quantile of the error distribution). It characterises the tail of the error—the near-worst-case accuracy the behaviour planner must tolerate—whereas the mean characterises typical accuracy.

Evaluation Set	Mean Error (cm)	P95 Error (cm)
Simulation (held-out)	4.1	9.3
Real-world (no fine-tuning)	7.8	17.5

To understand which design choices drive this accuracy, we ablated the localizer's training and decoding strategy, measuring the fraction of predictions that fall within 5 cm of ground truth (Fig. 4). The baseline—only eight scan orientations in training, with direct mean-squared-error coordinate regression—places 54.5% of predictions within 5 cm. Counter-intuitively, simply quadrupling the data to 36 orientations reduces this to 43.3%: the additional symmetric views destabilise the regression head rather than help it, confirming that for this problem the decoding architecture—not the volume of data—is the limiting factor. Replacing the regression head with a grid-classification head decoded by the Soft-Argmax operator of Section 6.2 raises the figure to 63.4%, and adding the full domain-randomization schedule lifts it to 73.3%—an isolated gain of 9.9 percentage points (pp) attributable to domain randomization alone. On the real-world set this final model keeps a median error of 3.1 cm, well below the 7.8 cm mean of Table 7; the gap between median and mean stems from a minority of poses at symmetric locations with near-identical LiDAR profiles, for which a short temporal filter over consecutive scans is the planned remedy.

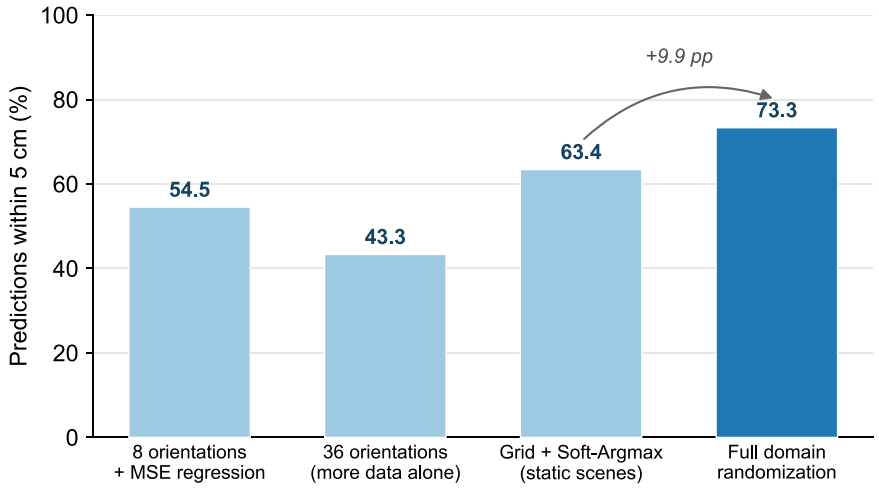

Figure 4: Ablation of the LiDAR localizer: share of pose predictions within 5 cm of ground truth for four successive design choices. Adding training views alone (8 → 36 orientations) lowers accuracy; replacing direct coordinate regression with a grid-classification head decoded by Soft-Argmax recovers and improves it, and domain randomization adds a further 9.9 percentage points.

7.6 Summary of Deployment Modes

Combining the above measurements, Table 8 compares the three deployment modes supported by the proposed framework along the metrics that matter most for an autonomous mobile robot. To make the table self-contained, its rows are drawn from the experiments already reported above: the Wi-Fi and 5G end-to-end latency values are the single-stream entries of Table 3, and the corresponding P95 values are the 95th percentile of the per-frame latencies recorded during those same single-stream runs; the onboard latency column reports detection-only inference measured directly on the Jetson Orin NX (no network path); and the power and RAM rows are taken from Table 6. Consistently with Table 3, the latency row reports run-averaged means, not medians.

Table 8: Comparison of deployment modes for the YOLO26-M perception workload on the ROSMASTER R2 platform. [≤]The onboard column reports the YOLO26-M detector in isolation; when the complete perception-and-control stack runs onboard simultaneously, the platform degrades to 8 to 10 fps with thermal throttling after roughly 30 min (Section 7.4).

