Machine Learning for Autonomic System Operation in the Heterogeneous Cloud-Edge Continuum

Contract Number 101092912

D5.3 Final Integration and Evaluation Report

Lead Partner	UBIW
Contributing Partners	UTH, UNICAL, TUD, UCD, INRIA, FhP, NTT, MLNX, NUBIS, CC, AUG
Owner / Main Author	Bruno Pereira (UBIW)
Contributing Authors	UTH: Christos Antonopoulos, Nikos Bellas, Alexandros Patras, Foivos Pournaropoulos, Spyros Lalis; UNICAL: Raffaele Gravina, Gianluca Aloi, Michele Gianfelice, Francesco Pupo, Giovanni Staino, Andrea Manna, Giuseppe Caliciuri; TUD: Kaitai Liang, Rui Wang; UCD: Dimitris Chatzopoulos, John Byabazaire, Theodoros Aslanidis; INRIA: Valeria Loscri, Jiali Xu, Ildi Alla; FhP: Carlos Resende, João Oliveira; NTT: Massimiliano Rossi, Andrea Pazienza, Marco Brambilla; MLNX: Dimitris Syrivelis; NUBIS: Charalampos Mainas, Dimitrios Karadimas, Konstantinos Papazafeiropoulos, Maria Gkoutha, Maria Gkeka, Anastasia Mallikopoulou, Panagiotis Mavrikos, Anastassios Nanos, CC: Marcell Feher; AUG: Dimitris Akridas, Spyros Evangelatos, Giorgos Giannios, Phivos Papapanagiotakis, Nektarios Sfyris
Reviewers	Filipe Sousa (FhP), Nikolaos Bellas (UTH)
Contractual Delivery Date	31/12/2025
Actual Delivery Date	16/1/2026
Version	1.0
Dissemination Level	Public

HORIZON EUROPE

The research leading to these results has received funding from the European Community's Horizon Europe Programme (HORIZON) under grant no. 101092912.

Change Log

Version	Summary of Changes
0.1	Draft Proposal.
0.2	First contribution Round.
0.3	First release of the document.
0.4	Internal Review
0.5	Second contribution round
1.0	Final version for release to the EC.

Table of Contents

CHANGE LOG **2**
TABLE OF CONTENTS **3**
LIST OF FIGURES **8**
LIST OF TABLES **11**
SUMMARY **12**
ABBREVIATIONS **13**
1 INTRODUCTION **15**
2 GENERAL INTEGRATION OVERVIEW **16**
2.1 CONTEXT 16
2.2 KEY FUNCTIONALITIES OF THE INTEGRATED MLSYSOPS FRAMEWORK 17
2.2.1 Application and System Infrastructure Descriptions *17*
2.2.2 Agent Hierarchy *17*
2.2.3 Application Deployment *18*
2.2.4 Application Component Versions and Execution Enclaves *18*
2.2.5 Relocation of Application Components *18*
2.2.6 Application-level Communication and Traffic Redirection *18*
2.2.7 Telemetry *18*
2.2.8 ML Model Discovery, Activation, Invocation, and Deactivation *19*
2.2.9 Flexible Exploitation of Accelerators for Compute-intensive Operations *19*
2.2.10 Node-level settings *19*
2.3 INTEGRATION STATUS 19
3 USE-CASE SPECIFIC INTEGRATION **23**
3.1 APPLICATION-DRIVEN FRAMEWORK ADAPTATION/INTEGRATION/TESTING 23
3.2 SMART CITY APPLICATION 24
3.2.1 Concept, Role of MLSysOps, Application KPI & System Optimization Objective *24*
3.2.2 Application Structure *25*
3.2.3 Development of Application Components *26*
3.2.4 Framework Instantiation and Agent Logic *27*
3.2.5 Integration and Testing *28*
3.2.6 Data Collection *31*
3.2.7 Machine Learning Training and Evaluation *32*
3.2.8 Evaluation *33*
3.3 SMART AGRICULTURE APPLICATION 43
3.3.1 Concept, Role of MLSysOps, Application KPI & System Optimization Objective *43*
3.3.2 Application Structure *43*
3.3.3 Development of Application Components *45*
3.3.4 Framework Instantiation and Agent Logic *45*
3.3.5 Integration and Testing *47*
3.3.6 Data Collection *51*
3.3.7 Machine Learning Training and Evaluation *52*
3.3.8 Evaluation *62*
3.4 OVERALL STATUS ASSESSMENT 70
4 EVALUATION OF APPLICATION-NEUTRAL FUNCTIONALITIES **72**
4.1 STRUCTURED SYSTEM AND APPLICATION DESCRIPTION 72
4.1.1 RG-KPI 1.1: System infrastructure description can capture at least the infrastructure of MLSysOps application testbeds and research testbeds *72*

4.1.2 RG-KPI 1.2: Describe application components so that they can be freely placed in at least two different layers of the continuum and linked with RG-KPI 2.1 and RG-KPI 2.2 72
4.2 Application Deployment and Orchestration 72
4.2.1 RG-KPI 2.1: Deploy an application with at least three components so that at least one component is placed at the Far-Edge, Smart-Edge, and Edge/Cloud Infrastructure 73
4.2.2 RG-KPI 2.2: Deploy application components on at least two types of Smart-Edge and two types of Far-Edge nodes featuring different CPUs/ Micro Controller Units (MCU) and/or sensors 73
4.2.3 RG-KPI 2.3: Have a deployment where at least two application components can interact with each other over either 4G/Internet, Wi-Fi, IEEE 802.15.4, or Bluetooth links 73
4.2.4 RG-KPI 2.4: The initial deployment plan is close to optimal, i.e., within 10% vs. a plan produced by an offline/oracle algorithm 74
4.2.5 RG-KPI 2.5: When changes in system state and application execution profile are detected, produce/execute an adapted deployment plan close to optimal, i.e., within 10% vs. a plan produced by an offline/oracle algorithm 74
4.3 Node-level Resource Usage & Management 75
4.3.1 RG-KPI 3.1: Different combinations of power configurations spanning the energy efficiency space are supported for at least one type of Cloud/Edge Infrastructure node, Smart-Edge node, and Far-Edge node 75
4.3.2 RG-KPI 3.2: Offer at least three different function implementations (CPU Only, GPU, Field Programmable Gate Arrays (FPGA)) that are transparently invoked by at least two different application components 75
4.3.3 RG-KPI 3.3: The performance overhead for the transparent usage of the acceleration hardware should be low (< 5% vs. a hardwired invocation of the respective implementation) 76
4.3.4 RG-KPI 3.4: Offer acceleration support for at least one high-level ML framework (TensorFlow or PyTorch) for inference and training 76
4.3.5 RG-KPI 3.5: The initial node level and local application configuration are close to optimal, within 10% vs. a configuration produced by an offline/oracle algorithm 76
4.3.6 RG-KPI 3.6: When changes in the local node state and application execution profile are detected, the adapted configuration falls within a 10% margin vs. a plan produced by an offline/oracle algorithm 76
4.4 Storage 77
4.4.1 RG-KPI 4.1: Integrate at least 20 cloud storage locations across at least 4 commercial cloud providers. 77
4.4.2 RG-KPI 4.2: Share availability and performance measurements with the (ML-driven) policies within 60 minutes of the data transfer. 77
4.4.3 RG-KPI 4.3: Record availability and performance of all cloud and edge storage locations at least once every 6 hours. 78
4.4.4 RG-KPI 4.4: Share file access events with the (ML-driven) policies within 15 minutes of the event. 78
4.4.5 RG-KPI 4.5: Realize the storage representation changes decided by the (ML-driven) policies in a maximum of 15 minutes per MB of data affected. 78
4.5 Trust 79
4.5.1 RG-KPI 5.1: Reputation/credit calculation is performed in real-time, within a few milliseconds. The calculation aims to consume <=5% of the energy consumption during normal application execution. The bandwidth cost is similar to the cost of a normal application communication message (with or without authentication, the bandwidth performance remains similar). Furthermore, resource and application allocation and policy adjustments related to credit should be performed on the same scale as above. 79
4.5.2 RG-KPI 5.2: The authentication should be 100% accurate, i.e., once an entity passes the authentication, it should be 100% what it claims to be. 80
4.5.3 RG-KPI 5.3: The time for adapting the encryption/decryption to the trust level of nodes will be a few milliseconds, and there should not be more than 5% energy cost vs normal application execution. The extra bandwidth the encryption brings will be restricted by the 15-20% expansion of the original communication data. 80
4.6 Wireless Network Management and Security at the Edge 81

4.6.1 RG-KPI 6.1: Manage a network with at least 3 edge nodes (one mobile Smart-Edge node and some fixed Smart-Edge nodes) with latency kept below 10 milliseconds and Packet Delivery Ratio (PDR) higher than 90%.....81
4.6.2 RG-KPI 6.2: Manage a network with at least 3 edge nodes (one mobile Smart-Edge node and some fixed Smart-Edge nodes) with anomalies detected with an accuracy higher than 90% and detection time lower than 100 milliseconds.....82
4.7 5G NETWORK MANAGEMENT.....83
4.7.1 RG-KPI 7.1: Latency is improved with the autonomic changes in the connectivity path between two or more interacting application components (core network NF) 84
4.8 OPTICAL NETWORKING IN THE DATACENTRE 85
4.8.1 RG-KPI 8.1: The network management policy dynamically switches between at least two network topologies (Fat-Tree, 3D-Torus) depending on the arriving workload mix, excluding switching layers that are not required and switching off the relevant devices.....86
4.8.2 RG-KPI 8.2: The network management policy dynamically changes at least one parameter of the physical network (e.g., bandwidth steering) (R8.1).....86
4.8.3 RG-KPI 8.3: Use at most 2 switching layers to execute application workloads involving Deep Learning Recommendation Models (DRLM) and Large Language Models (LLM). (R8.1, R8.2).....86
4.8.4 RG-KPI 8.4: The network power consumption is reduced by 30% due to the power down of the network elements of the bypassed layer. (R8.1) 86
4.8.5 RG-KPI 8.5: The network port-to-port latency is reduced by 25% with the use of a switching layer bypass. (R8.1, R8.2).....87
4.8.6 RG-KPI 8.6: Arriving workloads are expected to be 100% accommodated (job scheduling and topology reconfiguration on an AI Cluster) based on the decisions generated by the MLSysOps framework. (R8.1, R8.2).....87
4.9 ENERGY-EFFICIENT AND GREEN COMPUTING IN DATACENTRES.....87
4.9.1 RG-KPI 9.1: Implement carbon-aware orchestration that dynamically adapts to spatio-temporal variations in carbon intensity.....88
4.9.2 RG-KPI 9.2 Achieve operators' costs within 5% of those of adaptive, non-ML state-of-the-art policies...88
4.10 MACHINE LEARNING.....89
4.10.1 RG-KPI 10.1: At least two different models are used interchangeably without modifying the underlying resource management and configuration mechanisms.....89
4.10.2 RG-KPI 10.2: The effectiveness of continual learning shall improve system performance whenever model drifting is detected (i.e., it should at least bring the system to the performance levels before model drifting).....89
4.10.3 RG-KPI 10.3: Performance isolation will be achieved between the running application and the ML model (training/inference). Application QoS targets are met, although resources are used for applications and model re-training / continual learning mechanisms.....90
4.10.4 RG-KPI 10.4: Explanation mechanisms provide sufficient justifications in terms of feature importance for decisions made by an ML-based mechanism. 90
4.10.5 RG-KPI 10.5: System and running applications continue working even when ML-driven decision-making is deactivated. 91
4.10.6 RG-KPI 10.6: All relevant telemetry data is successfully captured and processed using data-centric AI techniques to produce summaries of good quality (i.e., the resulting quality will not negatively affect model performance) that can be used as experience memory and made available for further analysis or utilisation. 91
5 EVALUATION VS PROJECT LEVEL KPIS 92
5.1 DELIVER AN OPEN AI-READY, AGENT-BASED FRAMEWORK FOR HOLISTIC, TRUSTWORTHY, SCALABLE, AND ADAPTIVE SYSTEM OPERATION ACROSS THE HETEROGENEOUS CLOUD-EDGE CONTINUUM 92
5.1.1 P-KPI 1.1 At least 2 real-world applications transparently combine services from cloud and edge layers and/or different infrastructure providers.....92
5.1.2 P-KPI 1.2 Deployment and orchestration on at least 4 families of devices spanning from cloud to Far-Edge 92

5.1.3 P-KPI 1.3 Support for at least 2 different cloud/edge resource provisioning /allocation/orchestration frameworks 93
5.1.4 P-KPI 1.4 Support at least 2 AI policies for any managed resource without changes to the management mechanisms 93
5.1.5 P-KPI 1.5 Support systems with at least 250 nodes across the continuum (validated through simulation) 93
5.1.6 P-KPI 1.6 Sufficiently lightweight agent implementation, targeting memory-limited (less than 1GB) Smart-Edge devices and agent coordination overhead of less than 10% of application data traffic..... 93
5.2 Develop an AI Architecture Supporting Explainable, Efficiently Retrainable ML Models for End-To-End Autonomic System Operation in the Cloud-Edge Continuum 94
5.2.1 P-KPI 2.1 Explainable ML models with decision quality within 5% of traditional state-of-the-art resource management algorithms..... 95
5.2.2 P-KPI 2.2 Eliminate the effect of model drifting without any perceivable reduction (less than or equal to 1%) in the QoS of applications in the presence of system slack. & P-KPI 2.3 Reduce the effect of model drifting at a QoS penalty of less than 5% for running applications in the presence of resource pressure. 95
5.3 Enable Efficient, Flexible, and Isolated Execution Across the Heterogeneous Continuum..... 95
5.3.1 P-KPI 3.1 Package and deploy in the continuum applications consisting of at least 10 components, transparently and on-demand, targeting 4 execution enclaves (VMs, microVMs, unikernels, containers) at comparable latency with state-of-the-art standalone enclave generation methods (package should not exceed 60% of the generic packaging time and deployment should not exceed 10% of the generic deployment time). 95
5.3.2 P-KPI 3.2 Support Arm Cortex-M family devices as legitimate application deployment and resource orchestration targets. 96
5.3.3 P-KPI 3.3 Support resource-constrained devices (Class 1 as defined by IETF RFC 7228) over constrained networks (as defined by IETF RFC 7228) and reduce reprogramming time by at least 20% compared to a full firmware update. 96
5.3.4 P-KPI 3.4 Enable the use of at least three (3) execution devices (cloud GPU accelerators, edge CPUs, edge GPU accelerators) by a single application binary. 97
5.4 Support Green, Resource-Efficient, and Trustworthy System Operation while Satisfying Application QoS/QoE Requirements..... 97
5.4.1 P-KPI 4.1 Reduce the IT energy footprint of 2 real-world use cases by at least 20%. 97
5.4.2 P-KPI 4.2 Reduce the total operational carbon footprint of cloud/edge workloads by at least 15% compared to standard baselines. 98
5.4.3 P-KPI 4.3 Reduce network power consumption by 30% by exploiting optical circuit switching, which reduces the required number of power-hungry electrical packet switches without introducing oversubscribed or longer network paths that can affect application performance. 98
5.4.4 P-KPI 4.4 Far-Edge network latency to gateway less than 10ms and packet failures lower than 10^{-7} with a device density of 100 devices per gateway. 99
5.4.5 P-KPI 4.5 True positive rate of network anomaly detection higher than 90% and permitted false positive rate lower than 5% for critical applications. 99
5.4.6 P-KPI 4.6 Reduced local data breakout latency at the edge by 10 milliseconds on a fully cloudified standalone 5G. 99
5.4.7 P-KPI 4.7 File access times decreased by at least 20%, and repair time after permanent storage location failure was less than 24 hours per TB of data. 100
5.4.8 P-KPI 4.8 Improved security and privacy for cloud and edge systems by at least 20% compared to traditional cybersecurity solutions on ENISA and NIST benchmarks. 100
5.5 Realistic Model Training, Validation, and Evaluation 101
5.5.1 P-KPI 5.1 Use 2 application-specific testbeds motivated by real-world scenarios. 101
5.5.2 P-KPI 5.2 Use at least 2 datacentre-class and 2 smart-/Far-Edge research testbeds in combination, covering resources and setups characteristic of all layers of the heterogeneous continuum. 101

5.5.3 P-KPI 5.3 Develop/extend at least 2 simulators, one for datacentre and one for edge environments, and use them for controlled scale-out experimentation and data collection.....102
5.5.4 P-KPI 5.4 Openly share at least 10 pre-trained ML models with the community and the respective training, validation, and evaluation datasets. 102
6 KPI PERSPECTIVE ON OVERALL PROJECT STATUS.....104
6.1 REQUIREMENT GROUPS KPIS.....104
6.2 PROJECT OBJECTIVES KPIS.....107
7 CONCLUSION AND OUTLOOK.....110
APPENDIX A. SIMULATION ENVIRONMENTS AND TESTBEDS.....111

List of Figures

Figure 1: MLSysOps framework core architecture. 16

Figure 2: Approach and integration of the MLSysOps framework to a specific application use case. 23

Figure 3: MLSysOps application description for the smart city use case. 26

Figure 4: MLSysOps framework for the smart city use case. 27

Figure 5: FSM used by the cluster MLSysOps agent to control the deactivation/reactivation of the image processing component. 28

Figure 6: (a) Satellite view of UBIW testbed, and (b) Smart lamppost, Node #0. #1 and #2 29

Figure 7: Housing and lab setup for the Smart-Edge node before it is mounted on the lamppost 30

Figure 8: Accuracy of the developed noise forecasting system as a function of the threshold used at the computational part of the system 33

Figure 9: Sensitivity Analysis & Threshold Optimization Plot 34

Figure 10: Node #0 (Private Parking) – Dec 23, 2025 37

Figure 11: Node #0 (Private Parking) – Dec 25, 2025 38

Figure 12: Node #3 (City Centre) – Jan 03, 2026 40

Figure 13: 72-Hour Continuous Analysis Plots of the private testbed 41

Figure 14: 72-Hour Continuous Analysis Plots of the public testbed 41

Figure 15: MLSysOps application description for the smart agriculture use case 44

Figure 16: MLSysOps framework for the smart agriculture use case 46

Figure 17: FSM used by the cluster MLSysOps agent to control the engagement/disengagement of the drone and the deployment of the drone weed detection component accordingly. 47

Figure 18: Detection window of video playback, with green points indicating the tractor-detected weeds. 48

Figure 19: Detection window of video playback, with yellow points indicating the drone-detected weeds. 49

Figure 20: Map visualization of the video-playback weed detection. Green and yellow points indicate tractor and drone-detected weeds, respectively; red points indicate sprayed weeds. The tractor is shown with its attached sprayer; red areas are sprayed while grey areas are not. 49

Figure 21: Tuning of sensitivity settings for drone detection on a static frame. Low sensitivity: Missed weeds. Medium sensitivity: All weeds are detected. High sensitivity: All weeds are detected, but with excessive noise. 49

Figure 22: Evaluating stability of the drone's trajectory and attitude using the video playback weed detection. 50

Figure 23: The left image is a screenshot from Google Maps showing the field's dimensions, while the right image is a photograph taken on-site. There is no sprayer attached to the tractor, as the data gathered from the tests suffice for the evaluation. 50

Figure 24: Images captured by the drone and tractor systems during the data collection sessions in the field. 52