Metric	Onboard[≤]	Wi-Fi + MEC	5G SA + MEC
E2E latency, 1 stream (ms)	38 (no network)	62	67
P95 latency (ms)	46	96	76
Throughput ceiling (streams)	1 (HD 25 fps)	4 (<90 Mbps)	4 (≤60 Mbps)
Robot power draw (W)	12.3	7.4	7.7
Robot RAM usage (GB)	6.0	2.7	2.7
Sensitivity to interference	None	High	Low

The three modes occupy clearly distinct operating points. Onboard inference of the detector in isolation actually has the lowest, network-free end-to-end latency (38 vs. 62 and 67 ms; Table 8); it is the only option when the robot leaves the radio coverage area, but it raises the power draw by roughly two-thirds and saturates the GPU budget. It is therefore important to state plainly that offloading the detector is not a latency win—Table 3 ranks only the two networked modes against each other, and the case for offloading rests on energy, thermal headroom and full-stack feasibility (Sections 7.3–7.4 and 8). Among the networked modes, Wi-Fi + MEC offers the lower single-stream latency at the cost of high variability and rapid degradation under load, whereas 5G SA + MEC adds a 5 ms single-stream overhead but is markedly more predictable, overtakes Wi-Fi from three streams onward, and supports four streams simultaneously, which is the configuration that the framework targets for production deployments.

8 Discussion

The results of Section 7 support five practical observations that generalise beyond the ROSMASTER R2 platform used in this study.

Average latency is the wrong target. At a single stream Wi-Fi delivered slightly lower average latency than 5G SA (62 vs. 67ms end-to-end, Table 3), yet its jitter (14.8 ms standard deviation) is roughly seven times that of 5G SA, and the advantage disappears entirely once three or more streams share the bearer. For closed-loop control the worst-case latency matters far more than the average, because the controller must remain stable under the maximum delay the bearer can produce; evaluations of connected robotic platforms should therefore always report jitter, and the load at which the measurement was taken, alongside latency.

Offloading is an energy decision, not a latency one. The detector runs with lowest latency onboard (Section 7.6); the case for offloading is energy and thermal. Running YOLO26 on the Jetson Orin NX raises

power draw by roughly two-thirds (7.4 to 12.3 W), and becomes decisive when the full stack is forced onboard (Section 7.4): detection, line-keeping, localization and streaming together drop the platform to 8 to 10 fps and into thermal throttling after ≈30 min. Onboard execution is thus a degraded fall-back, and the offloading decision should be evaluated under a power- and thermal-aware utility function, not only a latency-aware one.

Heterogeneous MEC nodes need a non-linear node selector. Choosing an offload target across heterogeneous nodes is not a matter of minimising a single resource but of avoiding any bottleneck, and a linear score cannot express that goal. The pressure function of (1) makes the utility index react to a node's proximity to saturation rather than to its raw utilisation: as any one metric approaches its ceiling, its weight in (3) grows hyperbolically, so a node close to exhausting a single resource is penalised even when its average load still looks moderate. The effect is visible in the re-balancing window of Section 7.2: when the GPU node takes a second workload its index climbs steeply to ~0.72, above the CPU-only node's ~0.55, so the selector correctly migrates the stream off the node nearest saturation—even though that node has the more powerful hardware—rather than clinging to it on the basis of raw capacity. Two mechanisms cooperate here. The non-linear pressure function guarantees that a node approaching its ceiling is penalised quickly enough for this migration to be timely; and the strict-improvement guard of Algorithm 1, which retargets only when a node is strictly better than the current one, then suppresses the per-cycle thrashing that the ensuing near-tie (~0.60 vs. ~0.55, oscillating) would otherwise induce. The selector thus encodes headroom explicitly—a node at 50% load with a powerful GPU is not scored the same as a node at 50% load on a CPU-only VM—while remaining stable at the crossing. The general lesson is that offload-target selection across heterogeneous edge nodes should be driven by a saturation-aware, non-linear utility, paired with a hysteresis-like guard against flapping, rather than by instantaneous load alone.

Domain randomization alone reaches sub-decimetre mean accuracy. The localization results of Section 7.5 confirm that, with carefully chosen domain randomization (Gaussian range noise, ray dropout, random occlusion, cyclic shift), a localizer can be trained entirely in simulation and deployed on a physical robot without any real-world fine-tuning: the increase from 4.1 to 7.8 cm mean error is easily absorbed by the downstream behaviour planner. For LiDAR-only localization in static indoor environments, the cost of a careful digital twin is repaid many times over by eliminating the real-world data-collection phase.