Figure 25: Prediction plot at +5 timesteps 54

Figure 26: Prediction plot at +5 timesteps 55

Figure 27: Prediction plot at +5 timesteps 56

Figure 28: Prediction plot at +5 timesteps 57

Figure 29: Prediction plot at +40 timesteps 58

Figure 30: Prediction plot at +40 timesteps 59

Figure 31: Prediction plot at +40 timesteps 60

Figure 32: Prediction plot at +40 timesteps 61

Figure 33: Model accuracy across different prediction horizons 62

Figure 34: On the left, the tractor's trajectory; on the right, an on-site photograph showing the detection of each component. The tractor is standing on one corner of the field while the drone is on the ground. 62

Figure 35: On the left, the tractor's trajectory; on the right, an on-site photograph showing the detection of each component. The tractor's camera is exposed to the sun, and the drone is on its mission to maximize weed detection performance. 63

Figure 36: The left image shows a map of the tractor's operation from points A to E. Dark areas indicate sprayed regions, light grey areas indicate unsprayed areas, and red dots are the sprayed weeds. Dark areas without red dots represent safe-mode spraying, which is enabled when weed detection is unreliable. The right image shows example captures from locations 1–4 on the map. Sun creates flare artifacts on paths BC and DE (captures 2 and 4), unlike paths AB and CD (captures 1 and 3). 64

Figure 37: Map visualization of a tractor operation with reliable weed detection performance. The tractor does not enter safe mode. Unlike the operation shown in Figure 36, this one is performed on a different field, in a straight line. 65

Figure 38: The ML model, as the tractor operates (and velocity increases), predicts correctly that the engagement of the drone is not needed when the weed detection of the tractor is reliable. 65

Figure 39: The ML model correctly predicts the need for drone engagement in two situations: (a) when a flare initially introduces artifacts, and (b) several seconds before the tractor enters the sun-facing trajectory. 66

Figure 40: The left image shows a map of the tractor's and drone's operation from points A to E. Dark areas indicate sprayed regions, light grey areas indicate unsprayed areas to save herbicide, red dots are the sprayed weeds, and yellow dots are the drone-detected weeds. Dark areas without red dots represent safe-mode spraying, which is enabled when weed detection is unreliable. The right image shows capture from the drone and tractor at locations 1 & 2 on the map. At both sites, the tractor's cameras exhibit sun flare artifacts, whereas the drone's cameras do not. 66

Figure 41: Safe mode time (sec) in accordance with operation time (sec) for different methods: Always-on, Heuristic-based, ML-based, and No drone engagement. 67

Figure 42: Flight time (sec) in accordance with operation time (sec) for different drone engagement methods: a) Always-on, b) ML-based. The plot on the left corresponds to the test shown in Figure 40, and the plot on the right a distinct test where the ML-model deemed the drone unnecessary. 68

Figure 43: Latency in milliseconds vs different Istio modes. 81

Figure 44: Performance comparison of Machine Learning models for jamming attack detection. 83

Figure 45: Distribution of online detection timing, including Data Preprocessing, Inference, and Total Time. 83

Figure 46: Improvement of 10 ms in user experience. 85

Figure 47: Temporal distribution of keep-alive durations by CLEAR. Longer durations are favoured during low-carbon hours, reflecting carbon-aware adaptation. 88

List of Tables

Table 1: Overview of general integration status. 20
Table 2: Main HW parts of the smart lamppost node in the UBIW testbed. 30
Table 3: Data/metrics collected for MLSysOps in the smart city use case. 32
Table 4: Sensitivity Analysis Data Table 35
Table 5: Overall Operational performance 35
Table 6: Data/metrics collected for MLSysOps in the smart agriculture use case. 51
Table 7: Metrics accumulated up to each point of Figure 36, during tractor operation. 64
Table 8: Metrics accumulated up to each point of Figure 40, during tractor and drone operation. 66
Table 9 Memory footprint of MLSysOps agents across the Cloud–Edge continuum 94
Table – 10 Status of KPIs for Requirement Groups 104
Table 11 – Status of KPIs for Project Objectives 107
Table 12: Summary of Simulation Environments/Testbeds and responsible partners 111

Summary

This document details the progress in integrating and evaluating the MLSysOps framework, which is designed to manage distributed applications across the cloud-edge continuum using ML models. This framework leverages a three-level agent hierarchy (continuum, cluster, node) to handle application deployment and execution, telemetry, and system configuration decisions. The key functionalities that are currently integrated into the framework are briefly presented, along with the integration/validation status and limitations/issues that were faced during integration. A general approach for integrating MLSysOps that is independent of specific use cases is presented. This approach outlines seven steps aimed at streamlining the onboarding process for any application into the MLSysOps framework and has been applied to both use cases. The concrete adaptation/configuration of the framework is presented for the two application use cases. In the smart city scenario, the framework balances power consumption and detection accuracy by using sound sensors to trigger image processing for traffic incident detection. In the smart agriculture use case, the framework is used to enhance weed detection by dynamically deploying a drone when its input is expected to significantly improve accuracy. Moreover, the progress with respect to the concrete KPIs is presented, both for so-called requirement group (RG) KPIs and project-level (P) KPIs.

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ARM	Advanced RISC Machine
AWS	Amazon Web Services
CA	Certificate Authority
CPU	Central Processing Unit
CQI	Channel Quality Indicator
DLRM	Deep Learning Recommendation Model
DRL	Deep Reinforcement Learning
ENISA	European Union Agency for Cybersecurity
FPGA	Field Programmable Gate Arrays
FSM	Finite-State Machine
GB	Giga Byte
gNB	Next Generation Node B
GPS	Global Positioning System
GPU	Graphics Processing Unit
IETF	Internet Engineering Task Force
IMU	Inertial Measurement Unit
IoT	Internet of Things
IP	Internet Protocol
KPI	Key Performance Indicator
LIME	Model-Agnostic Explanations
LLM	Large Language Model

MCS	Modulation and Coding Scheme
MCU	Micro Controller Unit
ML	Machine Learning
NF	Network Function
NIST	National Institute of Standards and Technology
OCI	Open Container Initiative
PDR	Packet Delivery Ratio
QoE	Quality of Experience
QoS	Quality of Service
RFC	Request For Comment
RG	Requirement Group
RISC	Reduced Instruction Set Computer
RPi	Raspberry Pi
SHAP	SHapley Additive exPlanations
SITL	Software In The Loop
SLA	Service Level Agreement
TB	Tera Byte
UE	User Equipment
UPF	User Plane Function
VM	Virtual Machine
WP	Work Package

1 Introduction

Deliverable D5.3 details the progress in integrating and evaluating the MLSysOps framework, which is designed to manage and optimise distributed applications across the cloud-edge continuum. This document focuses on how the framework is applied to real-world use cases and assesses final progress against the key performance indicators (KPIs).

The MLSysOps framework uses a three-level agent system (continuum, cluster, and node) to deploy and execute containerised applications, manage resources, integrate telemetry, and adapt to dynamic conditions under the management of ML heuristics. These capabilities have been developed, tested, and evaluated in both virtual and physical testbeds. Two use cases demonstrate the framework's practical applications. The smart city use-case focuses on the energy-efficient management of smart lampposts, which utilize cameras and noise sensors to monitor urban activity. The goal is to reduce power consumption while maintaining detection accuracy. The smart agriculture use-case involves coordinating a tractor and a drone to enhance weed detection in farming, striking a balance between performance improvements and efficient energy use.

Before delving into the specifics of integrating MLSysOps into the two use cases, the general integration status of the framework is introduced in Section 2, summarizing the functionalities and pointing to known limitations/issues.

Section 3 presents the adaptation/configuration of the framework for the two application use cases. It begins with the generic stepwise approach used to facilitate the onboarding of any application into the MLSysOps framework, which comprises seven steps discussed in detail. This is followed by the development, integration, and results for each application's use-case. Final progress in the smart city use-case includes the successful deployment and operation of the MLSysOps framework across two distinct physical clusters: a private parking area at UBIW headquarters and public settings in Aveiro. Comprehensive data collection has been completed, resulting in a rich repository of noise and video telemetry used to train and evaluate an LSTM-based noise forecasting model. This model is now fully integrated into the agent hierarchy, enabling proactive power management of image processing components. In the smart agriculture use-case, the project has successfully demonstrated the collaborative operation of tractor and drone nodes in real-world field conditions. Over 30 GB of synchronized telemetry and video data were collected to train an XGBoost-based model for drone engagement. Evaluation results confirm that the ML-driven policy accurately predicts the need for drone assistance, reducing tractor "safe mode" operation, and resulting in a 12% increase in effective weed-spraying operation time while optimizing drone battery usage.

Further, Section 4 and Section 5 provide a detailed assessment of the KPIs at both the requirement group (RG) and project (P) levels, respectively, covering various aspects, including structured system descriptions, efficient application deployment across the continuum, hardware configuration, system performance, energy consumption, and trust. Each KPI is discussed separately, also giving its current achievement level using a simple scale (“o” means that the KPI has not been achieved yet; “+” means that the KPI has been partially achieved; “++” means that the KPI has been fully achieved). To have a more global overview, Section 6 summarizes the overall KPI achievement. The project has successfully concluded with most of its ambitious targets fully realized. Currently, 100% of RG-KPIs are completed or fully addressed, with 87% of those reaching full achievement (++). Similarly, 100% of P-KPIs have been addressed, with 80% confirmed as fully achieved (++). These results reflect the successful integration of the framework across the heterogeneous cloud-edge continuum, the validation of ML-driven autonomic operations, and the attainment of significant improvements in energy efficiency and system performance across all layers of the architecture.

Finally, Section 7 summarizes the overall status of the project from the KPI perspective.

2 General Integration Overview

2.1 Context

The aim of the MLSysOps architecture is to combine different mechanisms (for deployment, telemetry and application/system configuration) and ML models (that can help in taking good deployment and management/configuration decisions) into a framework that supports the flexible and adaptive deployment and management of distributed applications across the cloud-edge-IoT continuum for a given system infrastructure slice. Figure 1 provides a high-level view of the framework architecture.

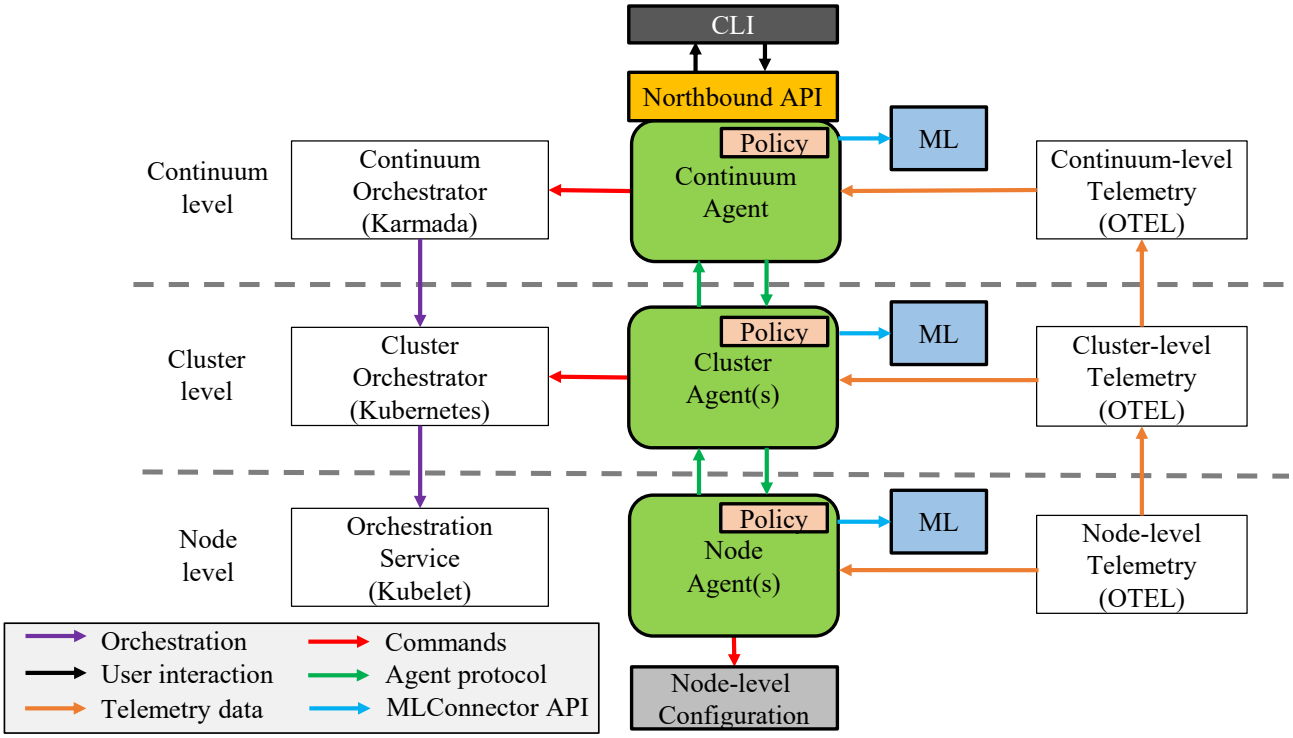

Figure 1: MLSysOps framework core architecture.

The backbone of the MLSysOps framework consists of a 3-level agent hierarchy that coordinates application deployment, system configuration, and telemetry mechanisms at each layer (continuum, cluster, node). The current implementation of MLSysOps relies on and extends as needed mature technology for the deployment and telemetry pillars (Karmada[1], Kubernetes[2], OpenTelemetry[3]). The agents embody the system's intelligence, connecting and interacting with these pillars at each layer as needed. More specifically, they capture and process telemetry to take system and application management and configuration decisions and then apply these decisions by invoking the deployment and configuration mechanisms through the southbound API. The logic of each agent for retrieving and processing telemetry information and taking management decisions can be flexibly configured through plug-in policies, which can be dynamically updated at runtime. Policies can use conventional heuristics or one or more ML models through the MLConnector API. Notably, the ML models

[1] Karmada, [Online]. Available: https://karmada.io/

[2] Kubernetes, [Online]. Available: https://kubernetes.io/

[3] OpenTelemetry [Online]. Available: https:/opentelemetry.io

themselves can be deployed via MLSysOps as a special type of application, exploiting computing resources that are part of the system slice.

The individual functionalities of the framework have been developed in WP3 and WP4 (documented in deliverables D3.3 and D4.4) and are incrementally integrated in WP5. This section provides a brief description of the functionalities that have been successfully integrated into the MLSysOps framework, along with an overview of the general integration status, highlighting limitations and potential extensions. Note that since June 2025, the MLSysOps framework and its various components have been released as open source[4].

We note that this work is referred to as "general integration" because it is not directly related to the two application use cases of the MLSysOps project (smart city and smart agriculture). The application-specific adaptation/configuration of the MLSysOps framework for each application use case is described in Section 3.

2.2 Key Functionalities of the Integrated MLSysOps Framework

The key functionalities supported in the integrated version of the MLSysOps framework are summarized below. The overall integration status for each functionality is provided in Section 2.3.

2.2.1 Application and System Infrastructure Descriptions

The application designer uses a suitable declarative description to capture the application's key deployment and orchestration requirements, resource needs, configuration options, and performance targets. A corresponding structured description also captures the nodes and resources available as part of the system slice managed by MLSysOps. These descriptions, along with telemetry information, guide the deployment and adaptation of applications in the continuum. Note that these descriptions also capture so-called Far-Edge nodes, which have significant resource constraints and do not support a proper Operating System (OS) or container runtime, allowing such nodes to be part of a system slice and be used as hosts for deploying and running (lightweight) application components.

2.2.2 Agent Hierarchy

Agents can be introduced at different layers of the continuum hierarchy, e.g., at the continuum level, the cluster level, and the node level. The agent hierarchy is involved in the initial application deployment and configuration, as well as in the ongoing monitoring and adaptation of the application. The continuum agent processes the application description and forwards it (or selected parts thereof) to individual cluster agents, which, in turn, proceed with deploying application components on the cluster nodes. Agents also consume telemetry information, use it to make application and system adaptation and reconfiguration decisions, and apply these decisions via the respective mechanisms. The current agent implementation follows a modular design, consisting of a core Python module that implements functionality shared across all agents, while each layer extends it with its own logic and tools. SPADE[6] is used for inter-agent coordination, whereas the Fluidity framework[5] (implemented in the context of the MLSysOps project) serves as the Kubernetes controller. Agents can be programmed to make decisions based on conventional heuristics or ML models, which can be engaged dynamically via the MLConnector API. Such logic can be provided in a flexible way, in the form of plug-in policies, which can be dynamically updated at runtime.

[4] MLSysOps open-source repositories: https://github.com/mlsysops-eu

[5] Fluidity framework: https://doi.org/10.1016/j.future.2024.03.031

2.2.3 Application Deployment

The components of the distributed application are deployed on nodes, based on their requirements and the available resources. The control flow of deployment is top-down, driven by the agent hierarchy, starting from the continuum level, moving through the cluster level, and ending at the node level, where components are effectively executed. The implementation uses Karmada for the continuum-level deployment and Kubernetes K8S/K3S[6] for the cluster- and node-level. Deployment is supported for various types of nodes, ranging from VMs running in datacentres and powerful workstations to standalone Smart-Edge (RPi, Jetson, ZCU102 FPGA SoC) nodes and highly resource-constrained Far-Edge nodes (Kallisto platform using an ARM-M4 microcontroller) through a special gateway.

2.2.4 Application Component Versions and Execution Enclaves

The code of application components is packaged in the form of containers. The same application component can have different implementations (container images) that target different platforms and/or provide different performance, resource usage, and accuracy trade-offs. In addition, application components can be prepared for execution in different execution enclaves (plain containers[7], sandboxed containers in micro-VMs[8], unikernels[9]). Deployment on Far-Edge nodes is supported via special container images and execution enclaves, in conjunction with a gateway environment running on a powerful edge node. This environment hosts virtual Kubelets and an MQTT broker, supporting code deployment and communication with the Far-Edge nodes.

2.2.5 Relocation of Application Components

Application components can be relocated from their current host to run on a different node. Component relocation is achieved by deploying a new instance of the application component on the target node, followed by the removal of the old instance. Relocation is supported for stateless application components, which will initialise and start running from scratch on the new host.

2.2.6 Application-level Communication and Traffic Redirection

Application components may interact with each other by exchanging data/information or via client-server invocations over IP. An application component may implement the desired interaction through one or more so-called services, in which case the respective contact information (Kubernetes-provided Virtual IP address) remains valid even if the component is relocated to a different node. The default network interface used for the interaction between application components (data plane) is also used for deployment control and telemetry (control plane). It is, however, possible to redirect application-level traffic over other network interfaces, e.g., to allow application components residing on nodes that have a Wi-Fi interface to communicate directly over such a link.

2.2.7 Telemetry

In tandem with the top-down deployment pillar (Karmada-Kubernetes-Kubelet), a reverse bottom-up telemetry pillar is implemented using OpenTelemetry (OTEL). This is used to capture system-level metrics and make this

[6] K3s, [Online]. Available: https://k3s.io/

[6] SPADE [Online]. Available: https://spade-mas.readthedocs.io/en/latest/readme.html

[7] https://github.com/opencontainers/runc

[8] https://katacontainers.io

[9] https://github.com/nubificus/urunc

information available at different levels of the agent hierarchy as needed to enable required node/resource monitoring. It is important to note that the same telemetry pipeline can be used to capture and emit application-level metrics from within application components running on the nodes through a generic/flexible API that connects to local (node-level) OTEL. Moreover, the telemetry system can be leveraged by other mechanisms (e.g., anomaly detection and trust score adaptation) to share status information (e.g., trust score updates) with the agents, allowing these updates to be considered when making management/configuration decisions.

2.2.8 ML Model Discovery, Activation, Invocation, and Deactivation

As mentioned above, agents can use ML models to make different system and application management decisions. The ML model lifecycle is supported via the MLConnector API. This can be used to search and find models, request the activation of an ML model, invoke the ML model, and request its deactivation. One can search for ML models based on simple features and elaborate search expressions based on tags (any number of tags can be added to the model during registration). ML model activation is implemented by leveraging existing deployment support, which involves submitting a special/custom application description linked to a suitably prepared container, and then following the usual deployment path. Once an ML model has been activated, an endpoint is returned, allowing it to be invoked by the agent that requested the activation. The ML Connector can be dynamically configured to provide an explanation for each inference made. Even when this is disabled, the MLConnector still saves explanations for every inference call made, so that these can be retrieved/played back at a later point in time. Furthermore, the MLConnector can be configured to perform model retraining automatically when large drifts are detected, or it can be triggered manually. Note that a model can be updated at runtime in a transparent way without invalidating the endpoint used to invoke it. Finally, when the client agent no longer wishes to use the ML model, it requests that it be deactivated, leading to the deallocation of the respective system resources.

2.2.9 Flexible Exploitation of Accelerators for Compute-intensive Operations

The flexible and transparent exploitation of the various types of accelerators available on a node is achieved through the vAccel framework. Application components that perform heavyweight processing operations can abstract those in the form of function calls. Instead of statically linking these calls to a fixed implementation, this is decided at runtime based on corresponding configuration settings. Depending on the type of processing, the same function can have different implementations, e.g., for multi-core CPUs, GPUs, and FPGAs.

2.2.10 Node-level settings

It is possible to dynamically set the frequency of the node's CPU/GPU. Higher frequency settings speed up computations at the cost of higher power consumption, while lower frequency settings slow down computations but also reduce the power consumption of the node. For Far-Edge nodes that communicate with the gateway over wireless, it is possible to switch between different transmission power levels, which is a major factor of energy consumption.

2.3 Integration Status

The integrated MLSysOps framework has been developed and tested through an intensive collaborative effort among various partners, who provided mechanisms, research testbeds, and nodes, enabling the validation of core system functionalities across a multitude of configurations. Table 1 provides an overview of the integration status for each of the above core functionalities, stating how each one was validated to confirm proper operation and any known limitations or issues. The table also lists the main partners responsible for developing each functionality and the testbeds (underlined partner names) used to confirm proper operation.

Table 1: Overview of general integration status.

Functional Aspect	Integration Status / Validation	Limitations / Possible Extensions	Testbeds	Partners
Application and system descriptions	Successful parsing of application descriptions for different applications with multiple independently deployable components. Extensions to support adaptation knobs have been added, while introducing permitted actions that MLSysOps agents can take in the MLSysOps application and Node descriptions.	No issues.	TB-R-5	UTH, UNICAL
Agent hierarchy	Complete agent hierarchy working, supporting the deployment and monitoring of applications. The Continuum agent runs in a VM, cluster agents run in separate VMs, and node agents run on VMs and physical nodes (Intel workstation, Jetson, Raspberry Pi). Agents that handle resource-constrained Far-Edge devices are co-hosted on available, more powerful nodes. Agent policies can be changed at runtime via the CLI.	No issues.	TB-R-5 TB-R-7	UTH, UNICAL, FhP
Application deployment	Successful deployment of an application with components running on a VM in a cluster, on a standalone Intel workstation, on different Smart-Edge nodes (RPi and Jetson), and on Far-Edge nodes (Kallisto).	No issues.	TB-R-5 TB-R-6 TB-R-7	UTH, NUBIS, FhP
Application component versions and execution enclaves	Successful deployment of an application component (image capture) with different versions targeting a powerful standalone node (Intel workstation) and a Smart-Edge node (RPi). Successful deployment of an application component (OpenCV Optical Flow processing) as a plain container and as a sandboxed container in a micro VM, taking advantage of hardware acceleration functionality through vAccel.	No issues.	TB-R-6	NUBIS, UTH
Relocation of application components	Successful relocation of application components within and across different clusters, between powerful nodes (VMs, workstations), Smart-Edge nodes (RPi, Jetson, ZCU102 FPGA SoC), and Far-Edge nodes (Kallisto).	No issues.	TB-R-5 TB-R-6 TB-R-7	UTH, NUBIS, FhP
Application-level communication and traffic redirection	Successful communication between components over the default path (used for the control plane). Successful application traffic redirection between different network interfaces (Ethernet, 4G, Wi-Fi).	Very frequent component relocation may disrupt communication, as it requires Kubernetes to update service routing internally. There may also be minor/temporary	TB-R-5 TB-R-6 TB-R-7	UTH, NUBIS, FhP

Functional Aspect	Integration Status / Validation	Limitations / Possible Extensions	Testbeds	Partners
		disruptions in the event of traffic redirection. Cross-cluster communication between application components is supported via Karmada/Submariner but requires extensive manual configuration. Application components on Far-Edge nodes communicate with other components via MQTT. Another option would be to support communication via COAP.		
Telemetry	Complete pipeline working for the Far-Edge, Smart-Edge, cluster, and continuum level. Successful generation, reception, and propagation of system-level telemetry information. Successful generation (through the MLSysOps telemetry API), reception, and propagation of application-level telemetry. Agents can flexibly configure, also at runtime, the telemetry they wish to consume.	No issues.	TB-R-5 TB-R-6 TB-R-7	UTH, NUBIS, FhP
ML model storage, discovery, activation, invocation, deactivation, and retraining	Successful discovery, activation (via the standard deployment path), invocation, and deactivation by agents via the MLConnector API. ML models can be deployed both locally (on the same node as the requesting agent) and remotely (on a node selected by MLSysOps). Drift detection and retraining are transparently performed via the MLConnector without invalidating the invocation endpoint. The images for the ML models are stored using the CC storage service.	No issues.	TB-R-5	UCD, CC, UTH
Flexible exploitation of accelerators for compute-intensive operations	Implementations of image processing functions for multi-core CPU, GPU, and FPGA. Successful runtime selection of function implementation on two nodes (Intel workstation, Jetson, ZCU102 FPGA SoC).	The desired vAccel plugins (daemon sets) must be pre-installed on the nodes that will host application components, depending on the	TB-R-5 TB-R-6	NUBIS, UTH

Functional Aspect	Integration Status / Validation	Limitations / Possible Extensions	Testbeds	Partners
		node's hardware (accelerator type, framework type, etc.). This is part of the infrastructure/framework configuration process and must be completed before running applications. This is currently done manually, but it could be automated by extending the framework deployment descriptions/instructions.		
Node CPU/GPU frequency setting	Successfully set CPU/GPU frequency on different physical nodes (Intel workstation, Jetson, RPi).	No issues.	TB-R-5 TB-R-6	UTH, NUBIS
Node transmission power setting	Successfully set the transmission power for the Wi-Fi radio on Kallisto nodes.	No issues.	TB-R-7	FhP
Node trust level adaptation	ML-based mechanism monitors CPU/memory resource usage, detects abnormal behaviour, and updates the trust level of the node, which is communicated to the cluster level via telemetry.	Supported/tested on RPi nodes, but could be ported to other platforms.	TB-R-5 TB-R-8	TUD, UTH
Storage gateway deployment	The gateway component to the CC storage service can be deployed and managed as a regular application component. The requirement to deploy the gateway before the components that will use it can be captured in the corresponding MLSysOps application description.	It is possible to deploy a single gateway that is accessible by different applications, but in this case, the deployment dependency must be ensured manually.	TB-R-5	CC, UTH

Overall, a wide range of functional aspects/mechanisms developed in the MLSysOps project are successfully integrated into the framework and work smoothly/robustly for different application components and combinations of physical nodes. Specific functionalities still have some limitations, mainly because the respective issues were not a high priority for the project and the application use cases. Given that the MLSysOps framework is open source, these limitations can be investigated and resolved by future contributors.

3 Use-case Specific Integration

This section provides the integration status for each of the MLSysOps project's two application use cases. First, we explain the process for supporting each use case using a suitably adapted instance of the MLSysOps framework. Then, we discuss the status of each use case separately.

3.1 Application-driven Framework Adaptation/Integration/Testing

The MLSysOps framework is designed to capture a wide range of system and application management scenarios. However, a given use case may not require all the framework's functionalities. Some functionalities may not be applicable due to constraints on the physical nodes, restrictions on accessing the system infrastructure, and the degree to which MLSysOps is allowed to perform system-level configuration.

The adaptation/configuration, integration, and testing of the MLSysOps framework for a given use case follows a structured approach, illustrated in Figure 2.

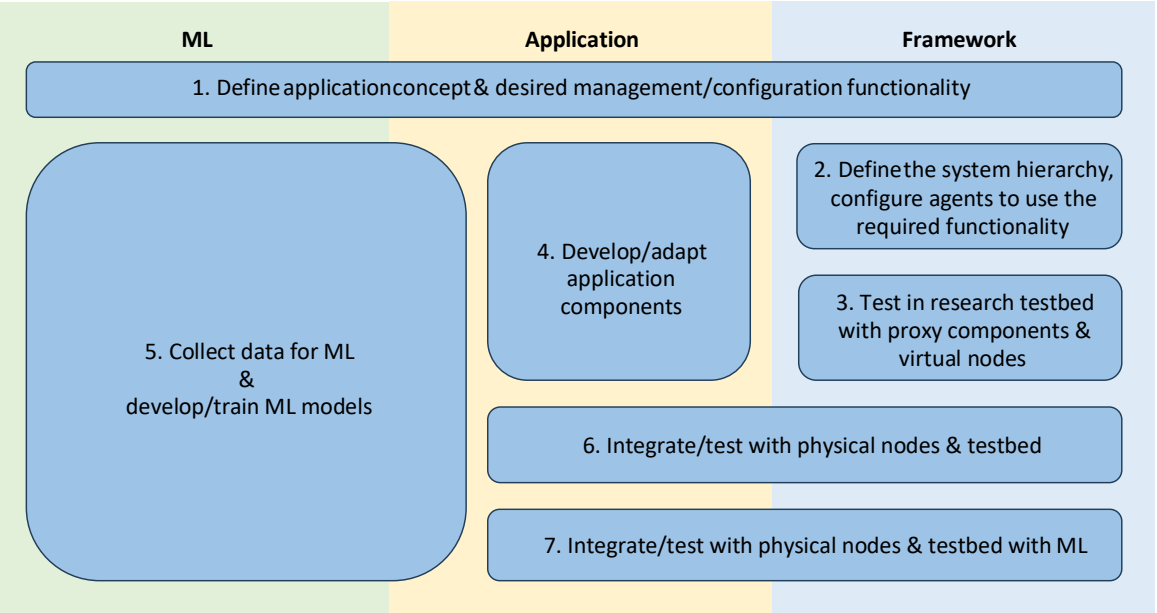

Figure 2: Approach and integration of the MLSysOps framework to a specific application use case.

The process consists of different tasks/activities taking place in three contexts with a focus on (i) the application, (ii) the framework, and (iii) the ML models. Note that most activities are loosely coupled and can run in parallel to a significant degree. Next, we briefly describe each activity following the enumeration in Figure 2.

1. Define the application use case, the nodes involved, the application components, the management responsibility of the MLSysOps framework, and the functionalities required to achieve this. At this stage, also define the application-level performance metrics (if any) and the respective “satisfaction” thresholds.
2. Decide the hierarchical structure for managing the system and configure the agents at each layer to use the required telemetry and mechanisms. At this stage, where ML models may not be available or sufficiently trained, decisions are taken using simple/conventional rules and heuristics. This makes it possible to anticipate the resulting behaviour. It is important to detect issues/bugs in the underlying mechanisms or the conceptual control loop before attempting to employ ML and real nodes in the

application testbeds. Note that these conventional heuristics also serve as a fall-back option in case an ML-based operation is decided to stop.

3. The resulting framework is tested in a research testbed using proxy application components and virtual nodes. This enables the systematic testing of the basic application management loop, including the corresponding control and telemetry flows, early on without requiring access to the physical nodes or the target testbed, not even to the real components of the application that will run in the testbed.
4. Develop/adapt the application components so that they can be deployed and orchestrated by the MLSysOps framework. The application components must also be programmed to emit the specified performance metrics.
5. Collect data to design and train the ML models that will drive system/application management. This is done, in part, in parallel to the development of the application components
6. Integrate and test the MLSysOps framework using the nodes of the physical testbed. At this stage, one confirms the basic management loop, as outlined in #3. However, this is done using the real application components and the real nodes of the application testbed. Note that no ML models need to be used at this stage. ML-driven policies should be applied only once basic management works robustly, based on simple/conventional rules/heuristics that result in predictable behaviour.
7. The developed ML models are plugged into the MLSysOps framework so that they can be used by the MLSysOps agents to make management/configuration decisions (vs. using the simple/conventional rules and heuristics of #3). The basic management logic of #3 is maintained as a fallback whenever the system is commanded to pause ML-based operation.

In the next sections, we explain how the framework was configured for the smart city and smart agriculture application use cases, present the corresponding data collection, ML development, and evaluation, and discuss the aspect of scalability for each use case.

3.2 Smart City Application

With respect to the framework adaptation/configuration and integration approach shown in Figure 2. In the second year of the project, the smart city use-case completed tasks #1, #2, and #3. In the third year of the project, work was done on tasks #4, #5, #6, and #7. In the following, we provide a summary of the application concept, explain how the MLSysOps framework supports the desired system/application management, discuss the development/integration work for this use case, and present the evaluation results.

3.2.1 Concept, Role of MLSysOps, Application KPI & System Optimization Objective

The goal of the smart city application use case is to improve the energy efficiency of smart lampposts, which are equipped with cameras and run image processing software. The application's objective is to detect/count people and cars passing by, as well as potentially identify problematic situations/incidents, and send this information to the city's datacentre or private cloud. Ultimately, this information can be used for various purposes, including producing traffic statistics, detecting dangerous driving behaviours, alerting authorities to hazardous situations such as vehicle collisions, and decreasing the response time for first responders, including emergency personnel and police forces, to road accidents (potentially saving lives).

The role of MLSysOps is to reduce the power consumption of the smart lampposts by completely deactivating image processing during periods when no detections occur. However, there is an inherent trade-off between energy saving and the detection accuracy/performance of the application. The longer the image processing remains fully deactivated, the more energy is saved; however, the likelihood of missing events of interest increases.

To reduce the probability of missed events during the periods when image processing is deactivated, the lamppost setup is augmented with the addition of a noise sensor. The assumption is that people/cars will create some noise that can be detected before they pass in front of the camera, which can serve as a trigger to re-activate image processing in time for the respective event to be detected. In other words, the noise levels are used as a hint that object detections are likely to occur soon. Since the activation of the image processing pipeline/component may take some time to start working properly, it is essential to be able to predict the noise level based on previous measurements to trigger image processing ideally in advance.

The detection performance of the smart lamppost is captured through suitable application-level metrics with specific satisfaction thresholds, as declared in the corresponding application description. At runtime, the application reports the current noise level, noise events, and image processing detections by emitting the respective values via telemetry, allowing this information to be consumed by the MLSysOps agent(s). From the high-level application perspective, the KPI is to keep the number of lost detections due to the deactivation of the image processing component below 5%. From a system perspective, the KPI is to minimize the energy spent by the lamppost while meeting the application-level target, which is a low number of lost detections.

3.2.2 Application Structure

The application consists of three components: (i) the noise processing component, (ii) the image processing component, and (iii) the telemetry/event forwarder. The first component accesses the noise sensor and emits the respective readings along with noise events that imply objects passing by in front of the camera with high probability. The second component processes images from the camera to detect objects/incidents and emits respective detection events. The third component receives and stores information generated from both components, allowing it to be used for further analysis, such as producing traffic statistics for a specific street, area, or entire city. It also utilizes the noise and detection events generated by the other two components to calculate a metric for the estimated lost detections resulting from inactive image processing. Finally, it forwards all metrics to MLSysOps telemetry, allowing them to be consumed by the agent. To focus on the essence of the use case, only the image processing component is managed by MLSysOps; the other two components are assumed to be pre-deployed on the node.

```
MLSysOpsApplication:
  name: ubiwhere-app
  cluster_placement:
    cluster_id:
    - "UTH-UBIW1"
  components:
    - metadata:
      name: noise-detection-app
      external_component: True
      restart_policy: OnFailure
      containers:
        - image: UBIW_noise_detection_app:latest
          image_pull_policy: IfNotPresent
      qos_metrics:
        - application_metric_id: NoiseEvents
    - metadata:
      name: object-detection-app
      restart_policy: OnFailure
      containers:
        - image: UBIW_object_detection_app:latest
```

```
          image_pull_policy: IfNotPresent
      qos_metrics:
        - application_metric_id: DetectionEvents
- metadata:
      name: forwarder-app
      external_component: True
      restart_policy: OnFailure
      containers:
        - image: UBIW_forwarder_app:latest
          image_pull_policy: IfNotPresent
      qos_metrics:
        - application_metric_id: EstimatedLostDetections
          target: 5
          relation: lower_or_equal
global_satisfaction:
    threshold: 0.9
    relation: greater_or_equal
    achievement_weights:
    - metric_id: EstimatedLostDetections
      weight: 1
```

Figure 3: MLSysOps application description for the smart city use case.

The core deployment description for this application based on the MLSysOps format is shown in Figure 3. For the sake of the example, we assume the application should be deployed in a specific cluster, representing a target area, e.g., the real-world testbed of UBIW in Aveiro. The placement requirement for the image processing component that is managed via MLSysOps is specified through corresponding node labels and/or continuum layer filtering options (it must reside on the same node as the noise component), which are matched with the respective cluster/node properties (defined in the system infrastructure description). As mentioned above, the noise and image processing components generate respective "NoiseEvents" and "DetectionEvents". Based on this information, the forwarder generates the "EstimatedLostDetections" metric with the respective target value being less than or equal to 5 (out of 100), which exclusively contributes to the global satisfaction level. Note, however, that this metric is merely an approximation for the detections lost due to image processing being inactive, assuming each noise event corresponds to a detection event. While this estimation can be inaccurate, it serves as a rough proxy for the golden truth (i.e., how many detections would have occurred if image processing were active), which is unknown to the application or system at runtime.

3.2.3 Development of Application Components

Image processing. The image processing component gathers the video stream from a camera via an RTSP endpoint. Then, it feeds the camera frames to a YOLO computer vision algorithm that detects cars, people, and other objects commonly found on the streets. While this was initially thought to be deployed via Docker, it was decided to alter the approach and deploy the application using Kubernetes with a Helm chart. A few considerations were also made due to the CPU architecture of the edge nodes (Nvidia Jetsons) and the support of CUDA, enabling these edge nodes to perform to the best of their ability. This resulted in a few changes to support the Arm CPU architecture and a proper Docker base image that enables the Jetsons iGPU to process the image. Crucially, the component now includes a detection delta logic: instead of simply broadcasting raw counters, it calculates the incremental change in detected objects between frames (e.g., car_delta, moto_delta). This processed metadata is pushed to the local MQTT broker and subsequently forwarded to the MLSysOps telemetry pipeline.

Sound processing. The sound processing component interfaces directly with the ESP32 noise sensor, which is hosted within the smart lamppost. Beyond simply reading raw data, this component has been enhanced to perform local pre-processing: it analyses the audio stream to identify potential Noise Events—acoustic signatures that exceed the dynamic background threshold. These derived events, along with the raw noise levels, are broadcast via MQTT, serving as the primary trigger signal for the MLSysOps prediction logic.

Event collector. A dedicated telemetry forwarding component bridges the gap between the edge sensing layer and the MLSysOps framework. This component subscribes to the local MQTT topics populated by the Image and Sound processing units, aggregating the asynchronous data streams. Its primary function is to normalize these heterogeneous inputs into a standardized format and forward the synchronized metrics—including Detection Deltas and Noise Events—directly to the MLSysOps observability stack via OpenTelemetry (OTEL). By decoupling the core sensing logic from the telemetry transmission, this component ensures robust data buffering and enables the generation of high-level composite metrics required for system evaluation.