Integrate tightly but couple loosely. The three planes form one integrated system—co-designed at the two interfaces that matter, the RTT-coupled node selector and the offloading-enabled onboard compute headroom—yet communicate only through a stable OpenVPN overlay and a ROS 2 control channel. This loose coupling paid off concretely during the study: a second MEC node (the Virtual Server 2) was added without changing anything on the robot, and the bearer could be switched mid-experiment with no visible disruption to the running ROS 2 stack. We recommend the same pattern—one integrated system built from loosely coupled planes—for any deployment combining wireless connectivity, edge compute and learned policies.

Limitations

Three limitations remain. First, the full autonomy stack—LiDAR localization and vision-based driving—is evaluated on a single ROSMASTER R2; the connectivity and MEC planes, however, are driven by two to three physical robots streaming concurrently, so the radio link and the node selector's stream-distribution role are a demonstrated capability rather than a projection. What remains out of scope is cooperative perception and control (robots acting on one another's detections or poses, rather than merely contending for shared infrastructure), the next step outlined in Section 9. Second, the node selector currently chooses between two MEC nodes, only one with a GPU; studying how it trades GPU power against network

proximity would require several GPU servers of different generations, unavailable at the time of writing. Third, validation is indoor-only: outdoor deployment would expose the framework to weaker signal, larger shadowing and frequent handovers; our intra-cell measurements indicate the OpenVPN tunnel survives handovers gracefully, but a full outdoor study remains future work.

9 Conclusion and Future Work

This paper presented an integrated three-plane framework for connected autonomous mobile robots combining a private 5G SA access network, an MEC infrastructure with adaptive workload offloading, and a digital-twin training pipeline on NVIDIA Isaac Sim. It was fully implemented on the Yahboom ROSMASTER R2 with a Jetson Orin NX, a Quectel RM530N-GL 5G modem and a Raspberry Pi 5 orchestrator: a ResNet-1D-SE network trained entirely in simulation performs LiDAR localization, a Soft Actor-Critic policy provides a network-independent vision-based driving fall-back, and a YOLO26-M detector—a representative off-the-shelf workload—is offloaded to one of two heterogeneous MEC nodes by a utility-based node selector that combines CPU, RAM, GPU and RTT telemetry into a single non-linear ranking.

The unified evaluation showed that the three deployment modes—onboard only, Wi-Fi + MEC and 5G SA + MEC—occupy clearly distinct operating points along the latency, throughput, energy and stability axes. The private 5G SA link delivers seven times lower jitter than Wi-Fi for a 5 ms higher single-stream end-to-end latency, an overhead that is reversed once three or more streams share the bearer; offloading YOLO inference halves the robot's RAM footprint and cuts its compute power budget by 40% (propulsion excluded); and running the entire perception-and-control stack onboard is only a degraded fall-back (8 to 10 fps with thermal throttling after roughly half an hour). The multi-stream experiments, driven by two to three concurrent robots, validate the connectivity and MEC planes under genuine multi-robot load, and the sim-trained localizer transfers to the physical robot with a mean real-world error below 8 cm and no fine-tuning.

Three directions are immediate. First, having validated infrastructure sharing across two to three concurrent robots, the framework will be extended towards genuine cooperative behaviour—robots acting on one another's detections and poses (detection sharing to extend the field of view, or leader–follower convoys)—rather than merely contending for shared resources. Second, an adaptive bitrate controller will be added on top of the GStreamer pipeline to react to the 60 Mbps throughput envelope of the current 5G modem and the variable GPU-3D load on Virtual Server 1 (Section 7.2). Third, the SAC driving policy, already deployed onboard as a network-independent fall-back (Sections 6.3 and 7.4), will be hybridised with a model-predictive controller for stronger formal safety guarantees. Together these extensions move the framework towards genuine cooperative perception and swarm control.

Acknowledgement: The authors thank the Faculty of Electrical Engineering and Informatics, Technical University of Košice, for access to the private 5G SA testbed, the GPU MEC server and the autonomous-vehicle laboratory.

Funding Statement: Co-funded by the European Union through the Slovakia Programme under Project No. NFP401101C689: Research, Development, Testing and Validation of AI-Enabled 5G/6G and Edge Computing Integration for Ultra-Low-Latency, Reliable and Secure Communication Systems for Industrial, Transportation and Healthcare Applications.