3.2.4 Framework Instantiation and Agent Logic

The smart lamppost node is part of a cluster where the application is deployed. Many such nodes can be managed within the same cluster or multiple clusters for different areas of the city. A continuum-level agent manages the system slice assigned to MLSysOps, while a corresponding cluster agent manages each cluster within that slice. Finally, each node runs a node-level agent.

Each smart lamppost is assumed to operate independently, so we focus on controlling a single lamppost within a cluster. More advanced scenarios for coordinated resource sharing among multiple smart lampposts could be explored in a subsequent phase. Figure 4 illustrates the system setup, core framework components, and application deployment on the lamppost node.

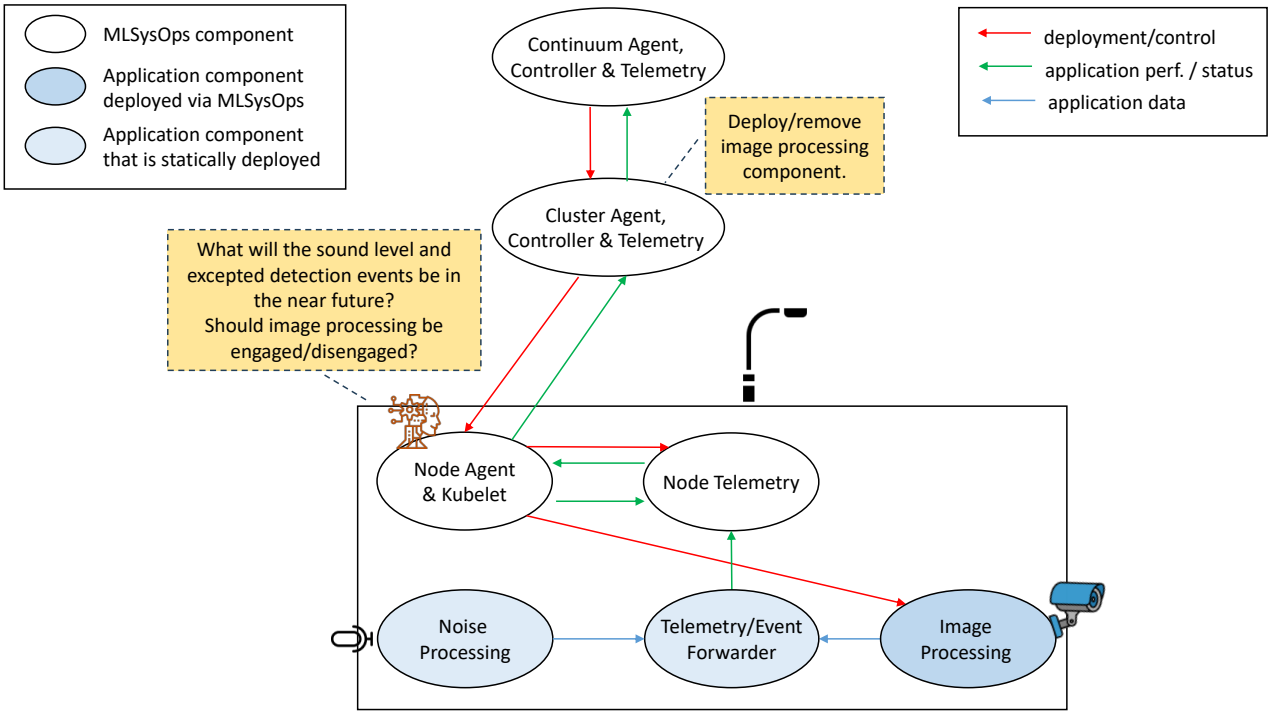

Figure 4: MLSysOps framework for the smart city use case.

System/application management is distributed between the node and cluster levels. The node agent retrieves information generated by the noise and image processing components from telemetry and uses this information to predict the presence of noise levels and detection events for the next few minutes. The prediction, along with

a hint to activate/deactivate image processing, is propagated to the cluster level through telemetry. The cluster agent monitors the state of the nodes and the prediction of detection events and activation/deactivation hint, and it uses this information to control the activation and deactivation of the image-processing component on the lamppost.

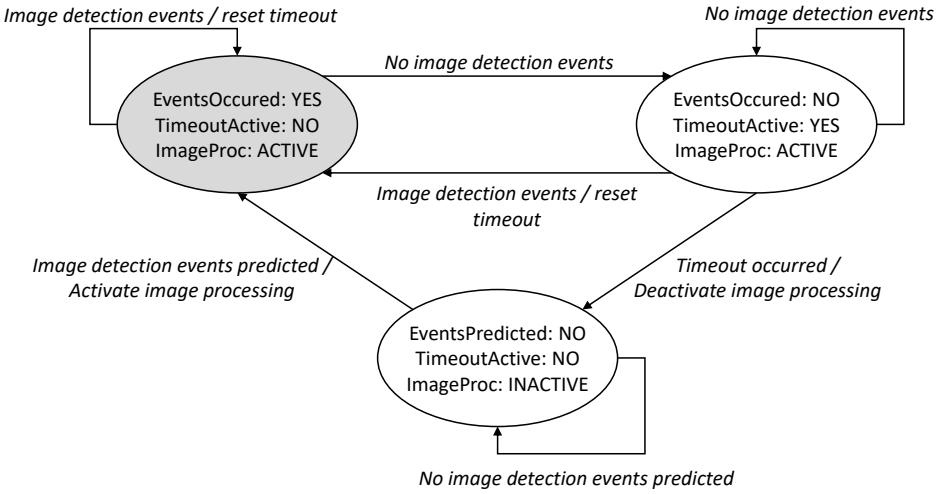

Figure 5: FSM used by the cluster MLSysOps agent to control the deactivation/reactivation of the image processing component.

The cluster agent makes decisions based on a finite-state machine (FSM), as shown in Figure 5. In summary, the image processing component is deactivated when no detection events occur, and it is also predicted that this will not be the case in the near future, and an activation timeout expires; the role of the timeout is to keep image processing active despite the predicted absence of noise and image processing detection events, thereby avoiding frequent oscillations between deactivation and activation. Image processing is reactivated when it is predicted to lead to detection events.

For simplicity, the above FSM assumes that the lamppost node is available. The actual (implemented) FSM is more complex, handling the case where the lamppost becomes available after being temporarily unavailable. In this scenario, both the sound processing component and the image processing component are deployed to establish the default configuration.

3.2.5 Integration and Testing

3.2.5.1 Using Virtual Nodes

The above system setup was initially tested using UTH's research testbed. Instead of the real smart lamppost node, we use a virtual machine (VM) running in the UTH cluster. Using this setup, exhaustive tests were conducted to verify the correct operation of all control/telemetry flows, the agent logic, and the functionality of the deployment mechanisms of the MLSysOps framework. Since we want to stress test the MLSysOps framework, instead of using the real application components of UBIW, we use proxy components that emit similar metrics. These components are programmed to generate these metrics in a controlled manner, triggering the deactivation and reactivation of image processing according to the desired test scenarios. Thanks to this setup, numerous tests were performed to verify the robustness of MLSysOps software before trying to use it with the real smart lampposts.

Notably, this setup can be used in conjunction with the smart lamppost node(s) of the real-world testbed to explore scenarios and evaluate system-level functionality across multiple nodes. However, large-scale evaluation scenarios involving many nodes will have to be conducted using simulations.

3.2.5.2 Using Physical Nodes

Tests using physical nodes were constructed using two clusters of worker nodes: cluster 1, encompassing the nodes installed at UBIW Headquarters, and cluster 2, comprising physical nodes that capture information in a public setting in the city of Aveiro.

Private Testbed (Cluster 1)

(a) **(b)**

Figure 6: (a) Satellite view of UBIW testbed, and (b) Smart lamppost, Node #0. #1 and #2

This testbed features three smart lampposts, indicated by the yellow pins (#0, #1, and #2) in Figure 6 (a). These lampposts are placed at the boundary of UBIW's private parking area. They run the application software to detect and count mobility-related events, such as the number of cars, motorcycles, bicycles, and people entering/leaving the parking lot. This environment represents a controlled "low-noise" scenario ideal for calibrating energy-saving logic.

All nodes are capturing real street footage. Basic functionality has been validated based on (i) successful deployment and dynamic management according to Kubernetes status information and MLSysOps agent decisions, (ii) application-level information collected via telemetry (observing detection counters and noise level patterns, verifying their behaviour aligns with expected patterns), and (iii) system-level metrics collected via telemetry (monitoring CPU, GPU, and memory usage across operational cycles).

Public Testbed (Cluster 2)

The second testbed extends the evaluation to a live urban environment. It comprises two edge nodes (Nodes #3 and #4) deployed in the city of Aveiro. Unlike the private cluster, these nodes monitor a public street with continuous vehicular and pedestrian traffic. The sensors here are mounted at a higher elevation, creating a

"continuous-noise, high-density" scenario. This setup is critical for stress-testing the system's logic and validating the adaptive normalization algorithms against the constant acoustic hum of the city.

All nodes are capturing real street footage. Basic functionality has been validated based on (i) successful deployment and dynamic management according to Kubernetes status information and MLSysOps agent decisions, (ii) application-level information collected via telemetry (observing detection counters and noise level patterns, verifying their behaviour aligns with expected patterns), and (iii) system-level metrics collected via telemetry (monitoring CPU, GPU, and memory usage across operational cycles).

Hardware Integration

All smart lampposts across both testbeds are equipped with identical MLSysOps-ready hardware. To this end, additional equipment was acquired in 2024 and subsequently configured before being mounted. Figure 7 shows the test setup used for the Smart-Edge node in the lab before installation.

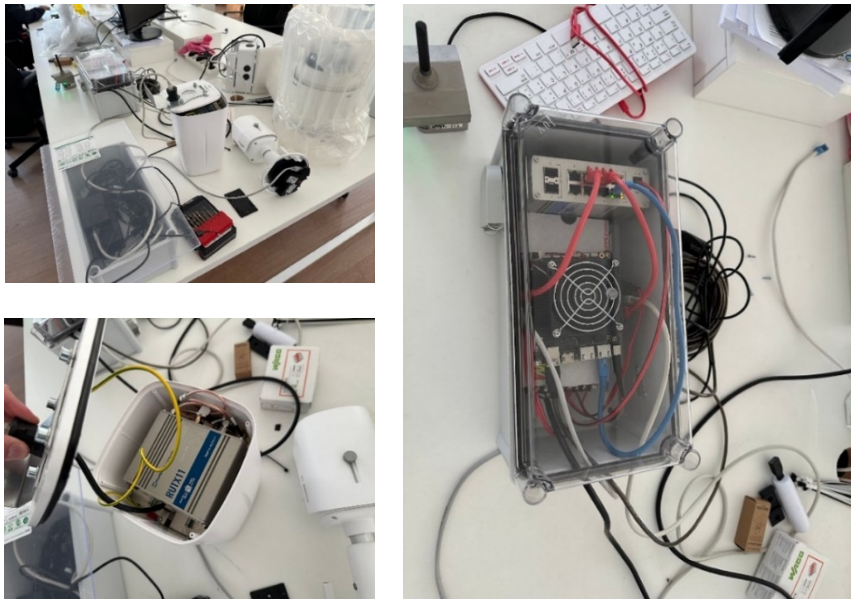

Figure 7: Housing and lab setup for the Smart-Edge node before it is mounted on the lamppost.

The basic hardware (HW) parts of the Smart-Edge node are listed in Table 2. All the smart nodes have similar configurations. It is essential to note that the node setup is extensible, allowing for the inclusion of additional peripherals.

Table 2: Main HW parts of the smart lamppost node in the UBIW testbed.

Equipment type	Hardware
Video Camera	Hikvision h.265+ EXIR VF Bullet Network Camera
Compute Unit	Nvidia Jetson Orin AGX 32GB or 64GB
Noise Sensor	ESP32 & Grove LM2904
Switch	Comparta CPGU-0500 or Teltonika TSW200

Smart Relay	Shelly Pro 2 PM
Power Supply	Mean Well RD-125-4812

Infrastructure

The Smart-Edge nodes on the lampposts are connected to a K3S cluster, which, in turn is connected to a system slice managed by Karmada. The system slice is managed both at the continuum level and the cluster level by MLSysOps. The cluster is configured to manage a mixed infrastructure consisting of physical Smart-Edge nodes and several virtual machines (VMs) hosted in Proxmox (virtualization platform) within the Barrocas datacentre (private UBIW datacentre).

This architecture was deliberately designed to create a secure abstraction layer between the physical infrastructure and the MLSysOps framework. Specifically, the setup instantiates a Virtual Machine (VM) for each physical smart lamppost node. These VMs function as exact Digital Twins of the physical hardware within the secure DMZ environment. The communication pipeline is established as follows:

1. Real-Time Telemetry: The physical nodes continuously transmit real-world sensor data (noise levels, video metadata, power consumption) to their corresponding VMs via a secure MQTT bridge.
2. Proxy Execution: The MLSysOps framework and agents are deployed on these VMs. They process the incoming telemetry as if they were running directly on the edge device.
3. Control Propagation: Decisions made by the MLSysOps agents (such as deactivating the image processing application) are propagated back to the physical node for execution.

Critically, this configuration protects UBIW's core physical infrastructure from experimental software updates while enabling comprehensive testing and deployment of MLSysOps without operational barriers. By isolating the MLSysOps-managed system slice, the framework can operate in a production environment with real physical nodes without compromising existing services. Because the VMs receive the exact live telemetry stream with negligible latency, the input data for the ML models and the system state monitored by the agents are identical to what would be observed in a direct deployment. The testbed effectively operates as a high-fidelity Digital Twin, ensuring that all observed node metrics, energy savings, and detection rates are representative of real-world performance.

Currently, all physical nodes are fully functional and transmitting telemetry data to the designated VMs. At each VM worker node, an MLSysOps agent runs locally, retrieving system and application metrics through a suitably configured telemetry pipeline. These node-level agents collect comprehensive data, including detection events from the computer vision components and noise levels from the sound sensors. The agents process this information locally and forward relevant telemetry to the cluster-level MLSysOps agent, which aggregates data from all nodes and uses it to make informed management decisions. Notably, MLSysOps now controls the activation and deactivation of the image processing components on the nodes, dynamically managing their operation based on the prediction of detection events (informed by both visual detections and noise patterns) and system performance metrics.

3.2.6 Data Collection

The data collected using the above setup via telemetry includes system-level metrics (GPU load, CPU load, memory usage) and application-level metrics (inference speed, tracking speed, number and type of detections). Table 3 gives an overview of the collected data.

Table 3: Data/metrics collected for MLSysOps in the smart city use case.

Source	Metric	Unit	Description
System	CPU Load	Percentage (%)	Instant CPU usage.
	GPU Load	Percentage (%)	Instant GPU usage.
	Memory Usage	Percentage (%)	Instant memory usage.
Application	Inference Speed	Milliseconds (ms)	The time it takes for the application to process one frame and detect objects.
	Tracking Speed	Milliseconds (ms)	The time it takes for the application to track the detected objects within a frame (vs previous frames).
	Detection Event	class (string); count (integer)	The application detected a new event.
Sensor	Noise	Relative Intensity (0-100)	Ambient sound intensity

3.2.7 Machine Learning Training and Evaluation

Considering the need to predict the need for running the activity detection model, which, by itself, requires time to initialise and process the input frames and detect activity, we introduce the use of a Long Short-Term Memory (LSTM) network. The fundamental motivation for using an LSTM network comes from the challenge of processing sequential data where the context needed to make a correct prediction is separated by a significant gap in time.

This model is designed to forecast future noise levels by analysing a single stream of past noise measurements. It processes one noise reading at a time using an architecture built from three stacked layers, which allows the system to capture both immediate fluctuations and longer-term noise trends. Internally, the model expands the single input into sixteen distinct features, giving it the capacity to recognize complex sound patterns. To ensure the system remains robust and does not merely memorize the training data, it randomly ignores twenty percent of its internal connections during the learning process. Finally, it condenses this memory down to a single value representing the predicted noise intensity.

We complement the designed LSTM model that is developed to predict future noise values with conditional logic. Specifically, we compare the average noise value of the *input* window (noise values from the past timestamps fed into the model) and the *output* window (predicted noise values of some future timestamps returned by the model), and check if the difference crosses a predefined *threshold*. If we register such a significant change, we consider it equivalent to a detection. In terms of evaluation, if our LSTM model can correctly predict a sustained change in noise levels for future timestamps, we consider it a success. In our assessment, as depicted in Figure 8 below, the model successfully predicted up to 88% of the detections for *threshold* values as low as 0.01, and the accuracy was 99% for values as high as 0.1.

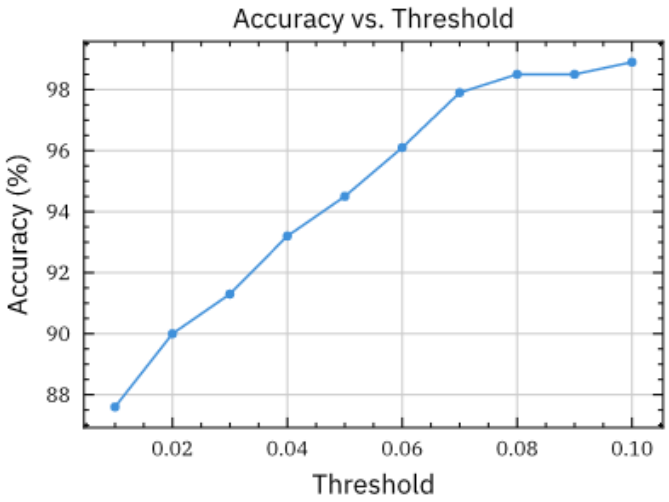

Figure 8: Accuracy of the developed noise forecasting system as a function of the threshold used at the computational part of the system

3.2.8 Evaluation

3.2.8.1 Results and KPI measurements

To evaluate the efficacy of the MLSysOps framework in the smart city domain, we analysed the system's performance across different deployment environments and temporal windows. Specifically, we compared the energy profile and detection accuracy of the smart lampposts under three distinct management strategies. The results presented below are derived from real-world telemetry collected from Cluster 1 (Nodes #0, #1, #2 at UBIW Headquarters/Parking) and Cluster 2 (Nodes #3, #4 in public settings in Aveiro).

To establish a basis for the coherent evaluation of the trade-off between sustainability and service quality, the following metrics were defined:

- Energy Consumption (kW): The cumulative energy used by the edge node.
- Energy Savings (%): The percentage reduction in energy consumption compared to the baseline "Always-On" configuration.
- Capture Rate (%): The ratio of vehicle detections (Car, Moto, Truck, Bus, Bike) successfully recorded by the system, relative to the ground truth (Captured Detections vs Real Detections).
- High-Level KPI: The primary objective is to maintain a capture rate above 95% (accepting less than 5% lost detections) while maximizing energy savings.

CV Component Activation Strategies

1. Always-On (Baseline): The image processing application component runs continuously. This represents the reference point for calculating savings and establishes the theoretical maximum for Capture Rate (100%). Ground truth was obtained by running video streams in parallel, in-house applications, during the test period.
2. Heuristic-based Engagement: Decision-making is based on a static Simple Moving Average (SMA) threshold.
3. ML-based Engagement (MLSysOps): Utilizes the LSTM-based Variation model to predict activity.

Crucially, both strategy 2 and 3 incorporate Hysteresis Logic via a Minimum Runtime Timer. Once activated, the CV component is forced to remain "ON" for a minimum of 10 minutes during the day and 5 minutes at

night. This ensures the system protects hardware from high-frequency oscillation and ensures detection continuity.

Adaptive Normalization for Installation Variance

A major challenge in scaling IoT deployments is the variability in physical installation conditions. Noise sensors mounted on standard streetlights (8–12 meters high) register significantly lower sound pressure levels due to signal attenuation compared to sensors mounted on lower infrastructure. Consequently, a fixed activation threshold is non-viable.

To address this, the MLSysOps framework implements an Adaptive Normalization layer. The system dynamically calculates the operational noise range for each node, defining the upper bound as the 99th Percentile (P99) of its historical profile. This normalizes the data based on relative variation rather than absolute volume. Whether a passing vehicle generates a 40dB spike (low altitude) or a subtle 5dB ripple (high altitude), the normalization maps both events to the same [0, 1] input feature space. This allows the single pre-trained LSTM model to generalize across heterogeneous physical deployments without site-specific recalibration.

Threshold Sensitivity Analysis

To identify the optimal operating point, a systematic sensitivity analysis was performed to identify the activation threshold that maximizes Energy Savings while strictly adhering to the 95% Capture Rate KPI.

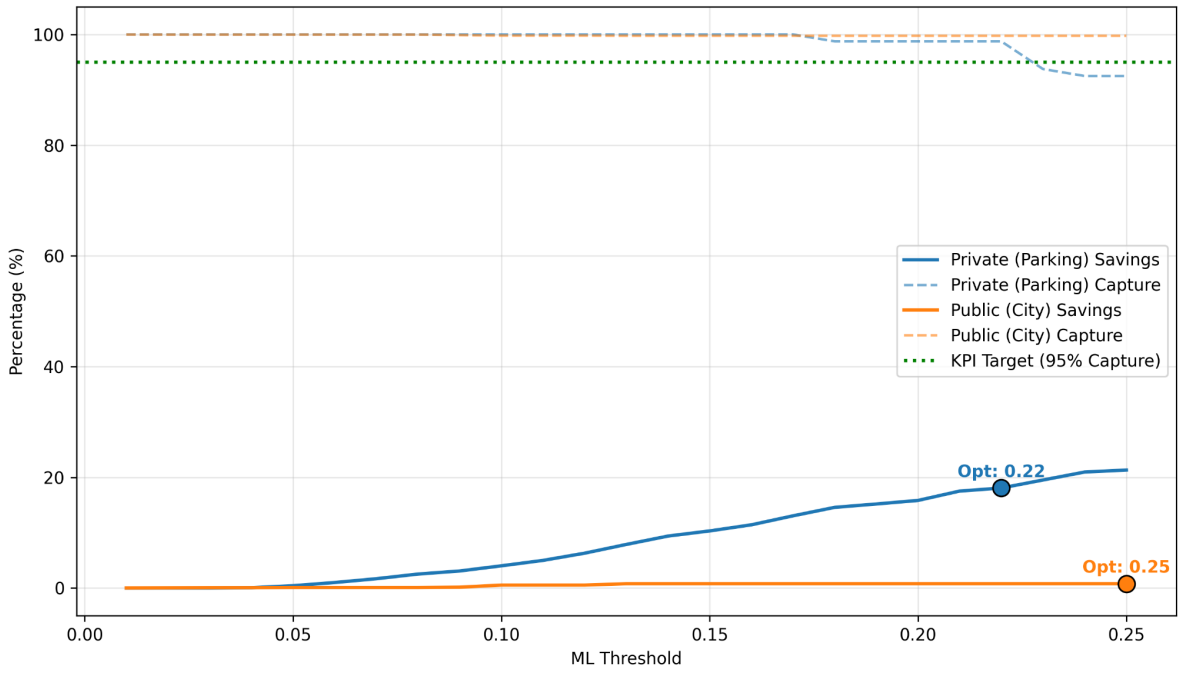

Figure 9: Sensitivity Analysis & Threshold Optimization Plot

As illustrated in Figure 9, the analysis yielded the following observations:

1. Robustness Region: For the private cluster, the system maintains a 100% Capture Rate up to a threshold of 0.175, indicating high model confidence in distinct traffic signatures.
2. The Safety Limit: Beyond a threshold of 0.23, the Capture Rate drops sharply below the 95% threshold, defining the upper limit of safe operation.

3. Public Independence: The public cluster shows minimal sensitivity to the threshold value. Due to high traffic frequency and the 10-minute hysteresis logic, the system effectively operates in a stable "Always-On" state during active hours, while maintaining high capture rates on less active times, regardless of the sensitivity setting.

Based on this analysis (summarized in Table 4), a global threshold of 0.20 was selected. This aggressive operating point provides a ~15.8% improvement in energy savings for private nodes (compared to 10.3% at 0.15) while still maintaining a safe 98.8% Capture Rate, comfortably exceeding the safety KPI.

Table 4: Sensitivity Analysis Data Table

Threshold	Deployment	Capture Rate (%)	Energy Savings (%)	Safety KPI
0,05	Private (Parking)	100	0,4	PASS
0,05	Public (City)	100	0,1	PASS
0,075	Private (Parking)	100	2,2	PASS
0,075	Public (City)	100	0,1	PASS
0,1	Private (Parking)	100	4	PASS
0,1	Public (City)	99,8	0,5	PASS
0,125	Private (Parking)	100	6,6	PASS
0,125	Public (City)	99,8	0,8	PASS
0,15	Private (Parking)	100	10,3	PASS
0,15	Public (City)	99,8	0,8	PASS
0,175	Private (Parking)	100	13,7	PASS
0,175	Public (City)	99,8	0,8	PASS
0,2	Private (Parking)	98,8	15,8	PASS
0,2	Public (City)	99,8	0,8	PASS
0,225	Private (Parking)	93,8	19,2	FAIL
0,225	Public (City)	99,8	0,8	PASS
0,25	Private (Parking)	92,5	21,3	FAIL
0,25	Public (City)	99,8	0,8	PASS

Overall Operational Performance

The deployment of the MLSysOps framework was monitored continuously from December 19, 2025, to January 5, 2026. Table 5 summarizes the key operational metrics across all monitored nodes, contrasting the high-efficiency profile of private infrastructure (Nodes 0–2) with the high-availability profile of public infrastructure (Nodes 3–4).

Table 5: Overall Operational performance

Node	Savings Min	Savings Max	Savings Avg	Capture Min	Capture Max	Capture Avg
node0	13,93%	27,86%	21,31%	42,86%	100%	80,99%
node1	3,48%	25,43%	13,11%	0%	100%	86,21%

node2	11,14%	40%	22,80%	16,67%	100%	71,12%
node3	0,79%	6,59%	2,83%	98,19%	99,76%	99,03%
node4	0,21%	7,22%	4,15%	98,09%	100%	99,06%

Note: Node 1's 0% capture on Dec 30 was an edge case involving a single vehicle event

The operational dataset confirms the system's dual-mode efficacy:

- High-Efficiency Mode (Private Nodes): In controlled environments, the system delivers double-digit energy savings (averaging ~19%), peaking at the theoretical hardware limit of 40% on quiet days
- High-Availability Mode (Public Nodes): In chaotic environments, savings are deprioritized (averaging ~3.5%) to maintain a capture rate consistently above 99%, validating the safety-first logic of the MLSysOps framework

Operational Performance: Cluster 1 (Private/Parking)

The deployment of the MLSysOps framework was monitored continuously from December 19, 2025, to January 5, 2026. The data reveals a distinct trade-off profile for the private parking area, characterized by sparse, irregular activity:

- **Energy Efficiency:** The ML strategy achieved substantial savings, averaging **~19%** across the cluster and peaking at the theoretical hardware maximum of **40%** on days with zero traffic (e.g., weekends).
- **Detection Integrity:** While the system frequently achieved 100% capture on active days, the average capture rates (71%–86%) reflect the inherent challenge of "cold-start" detection in low-traffic environments. The lower minimums (e.g., 42% on Node 0) typically occurred on days with fewer than 5 total events, where missing a single vehicle significantly impacts the daily percentage. This effect is likely exacerbated by low-noise vehicles (e.g., electric vehicles operating at low speeds, bicycles, etc.); without significant noise, these vehicles may fail to breach the activation threshold, constituting a known physical limitation of acoustic-only triggers.

Figure 10: Node #0 (Private Parking) – Dec 23, 2025

Figure 10 clearly differentiates the three approaches taken. The Always-On strategy (top) captures everything but wastes significant power during the long idle gaps. The Heuristic strategy (middle) is unable to differentiate between a noisier background and the actual noise events that are due to detection events. The ML-based strategy (bottom), however, successfully identifies the event's "shape." It pre-activates the node to capture the full sequence (98.8% Capture Rate) and then powers down, achieving ~16% energy savings—effectively combining the reliability of the baseline with the efficiency of the heuristic.

Visually, the dark markers indicate the ML Triggers—the specific points where the model identifies a noise variation and initiates a wake-up. The vertical green bands represent the resulting 'Active Mode' (system power-on). The success of the strategy is visible in the overlap of the solid green line (Captured Detections) and the

dashed red line (Real Detections). When these two overlap, it confirms the system was active at the exact moment of the traffic event. Any red lines appearing without a green overlap would represent missed detections.

A few ML triggers (dark markers) occur without subsequent detections because noise can spike from non-traffic sources. This causes the system to stay active briefly, leading to lower savings than theoretically possible but ensuring no real events are missed.

Figure 11: Node #0 (Private Parking) – Dec 25, 2025

Figure 11 highlights a failure mode for both power-saving strategies. The Heuristic strategy failed because several instances of no detection events crossed the static dB threshold. Similarly, the ML-based strategy interpreted the isolated event as background variation, failing to wake the node. Only the Always-On baseline

fully captured the events. This shows that for ultra-sparse traffic, with less than clear noise patterns, the energy vs. accuracy trade-off hits a hard limit where predictive models may not be ideal.

Operational Performance: Cluster 2 (Public/City)

The public deployment presents a distinct challenge: sensors are mounted at significant heights, resulting in attenuated noise readings (typically peaking below 20dB). The ML-based approach leverages Adaptive Normalization to compensate for this attenuation, successfully detecting traffic signatures even when the raw signal is weak.

In this high-density environment (Figure 12), energy savings are minimal (averaging ~3.5%). This is a deliberate outcome of the operational stability logic. Because traffic events occur frequently, the 10-minute minimum runtime timer is constantly reset, effectively maintaining the CV component in a continuous 'Active' state. While this limits energy recovery, it validates the system's safety logic: the system correctly identifies the high-traffic context and defaults to a High-Availability Mode, ensuring the Capture Rate remains consistently above 99%.

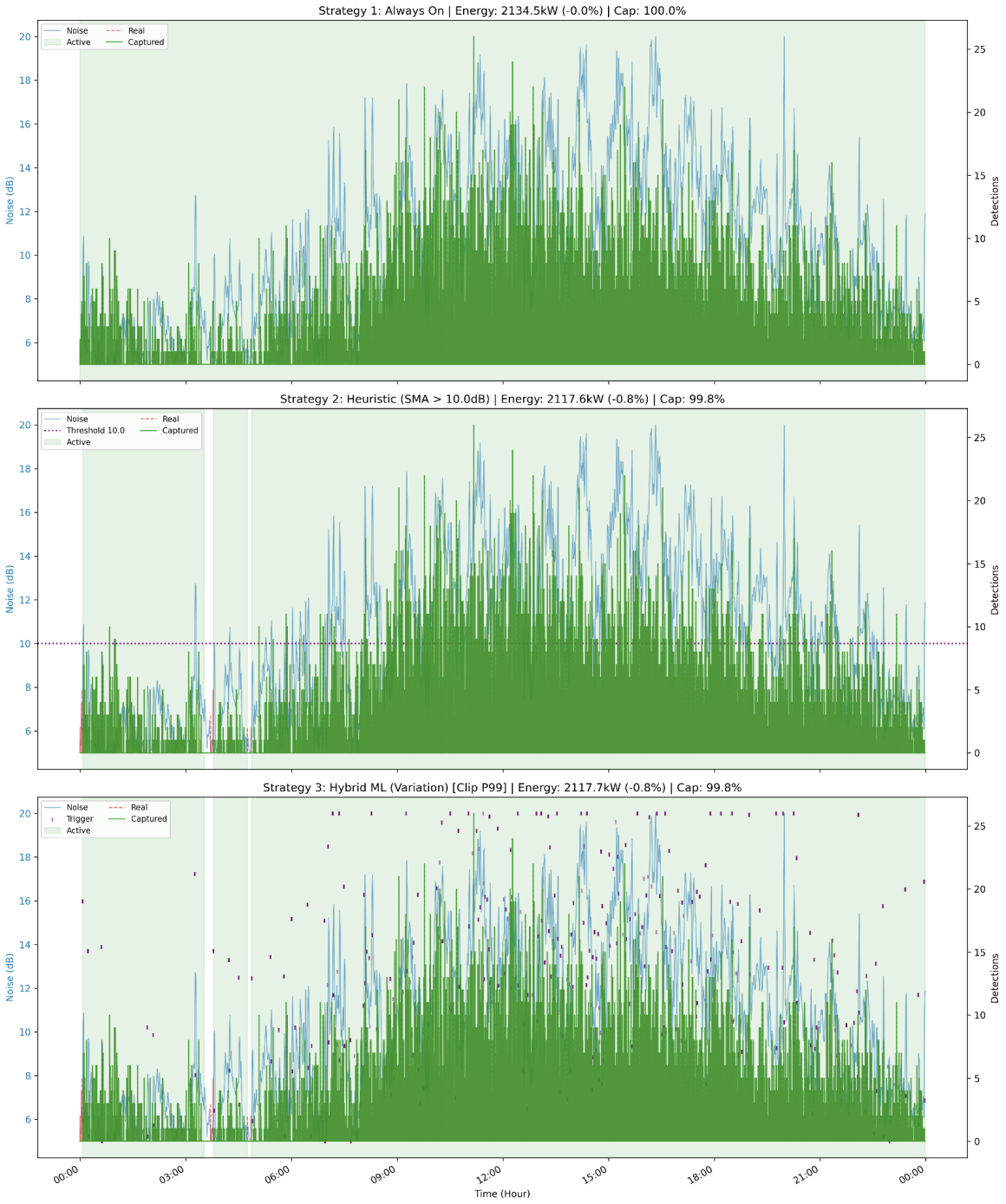

Figure 12: Node #3 (City Centre) – Jan 03, 2026

The ML-based strategy, utilizing adaptive normalization, correctly identifies that the traffic density requires continuous monitoring. It maintains a stable "Active" state similar to Always-On baseline, ensuring a 99.76% Capture Rate. The system effectively "decided" that attempting to save energy was not viable, defaulting to a high-availability mode.

Temporal Analysis

A 72-hour continuous analysis overviews the system's adaptability to the city's pulse (Figure 13, Figure 14). During Deep Night hours (01:00–05:00) is the only time that the system can power down on the Public City Centre test. Lowest noise levels and traffic were reported at the 3:00 to 4:00 am time range.

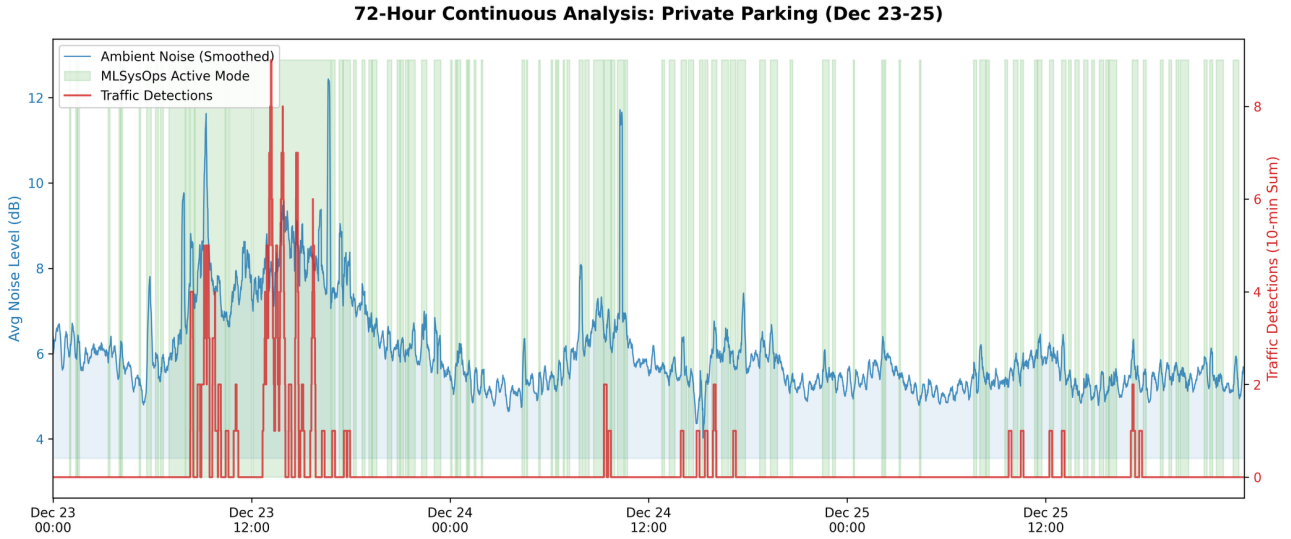

Figure 13: 72-Hour Continuous Analysis Plots of the private testbed

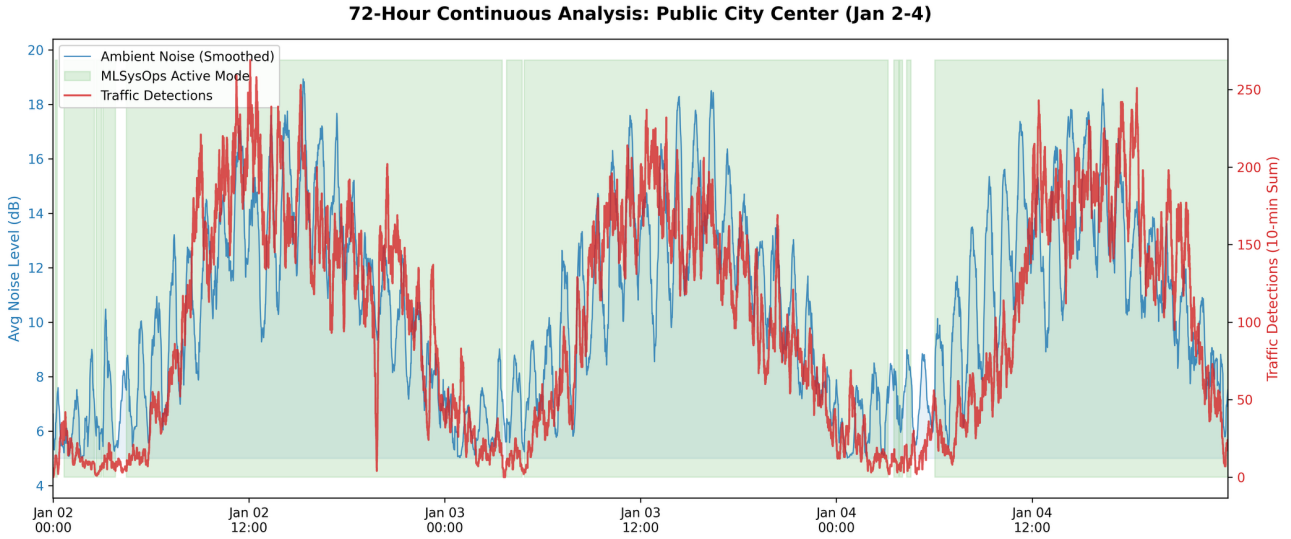

Figure 14: 72-Hour Continuous Analysis Plots of the public testbed

KPI Achievement Summary

The deployment of the MLSysOps framework demonstrated that intelligent edge management can balance sustainability with safety. The system achieved the maximum theoretical energy reduction of 40% during periods of zero traffic (representing the total deactivation of the CV component). It maintained an average reduction of ~20% across the Private Parking cluster during active operational days.