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization, Daniel Šolc, Eugen Šlapák and Juraj Gazda; software and data curation, Daniel Šolc and René Ivančák; formal analysis, visualization, writing—original draft and project administration, Daniel Šolc; resources and funding acquisition, Juraj Gazda and Gabriel Bugár; supervision, Juraj Gazda and Eva Chovancová; methodology, validation, investigation and writing—review and editing, all authors. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the Corresponding Author, Daniel Šolc, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chen L, Zhang Y, Tian B, Ai Y, Cao D, Wang FY. Parallel driving OS: a ubiquitous operating system for autonomous driving in CPSS. IEEE Trans Intell Veh. 2022;7(4):886–95.
2. Wang J, Liu J, Kato N. Networking and communications in autonomous driving: a survey. IEEE Commun Surv Tutor. 2019;21(2):1243–74. doi:10.1109/comst.2018.2888904.
3. Lema MA, Laya A, Mahmoodi T, Cuevas M, Sachs J, Markendahl J, et al. Business case and technology analysis for 5G low Latency applications. IEEE Access. 2017;5:5917–35.
4. Mao Y, You C, Zhang J, Huang K, Letaief KB. A survey on mobile edge computing: the communication perspective. IEEE Commun Surv Tutor. 2017;19(4):2322–58.
5. Tobin J, Fong R, Ray A, Schneider J, Zaremba W, Abbeel P. Domain randomization for transferring deep neural networks from simulation to the real world. In: Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2017 Sep 24–28; Vancouver, BC, Canada. p. 23–30.
6. Peng XB, Andrychowicz M, Zaremba W, Abbeel P. Sim-to-real transfer of robotic control with dynamics randomization. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA); 2018 May 21–25; Brisbane, QLD, Canada. p. 3803–10.
7. Yang D, Janes A, Danek J, Gorla P, Xu X, Becvar Z, et al. Advancing collaborative research in communication and Robotics: insights from experiments in a remote experience center. In: Proceedings of the 2025 IEEE International Conference on Communications Workshops (ICC Workshops); 2025 Jun 8–12; Montreal, QC, Canada. p. 1858–63.
8. Hong Z, Chen W, Huang H, Guo S, Zheng Z. Multi-hop cooperative computation Offloading for industrial IoT-edge–cloud computing environments. IEEE Trans Parallel Distrib Syst. 2019;30(12):2759–74. doi:10.1109/tpds.2019.2926979.
9. Chen X, Shi Q, Yang L, Xu J. ThriftyEdge: resource-efficient edge computing for intelligent IoT applications. IEEE Netw. 2018;32(1):61–5. doi:10.1109/mnet.2018.1700145.
10. Tiwari R, Khapre S, Singh A. Reinforcement learning in robotic systems: a review on sim-to-real transfer. Robot Auton Syst. 2026;198:105327.
11. Roda-Sanchez L, Zanzi L, Li X, Garí G, Costa-Pérez X. Network digital twin for 5G-enabled mobile robots. In: Proceedings of the 2025 IEEE Wireless Communications and Networking Conference (WCNC); 2025 Mar 24–27; Milan, Italy. p. 1–6.
12. GS MEC 003 v2.2.1. ETSI ISG MEC. Multi-access edge computing (MEC); framework and reference architecture. Sophia Antipolis: France: European Telecommunications Standards Institute; 2020.
13. Xavier R, Silva RS, Ribeiro M, Moreira W, Freitas L, Oliveira-Jr A. Integrating multi-access edge computing (MEC) into open 5G core. Telecom. 2024;5(2):433–50. doi:10.3390/telecom5020022.
14. Teerapittayanon S, McDanel B, Kung HT. Distributed deep neural networks over the cloud, the edge and end devices. In: Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); 2017 Jun 5–8; Atlanta, GA, USA. p. 328–39.
15. Xie B, Cui H. Deep reinforcement learning-based dynamical task offloading for mobile edge computing. J Supercomput. 2025;81(1):35. doi:10.1007/s11227-024-06603-x.
16. Raunholt T, Rodriguez I, Mogensen P, Larsen M. Towards a 5G mobile edge cloud planner for autonomous mobile robots. In: Proceedings of the 2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall); 2021 Sep 27–30; Norman, OK, USA. p. 1–5.
17. Tran-Dang H, Kim DS. Digital twin-empowered intelligent computation offloading for edge computing in the era of 5G and beyond: a state-of-the-art survey. ICT Express. 2025;11(1):167–80. doi:10.1016/j.icte.2025.01.002.