By combining predictive activation with stabilizing hysteresis timers and adaptive normalization, the system successfully met the 95% detection target in several of the tested environments, with focus on the public city testbed.

3.2.8.2 Portability, Generality & Scalability

While the DMZ-based abstraction layer served as a robust environment for prototype validation, the production deployment of MLSysOps relies on direct management of the edge infrastructure. To assess the feasibility of a city-wide rollout, we analyse the scalability limits across two key dimensions: Computational Capacity (Edge) and Orchestration Capacity (Cluster).

Computational Scalability (Edge Limits)

The immediate constraint for scaling is the finite resources of the individual Edge Node.

Inference Overhead: The additional load introduced by the MLSysOps local agents (LSTM noise prediction) is negligible. These models are lightweight and optimized for CPU execution, leaving the GPU entirely dedicated to the primary Computer Vision application.

Resource Contention: Stress tests indicate that the Edge Node can support the MLSysOps monitoring stack alongside the heavy YOLO-based CV application without performance degradation. The node architecture allows for the vertical scaling of additional lightweight sensors (e.g., air quality, humidity) without requiring hardware upgrades.

Infrastructure & Orchestration Scalability (System Limits)

Moving beyond the single node, the scalability of the central control plane (Cluster Agent and Karmada) becomes the defining factor.

Telemetry Bottlenecks: As the fleet expands from tens to thousands of nodes, the volume of telemetry data (MQTT messages) increases linearly. The current architecture mitigates this by performing Edge-Side Filtering: nodes only transmit necessary detection metrics rather than raw streams, while the remaining telemetry is consumed directly on the node. This significantly reduces network bandwidth usage.

Cluster Segmentation: The MLSysOps framework supports horizontal scalability through cluster federation. A single Cluster Agent can effectively manage hundreds of nodes before decision latency increases. For a city-wide deployment involving thousands of smart lampposts, the infrastructure would be segmented into geographical clusters (e.g., one cluster per city district).

Central Management: The Karmada control plane sits above these regional clusters, providing a unified policy view. This hierarchical design ensures that the system does not hit a hard limit on node count; instead, scalability becomes a function of adding additional regional aggregation points, allowing the network to grow indefinitely to cover entire metropolitan areas.

3.3 Smart Agriculture Application

As illustrated in Figure 2, the smart agriculture use case followed a phased framework adaptation and integration approach. Year 2 focused on tasks #1 through #4, while Year 3 addressed tasks #5 through #7. Below, we provide a summary of the application concept, explain how the MLSysOps framework supports the desired system/application management, discuss the development/integration work for this use case, and present the evaluation results.

3.3.1 Concept, Role of MLSysOps, Application KPI & System Optimization Objective

The goal of the smart agriculture application use case is to improve the performance of targeted weed spraying in the field by complementing the camera system and embedded processing platform running on a tractor with a similar system that is installed onboard a drone that operates in tandem with the tractor. The basic assumption is that, in some cases, the vertical camera view of the drone system can lead to more robust weed detection vs the camera view of the tractor system, which may have issues due to flares and shadows, leading to improved weed detection system availability and consequently the targeted spraying of herbicides only when and where they are needed.

The primary function of MLSysOps is to engage the drone when it is likely to enhance the tractor's weed detection performance significantly. Otherwise, the drone remains disengaged to conserve energy and battery life. There is a natural trade-off between improving weed detection and minimising energy consumption. Frequent or prolonged drone engagement may yield marginal improvements in weed detection, but at the cost of considerable energy usage, especially in large fields where continuous or repeated drone flights can limit area coverage, require mid-operation recharging, or risk disrupting the overall spraying operation. Additionally, deploying the drone requires substantial energy, including the time it takes to reach the tractor and begin assisting, as well as returning to its landing area after completing a mission.

The performance of weed detection is captured through suitable application-level metrics with specific satisfaction thresholds, declared as part of the corresponding application description. At runtime, the application reports the current performance by emitting the respective values via telemetry so that this information can be consumed by the MLSysOps agent(s). From the application perspective, the KPI is to improve by 5% the amount of time where the system has satisfactory performance, defined as periods during which weed detection operates outside safe mode and produces reliable detection results above predefined confidence thresholds, due to the drone operating in tandem with the tractor vs the tractor alone. From the system perspective, the KPI is to minimise the energy spent by the drone (i.e., the flight time) while meeting the application-level target.

3.3.2 Application Structure

The application consists of four basic components: (i) the drone driver/controller, (ii) the weed detection component for the drone, (iii) the weed detection component for the tractor, and (iv) the controller for the sprayer on the tractor. For practical (but also safety) reasons, the drone controller is pre-deployed on the drone node. MLSysOps is responsible for the deployment/management of the two weed detection components and sending commands to the drone controller to engage/disengage the drone. Both weed detection components are similar in their image processing functionality but also have some differences. On the one hand, the drone component sends weed detection information to the tractor component. On the other hand, the tractor component receives the data generated by the drone component and is responsible for generating control commands for the sprayer controller. It also sends tractor position/speed information to the drone controller. Both weed detection components emit application performance metrics via telemetry. The drone controller component also emits state information via telemetry based on whether the drone is ready to operate, is following the tractor, or is returning to land (when explicitly disengaged or running out of battery). Finally, the sprayer controller component is pre-deployed on the tractor. It drives the hardware at the rear-end of the tractor to start/stop

spraying through one or more nozzles, depending on the commands received from the tractor component. Note that when the tractor component cannot detect weeds, it enters the so-called “safe mode” of operation, where it asks the sprayer component to spray herbicide in a “blind” way through all nozzles (default spraying mode). In this case, the engagement of the drone and the activation of the drone weed detection component can provide the tractor component with the necessary weed location information to continue in spot spraying mode.

```yaml
MLSysOpsApplication:
 name: augmenta-app
 cluster_placement:
  cluster_id:
   - "UTH-AUG1"
 components:
  - metadata:
    name: tractor-app
    node_placement:
     labels:
      - "node-type:tractor"
    qos_metrics:
     - application_metric_id: TractorWeedDetection
       target: 50
       relation: greater_than
    restart_policy: OnFailure
    containers:
     - image: AUG_tractor_app:latest
       image_pull_policy: IfNotPresent
  - metadata:
    name: drone-app
    node_placement:
     labels:
      - "node-type:drone"
    restart_policy: OnFailure
    containers:
     - image: AUG_drone_app:latest
       image_pull_policy: IfNotPresent
 global_satisfaction:
  threshold: 0.9
  relation: greater_or_equal
  achievement_weights:
   - metric_id: TractorWeedDetection
     weight: 1
```

Figure 15: MLSysOps application description for the smart agriculture use case.

The core part of the deployment description for this application, based on the MLSysOps format, is shown in Figure 15. For the sake of the example, we assume the application should be deployed in a specific cluster, representing a certain system installation, e.g., the real-world testbed of AUG in Volos. The placement requirements for the two application components that are managed via MLSysOps are specified through corresponding node labels, which are to be matched with the respective node properties (specified in the system infrastructure description). The application performance metrics emitted by each component, along with the respective targets, are specified at the component level. More specifically, the tractor component emits a metric reflecting the local weed detection performance (TractorWeedDetection), with a target value greater than 50 (out of 100) indicating satisfactory detection quality. The global satisfaction level of the application depends on whether the tractor weed detection target is met.

3.3.3 Development of Application Components

Drone controller. The drone controller component has been fully developed and thoroughly tested using the drone simulation environment of UTH (see deliverable D4.3 “Final Version of System Simulators”) as well as on the real drone in the field. Several field tests have been conducted to confirm that the drone successfully performs the full operation cycle (arm, take-off, approach the tractor, follow the tractor, return to land, disarm) based on high-level commands issued to the drone controller from a remote computer in a manual way (terminal-based commands).

Tractor weed detection. The tractor weed detection component has been adjusted to (i) produce the desired application metrics, (ii) send telemetry information to the drone component (e.g., tractor location), and (iii) adjust the target rate of the sprayer control component based on the drone’s calculated weed locations if that is requested.

Drone weed detection. The drone weed detection component has the same capabilities as the tractor detection component. Still, it has been modified to model how the drone moves so it can accurately determine the location of the detected weeds. It also integrates the capability to send the calculated weed locations to the tractor component. Note that this component does not interact with the sprayer controller component.

Sprayer controller. The sprayer controller component is reused without any changes. It is the same version that is currently being used in the AUG tractor devices sold/installed worldwide.

3.3.4 Framework Instantiation and Agent Logic

The tractor and drone nodes are part of a cluster representing the field where the application is deployed and managed under the control of MLSysOps. There can be several clusters, each consisting of a tractor-drone pair running the same or different versions of the application, to support deployment and operation on different fields. A continuum-level agent manages the entire system slice. In turn, each cluster is managed by a corresponding controller/orchestrator agent. Finally, each node runs a node-level agent.

Clusters are assumed to operate independently; hence, we focus on the control within a single cluster. Figure 16 illustrates the system setup, the core framework components, and the application deployment when the drone is engaged, with both application components deployed on the respective nodes.

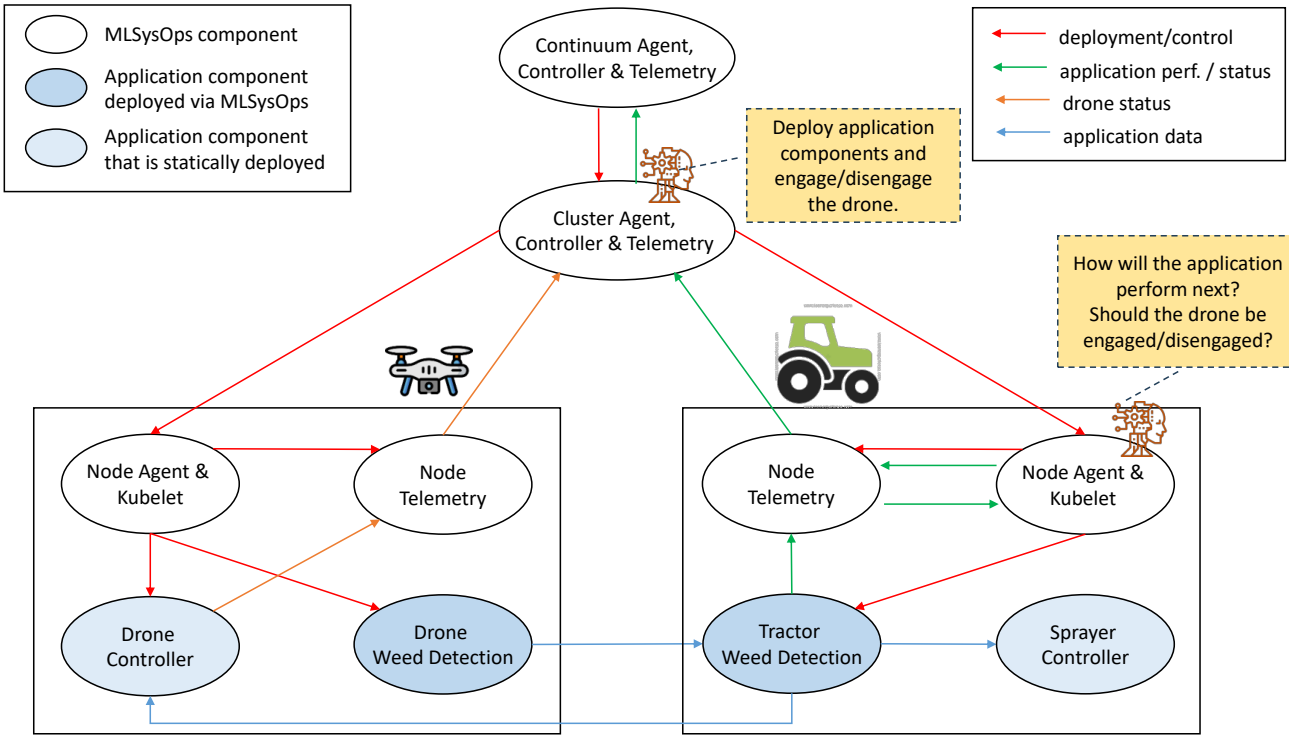

Figure 16: MLSysOps framework for the smart agriculture use case.

The intelligence of system and application management is distributed between the node and the cluster level. The tractor node agent retrieves from telemetry the satisfaction/performance level of the local weed detection component, and it uses this information to predict the expected weed detection performance in the future. In addition, the agent decides whether it makes sense for the drone to be engaged. The weed detection performance prediction and drone engagement hint are propagated to the cluster level through telemetry. The prediction of application performance and the generation of the drone engagement hint are done using an ML model trained using real application data from field operations. The cluster agent monitors the state of the nodes, the state of the drone controller, and the predicted weed detection performance and drone engagement hint, and it uses this information to decide whether to engage/disengage the drone. The cluster agent logic is based on a finite-state machine (FSM), shown in Figure 17.

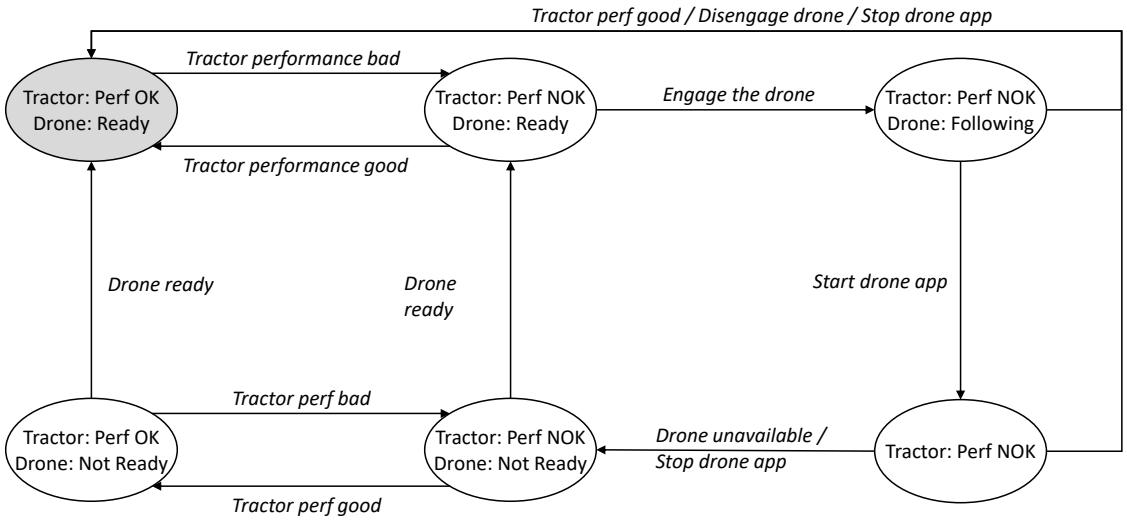

Figure 17: FSM used by the cluster MLSysOps agent to control the engagement/disengagement of the drone and the deployment of the drone weed detection component accordingly.

In a nutshell, the drone is engaged when the tractor's performance is expected to be bad for some time, and the tractor agent produces a hint to engage the drone. It is disengaged when tractor performance is expected to become good again, and the hint is produced to disengage the drone, or when the drone becomes unavailable (it runs out of batteries and must return to land). Note that the weed detection application on the drone is deployed and starts running only when the drone starts following the tractor, and it is removed when the drone is disengaged. Based on the results of the real-world experiments, the drone weed detection component is expected to have satisfactory performance; thus, no additional checks are added in the FSM for this. Thus, the two factors that determine the engagement of the drone are the predicted performance of the tractor weed detection component and the status of the drone.

To focus on the essence, the above FSM assumes that the tractor is always available and that the tractor component has already been deployed and is running there. The actual (implemented) FSM is more complex, handling the case where the tractor is/becomes unavailable (the drone remains/is disengaged). This is because the tractor is operated by a human who may decide to turn on/off the node at any point in time, so the system must be robust against such abrupt "node failures".

3.3.5 Integration and Testing

Integration and testing were performed in a stepwise and controlled way, using three different types of setups, each serving a different purpose. The concrete setups and the tests that have been performed are described in the following sections.

3.3.5.1 Virtual setup

The virtual system setup was prepared using UTH's research testbed. Instead of the real tractor and drone nodes, we use two virtual nodes (VMs) in the simulation environment of UTH (see deliverable D4.3 "Final Version of System Simulators"). The virtual tractor and drone nodes run real autopilot software (Ardupilot framework[10]) configured to operate in software-in-the-loop (SITL) mode[11] with the physics engine running in Gazebo[12]. This way, it is possible to run missions where the tractor follows a prespecified path to scan a field while the drone is engaged dynamically based on the decisions at runtime taken by the cluster agent. Furthermore, this ensures that the virtual drone node behaves similarly to the real drone. It is also important to note that the drone controller component is the same as that of the real drone of AUG. This considerably simplifies portability as this setup can be transferred to the real drone with minimal configuration changes; one merely needs to change the communication settings of the drone controller component to talk to the autopilot over a serial vs a UDP socket.

Using this setup, exhaustive tests were conducted to verify the correct operation of all control/telemetry flows, the agent logic, and the functionality of the deployment mechanisms of the MLSysOps framework. Since we want to stress test the MLSysOps framework, instead of using the real application components of AUG, we use proxy components that emit similar performance metrics. These components are programmed to generate these metrics in a controllable/predictable way to trigger the engagement and disengagement of the drone according to the desired test scenarios. Thanks to this setup, several bugs and glitches have been identified and repaired before trying to use MLSysOps with the real nodes.

[10] Ardupilot, [Online]. Available: https://ardupilot.org/

[11] SITL, [Online]. Available: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

[12] Gazebo, [Online]. Available: https://gazebosim.org/home

3.3.5.2 Desk setups

Two desk setups were used to perform extensive tests using the same nodes as in the field setup, before the actual field tests. These were mainly used to iron out technical issues regarding the specific hardware/software platforms of the tractor/drone devices and the networking with the cluster level and between them. There are two separate testing environments with distinct goals. The *MLSysOps framework* desk setup was used to test the networking and deployment of the tractor and drone applications through the MLSysOps framework, while the *video playback weed detection* desk setup was used to evaluate drone weed detection and localization performance, as well as verify weed transfer.

3.3.5.2.1 MLSysOps Framework

This setup uses two physical nodes, installed at AUG headquarters, representing the tractor and drone devices, featuring the same HW platform and peripherals as the real devices. These run the node-level parts of the MLSysOps framework and are connected over 4G (as this is the case in the field configuration) to the cluster agent and subsequently to the continuum agent, which are VMs hosted on AUG's cloud.

For this and all subsequent tests, the tractor and drone weed detection modules are packaged into standalone portable containers, which are deployed on the nodes through the MLSysOps framework. With this setup, all application and framework telemetry was running (see Figure 16) except for weed position transfer, which was tested in the second desk setup.

The drone controller is connected to the SITL configuration of the autopilot to validate its behaviour upon receipt of commands (Take-off, Return-To-Land). To maintain control over the tractor node's prediction, the ML model is substituted with a custom script that behaves in a predictable pattern upon manual modulations of the node's telemetry.

This setup provided the ability to controllably induce the framework to all possible real-world scenarios with no modifications to its installation or functionality. All tests that have been done using the fully virtual setup in the UTH testbed have also been successfully repeated in this extended setup. Additional tests were performed to validate the propagation of application telemetry from the tractor weed detection module to the drone controller, from the drone controller to the drone weed detection module, and from both weed detection modules to their respective node agents.

3.3.5.2.2 Video playback weed detection

This setup was used to develop drone detection and localization, and to verify weed transfer between the drone and the tractor. As mentioned earlier, it is a separate testing environment from the MLSysOps framework, as it relates solely to the weed detection application and not to the networking and deployment of the application containers on nodes.

Initially, several drone flights were conducted on the field to capture and save videos from cameras. These videos are fed to the video-playback weed detection environment of AUG that replays the full operation cycle (detecting, localizing, and finally spraying weeds) through visualization windows and maps (see Figure 18, Figure 19, Figure 20).

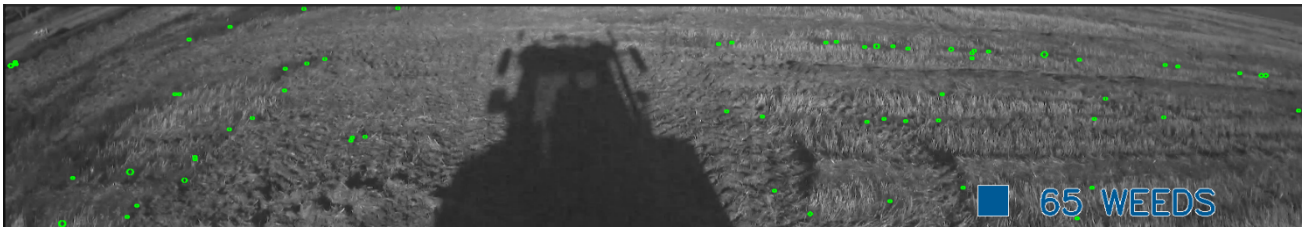

Figure 18: Detection window of video playback, with green points indicating the tractor-detected weeds.

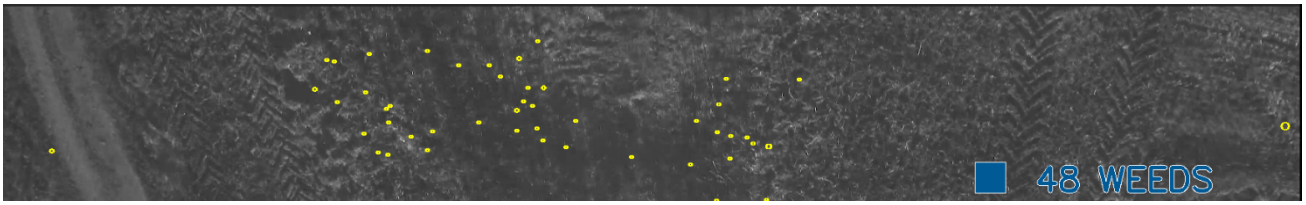

Figure 19: Detection window of video playback, with yellow points indicating the drone-detected weeds.

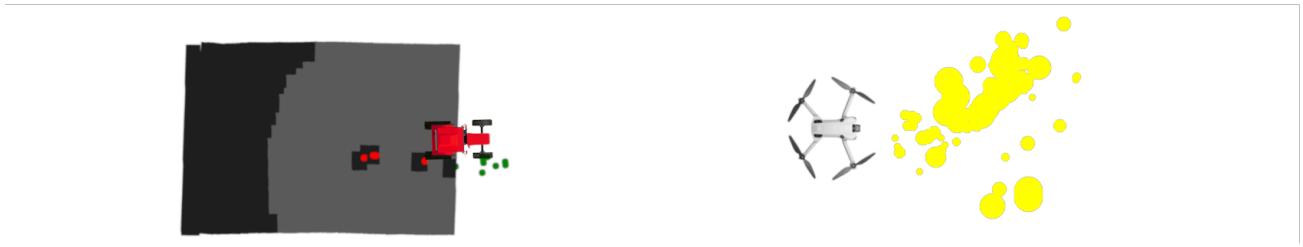

Figure 20: Map visualization of the video-playback weed detection. Green and yellow points indicate tractor and drone-detected weeds, respectively; red points indicate sprayed weeds. The tractor is shown with its attached sprayer; red areas are sprayed while grey areas are not.

Having these videos available, the first step was to adjust the detection sensitivity of the drone and finally evaluate the result across several frames. The aim was to set the optimal sensitivity for the drone weed detection algorithm to match the tractor's detection performance in good conditions (see Figure 21).

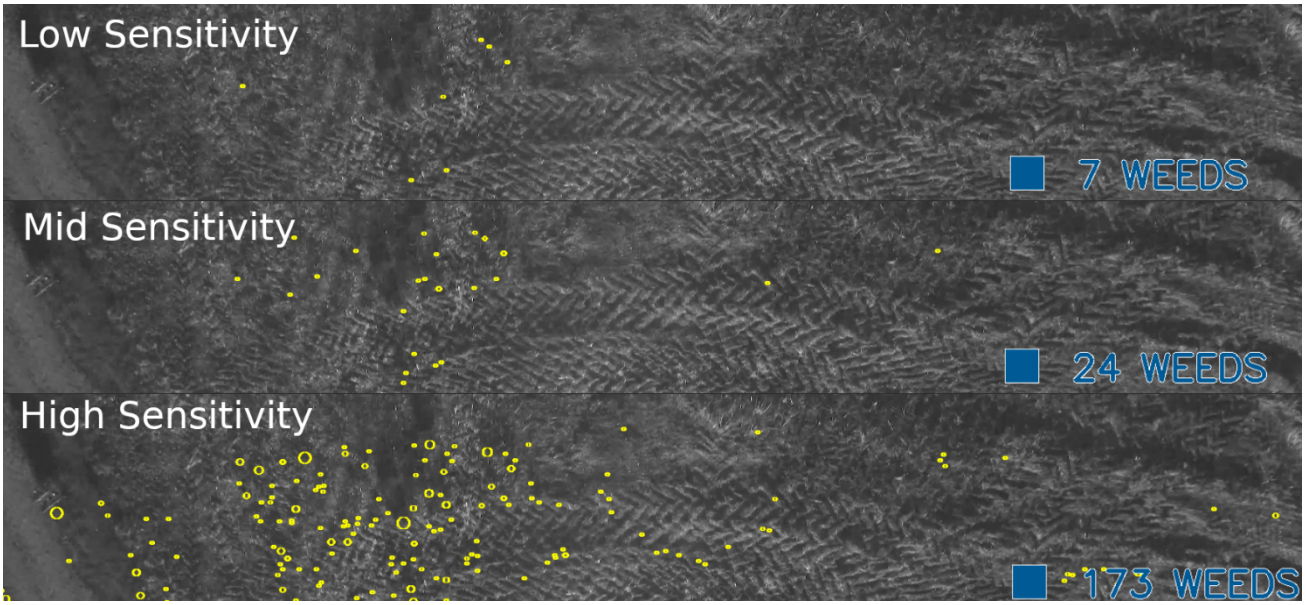

Figure 21: Tuning of sensitivity settings for drone detection on a static frame. Low sensitivity: Missed weeds. Medium sensitivity: All weeds are detected. High sensitivity: All weeds are detected, but with excessive noise.

After confirming that the drone detection performs effectively, the next step was to develop a localization scheme considering the drone's kinematic model. More specifically, a proper filtering method was developed to obtain accurate position, velocity, and orientation even when sensor data is noisy or incomplete. The stability of the drone movement was evaluated on several plots (see Figure 22).

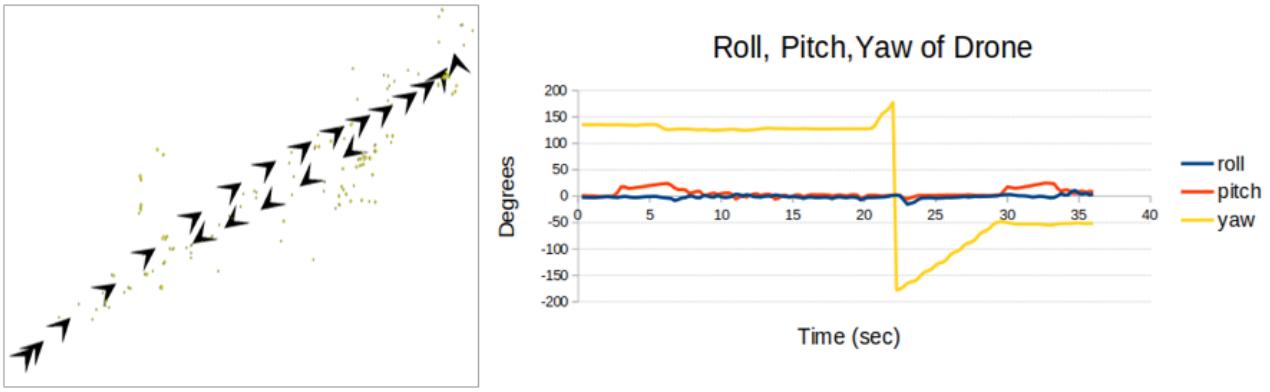

Figure 22: Evaluating stability of the drone's trajectory and attitude using the video playback weed detection.

The weed transfer functionality from the drone to the tractor was examined on the same map. The video-playback weed detection environment of AUG can assign different colours to weeds detected by the drone from the weeds detected by the tractor (see Figure 20). The localization accuracy of drone-detected weeds was validated by comparing their coordinates with those recorded by the tractor.

3.3.5.3 Field setup

The field used in these tests is located in Perivlepto, near the city of Volos. It is one of the test environments of AUG, primarily for weed detection experiments. Its dimensions are 200 m × 200 m, and it is slightly uphill but generally flat.

Figure 23: The left image is a screenshot from Google Maps showing the field's dimensions, while the right image is a photograph taken on-site. There is no sprayer attached to the tractor, as the data gathered from the tests suffice for the evaluation.

This final setup was used for complete end-to-end testing and subsequent evaluation. It consists of the tractor with its cabin-mounted device and the drone with its own device. The containerized applications are deployed on the nodes, the ML model is running, and telemetry/application status is being propagated across all nodes. The drone weed detection is calibrated and sends the positions of detected weeds to the tractor's spraying module in real time. Upon the receipt of commands from the cluster agent, the drone controller automatically controls the drone (for safety reasons, take-off and landing events are intercepted and approved manually before being propagated to the autopilot).

The field setup was also used to collect data, not just the tractor and drone camera feeds and system/application telemetry, but also for the physical movement of the tractor and the drone (used to parameterize the respective simulated movements in the desk setup properly).

3.3.6 Data Collection

As described above, field tests have been performed to validate that the setup is working as expected. During these tests, both the tractor and the drone device were configured to collect video and telemetry data. The latter includes system and application data as described in Table 6 and serve as the core feature for ML development. For application data, a Quality of Service (QoS) target has been generated that represents the acceptance threshold for sufficient weed detection.

Table 6: Data/metrics collected for MLSysOps in the smart agriculture use case.

Source	Metric	Type	QoS Target	Description
System	CPU load	%	N/A	Instant CPU usage
	GPU load	%	N/A	Instant GPU usage
	Memory usage	%	N/A	Instant CPU and GPU memory usage
Application	Quality Indicator 1	Int	> 50	Number of data correspondences between samples
	Quality Indicator 2	Int	> 50	Number of data points used to compute localization components
	Field Indicator 1	Int	N/A	Number of detected weeds
	Field Indicator 2	Float	< 0.1	Fraction of field under environmental variation
	Sensor Fault Probability 1	Float	< 0.3	Probability of the camera sensor being affected by the sun
	Environment Sensor 1	Float	> 600	Measurement of environmental conditions as recorded by the onboard sensor
	Processing Performance	Float	> 15	Average processing performance over time
	Success Rate	Float	1.0	Fraction of successfully processed samples

	Heading	Double	N/A	Instant heading of the vehicle in radians
	Velocity	Double	∈[2, 10]	Instant velocity of the vehicle in m/s
	Latitude	Double	N/A	GPS coordinate of the vehicle in degrees
	Longitude	Double	N/A	GPS coordinate of the vehicle in degrees
	Altitude	Double	N/A	GPS coordinate of the vehicle in meters

The amount of data that has been collected so far amounts to 30 GB for both drone and tractor devices from five successful data collection sessions in the field. This includes both video data (used for the video playback weed detection desk setup) and the telemetry features (used to train the ML model) described in Table 6.

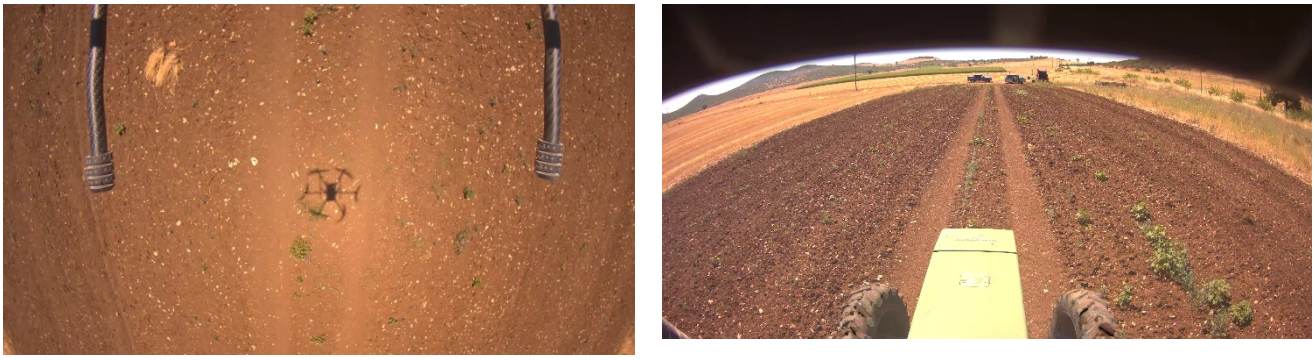

Figure 24: Images captured by the drone and tractor systems during the data collection sessions in the field.

Figure 24 shows indicative images of the drone and the tractor devices captured during the field tests. These images are fed as input to the respective weed detection components running on the devices to produce the metrics listed in Table 6.

3.3.7 Machine Learning Training and Evaluation

An end-to-end analysis and modelling of drone deployment data was performed to predict the *should_fly* signal. First, the dataset was loaded and pre-processed, ensuring timestamps were sorted and converted to datetime. The class distribution of *should_fly* was explored, confirming that the classes were relatively balanced. To capture temporal dependencies, lag features were created for key variables, such as *sensor_fault_probability_1*, *success_rate*, *processing_performance*, *velocity*, and *heading*. Then, a prediction horizon in terms of seconds was defined, allowing the model to forecast the *should_fly* signal several steps ahead, mimicking real-world deployment lead time.

Using these features, an Extreme Gradient Boosting (XGBoost) classifier was trained across multiple train-test splits and random seeds, recording performance metrics such as accuracy, precision, recall, and F1-score. The XGBoost classifier is one of the most popular and powerful machine learning (ML) algorithms used today. It is a decision-tree-based ensemble ML algorithm that uses a framework called gradient boosting. At its core, XGBoost is built on decision trees. More specifically, XGBoost builds multiple decision trees sequentially as

follows: it first creates a simple tree (model 1) to predict the target variable. As expected, the produced tree makes some errors. Next, the algorithm produces a second tree (model 2) that is specifically designed to predict the errors (residuals) made by the first model. Then, a third tree (model 3) tries to correct the errors remaining after models 1 and 2. This process repeats hundreds or thousands of times. Each new tree nudges the prediction closer to the truth by minimizing the "gradient" (the direction of the error) and boosting it towards the correct direction.

The figures below show the model's predictions over four recordings. In more detail, the ground truth values of the *should_fly* parameter were added in the background as shades of red. When the *should_fly* prediction variable is zero, the background is white, and when it is one, the background is shaded red. This variable indicates whether the drone should be dispatched to assist the tractor with the weed detection. With blue, we denote the sensor values of five useful variables: *sensor_fault_probability_1*, *success_rate*, *processing_performance, velocity*, and *heading*. With the red line, the model's predictions are shown.

The model makes predictions for +5 and +40 timesteps into the future, using past and current sensor values (Figure 25 to Figure 32). The longer the forecasting settings the more efficient drone dispatch potentially becomes but with an expected accuracy drop. When the tractor needs drone assistance, we want the drone to be

already nearby. Due to the time needed for the drone to take off and travel to the tractor's location, our model is trained to pre-emptively dispatch the drone so that it's ready to use at the right time.