18. Chen J, Zhang X, Peng X, Xu D, Wu D, Xin R. Shuffle-octave-yolo: a tradeoff object detection method for embedded devices. J Real-Time Image Process. 2023;20(2):25. doi:10.1007/s11554-023-01284-w.
19. Amin RA, Hasan M, Wiese V, Obermaisser R. FPGA-based real-time object detection and classification system using YOLO for edge computing. IEEE Access. 2024;12:73268–78. doi:10.1109/access.2024.3404623.
20. Grieves M. Digital twin: manufacturing excellence through virtual factory replication. Melbourne, FL, USA: Florida Institute of Technology; 2014.
21. Liu M, Fang S, Dong H, Xu C. Review of digital twin about concepts, technologies, and industrial applications. J Manuf Syst. 2021;58:346–61. doi:10.1016/j.jmsy.2020.06.017.
22. NVIDIA Corporation. NVIDIA Isaac sim: robotics simulation and synthetic data generation. 2024 [cited 2026 May 18]. Available from: https://developer.nvidia.com/isaac-sim.
23. Liu Y, Xu H, Liu D, Wang L. A digital twin-based sim-to-real transfer for deep reinforcement learning-enabled industrial robot grasping. Robot Comput Manuf. 2022;78(12):102365. doi:10.1016/j.rcim.2022.102365.
24. Tiboni G, Arndt K, Kyrki V. DROPO: sim-to-real transfer with offline domain randomization. Robot Auton Syst. 2023;166:104432. doi:10.1016/j.robot.2023.104432.
25. Shakerimov A, Alizadeh T, Varol HA. Efficient sim-to-real transfer in reinforcement learning through domain Randomization and domain adaptation. IEEE Access. 2023;11(3):136809–24. doi:10.1109/access.2023.3339568.
26. Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. Int Conf Mach Learn. 2018;80:1861–70. doi:10.48550/arXiv.1801.01290.
27. NVIDIA Corporation. NVIDIA jetson orin NX series: module data sheet. 2025 [cited 2026 Aug 05]. Available from: https://developer.nvidia.com/embedded/jetson-modules.
28. Raspberry Pi Ltd. Raspberry Pi 5 product brief. 2025 [cited 2026 Aug 05]. Available from: https://www.raspberrypi.com/products/raspberry-pi-5/.
29. Jocher G, Qiu J. Ultralytics YOLO26. 2026 [cited 2026 Jun 17]. Available from: https://docs.ultralytics.com/models/yolo26/.
30. Roboflow, Inc. Roboflow: computer vision tools for dataset management and annotation. 2024 [cited 2026 Jun 17]. Available from: https://roboflow.com.
31. Pixar Animation Studios. Universal scene description (OpenUSD). 2024 [cited 2026 Jun 17]. Available from: https://openusd.org.
32. Nibali A, He Z, Morgan S, Prendergast L. Numerical coordinate rcegression with convolutional neural networks. arXiv:1801.07372. 2018.
33. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016 Jun 27–30; Las Vegas, NV, USA. p. 770–8.
34. Hu J, Shen L, Sun G. Squeeze-and-Excitation networks. In: Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2018 Jun 18–23; Salt Lake City, UT, USA. p. 7132–41.
35. Shenzhen EAI Technology Co., Ltd. YDLIDAR TG30 data sheet. 2019. DOC#: 01.13.005300. [cited 2026 Aug 05]. Available from: https://www.ydlidar.com/product/category/tof/ydlidar-tg30.html.
36. Loshchilov I, Hutter F. Decoupled weight decay regularization. In: Proceedings of the 7th International Conference on Learning Representations (ICLR); 2019 May 6–9; New Orleans, LA, USA.
37. NVIDIA Corporation. NVIDIA TensorRT. 2024 [cited 2026 Jun 17]. Available from: https://developer.nvidia.com/tensorrt.
38. Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-Baselines3: reliable reinforcement learning implementations. J Mach Learn Res. 2021;22(268):1–8.