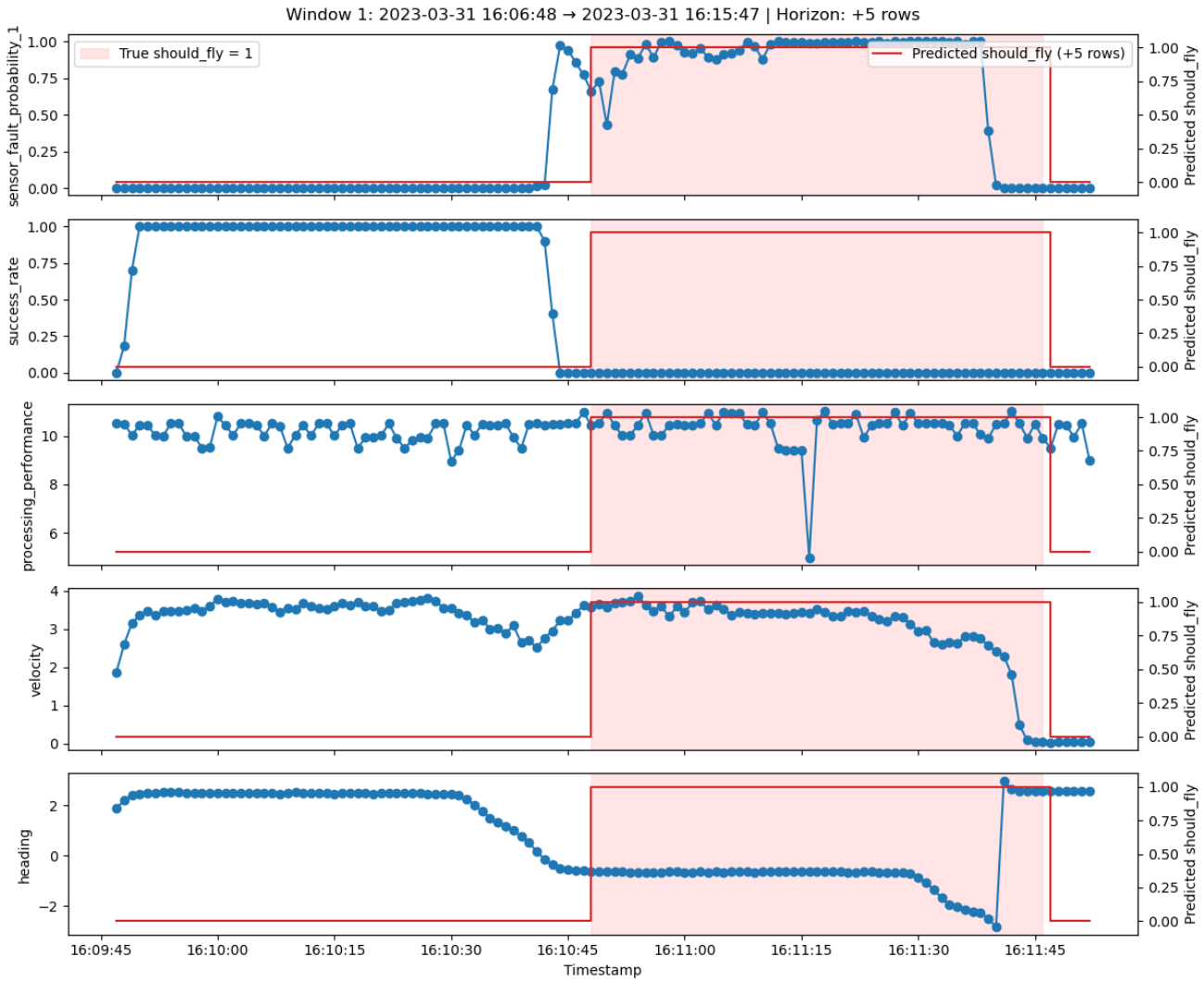

Figure 25: Prediction plot at +5 timesteps

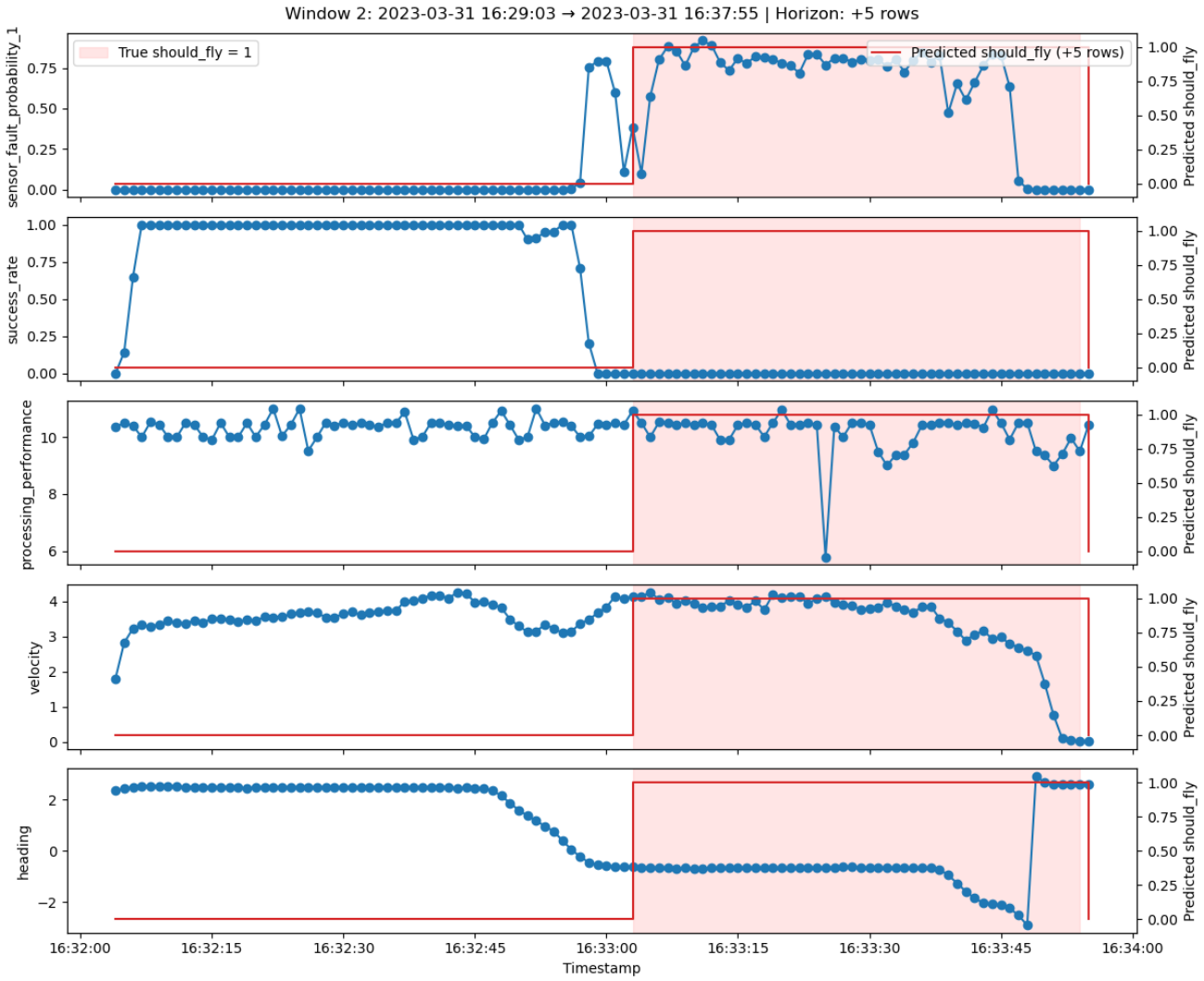

Figure 26: Prediction plot at +5 timesteps

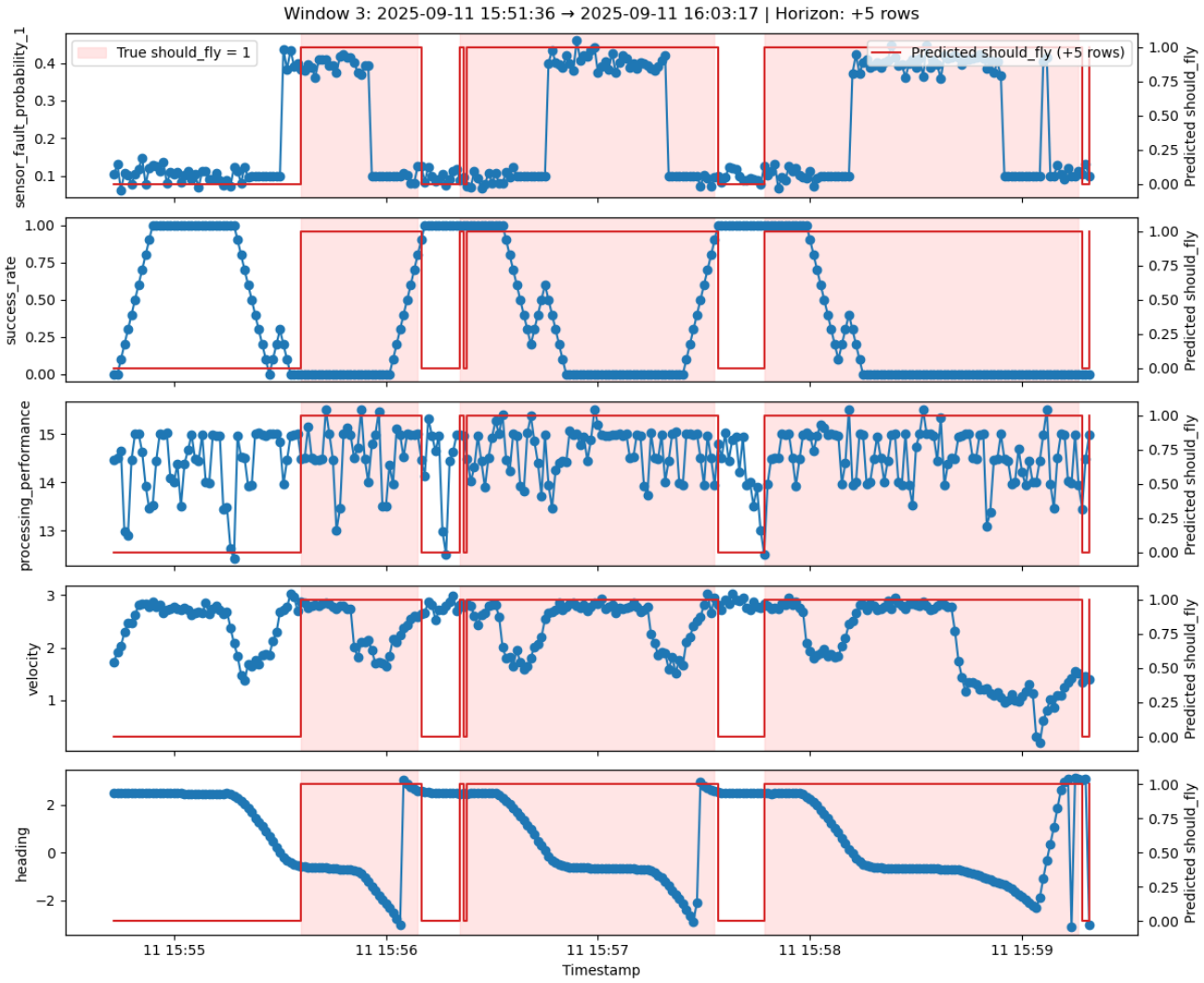

Figure 27: Prediction plot at +5 timesteps

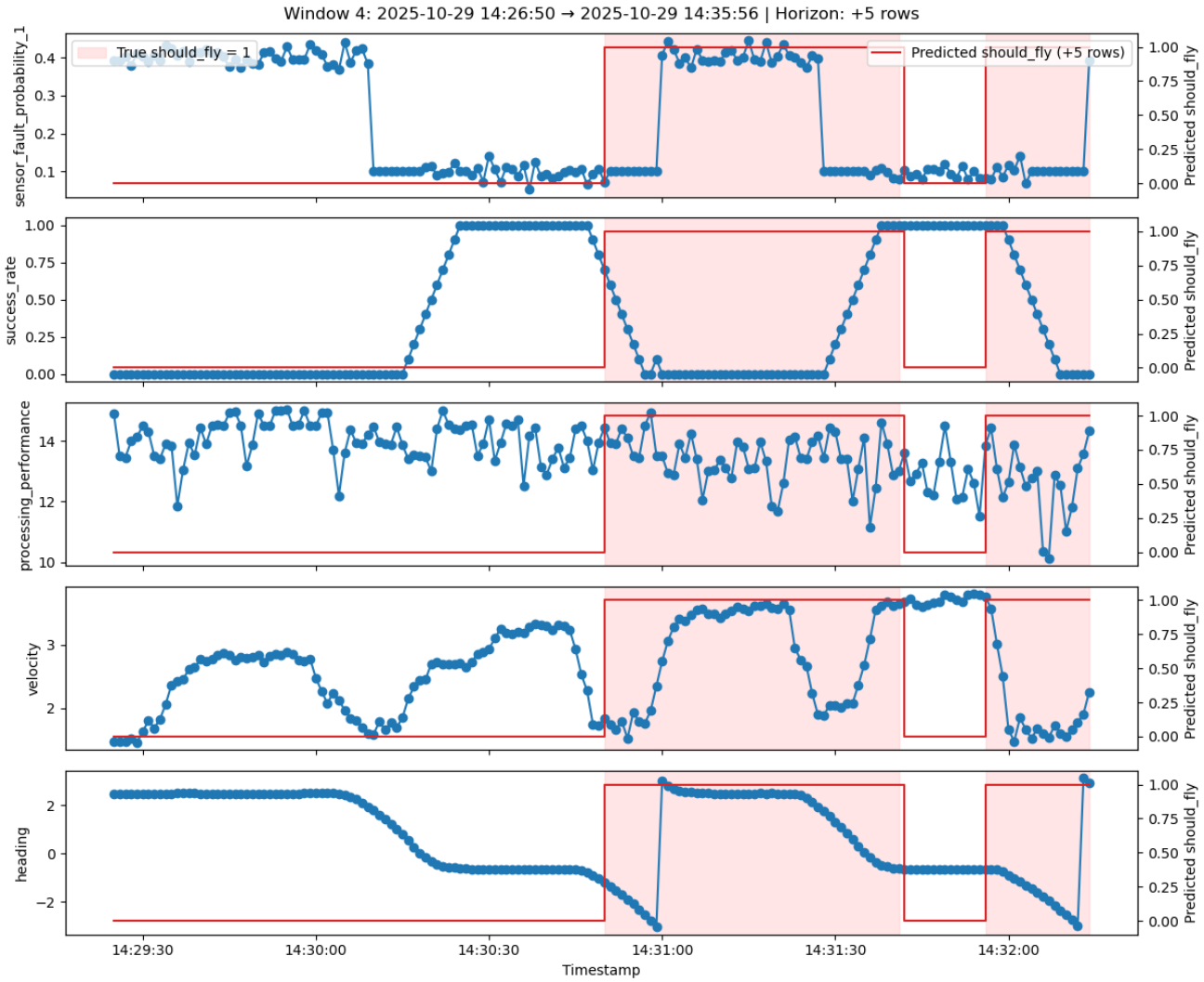

Figure 28: Prediction plot at +5 timesteps

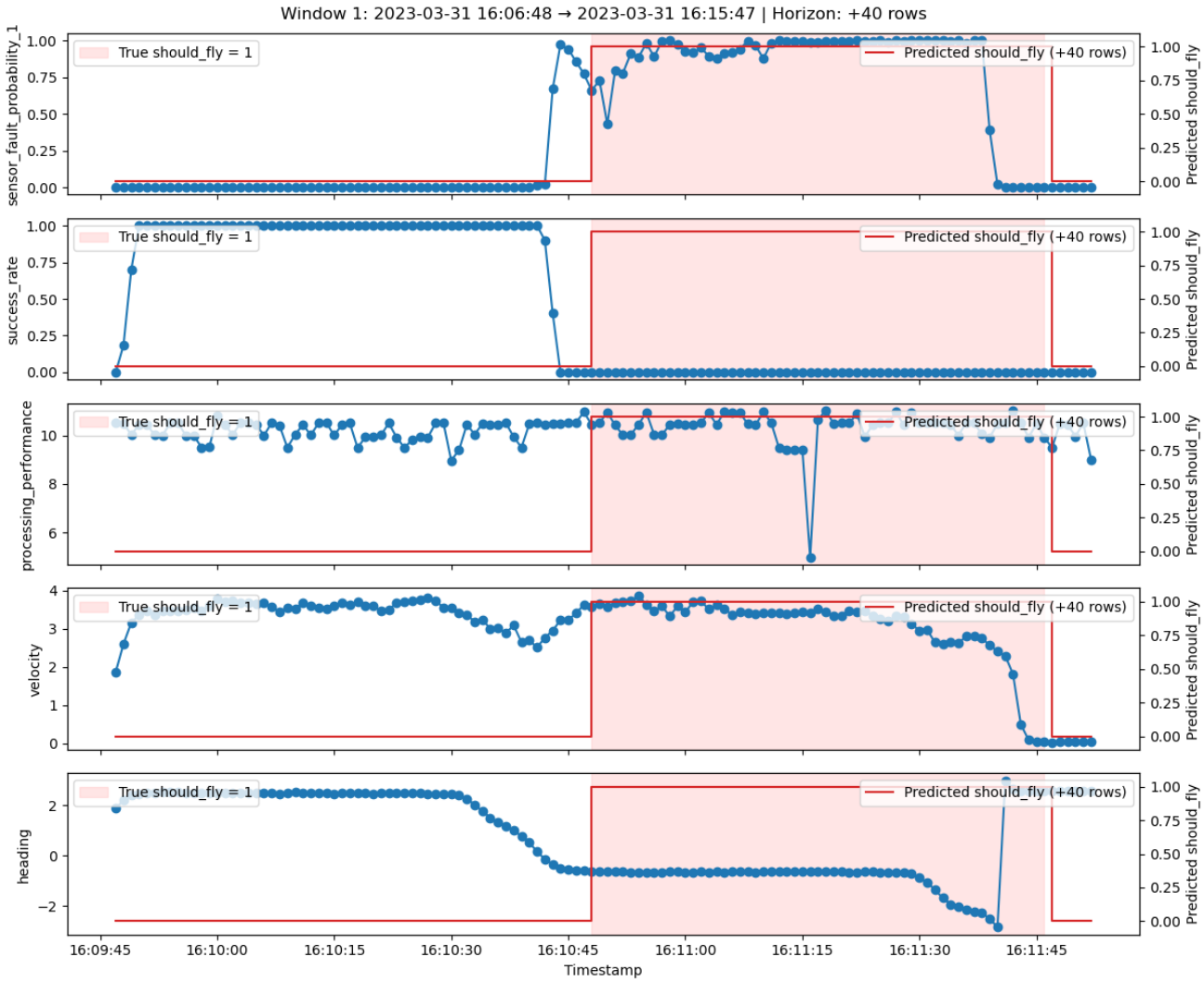

Figure 29: Prediction plot at +40 timesteps

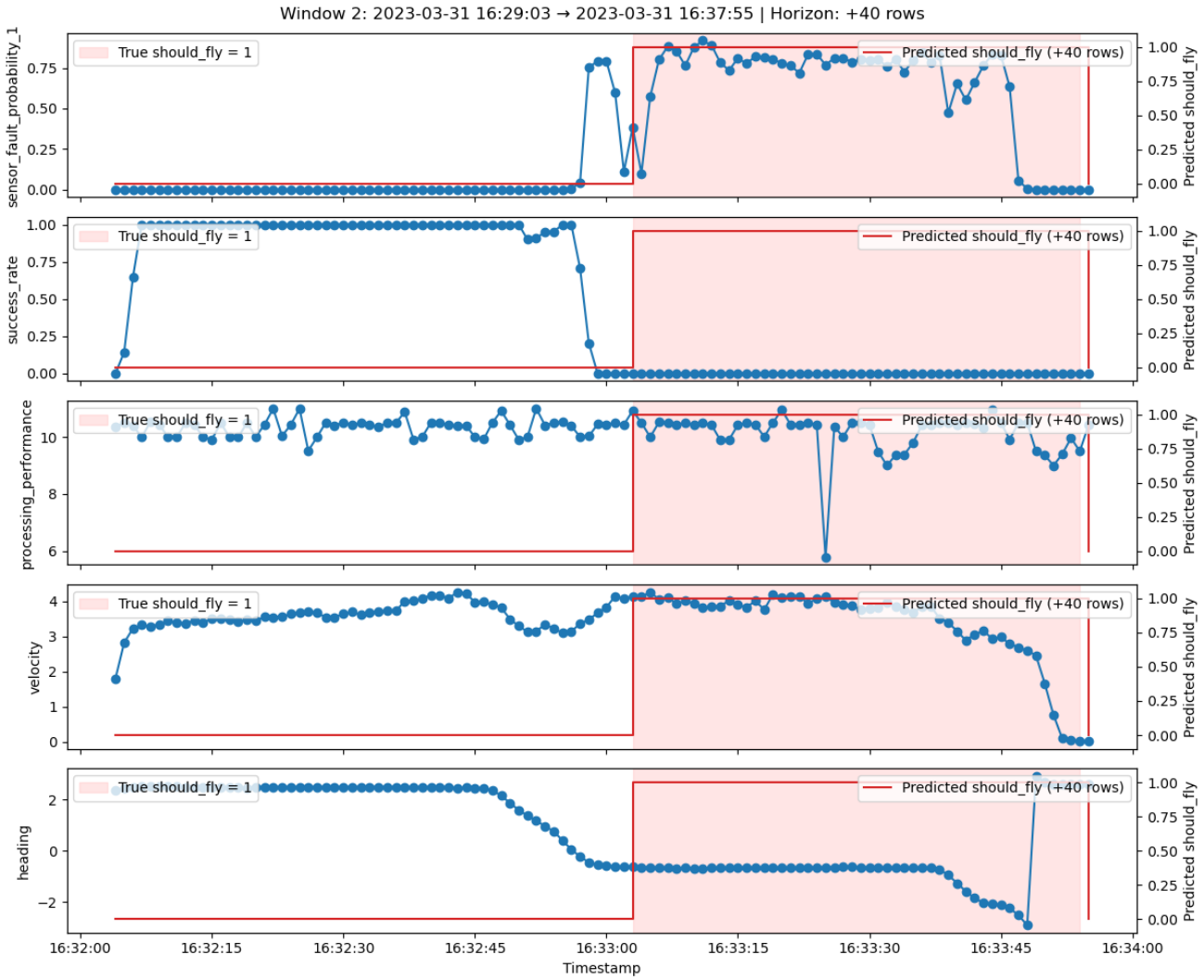

Figure 30: Prediction plot at +40 timesteps

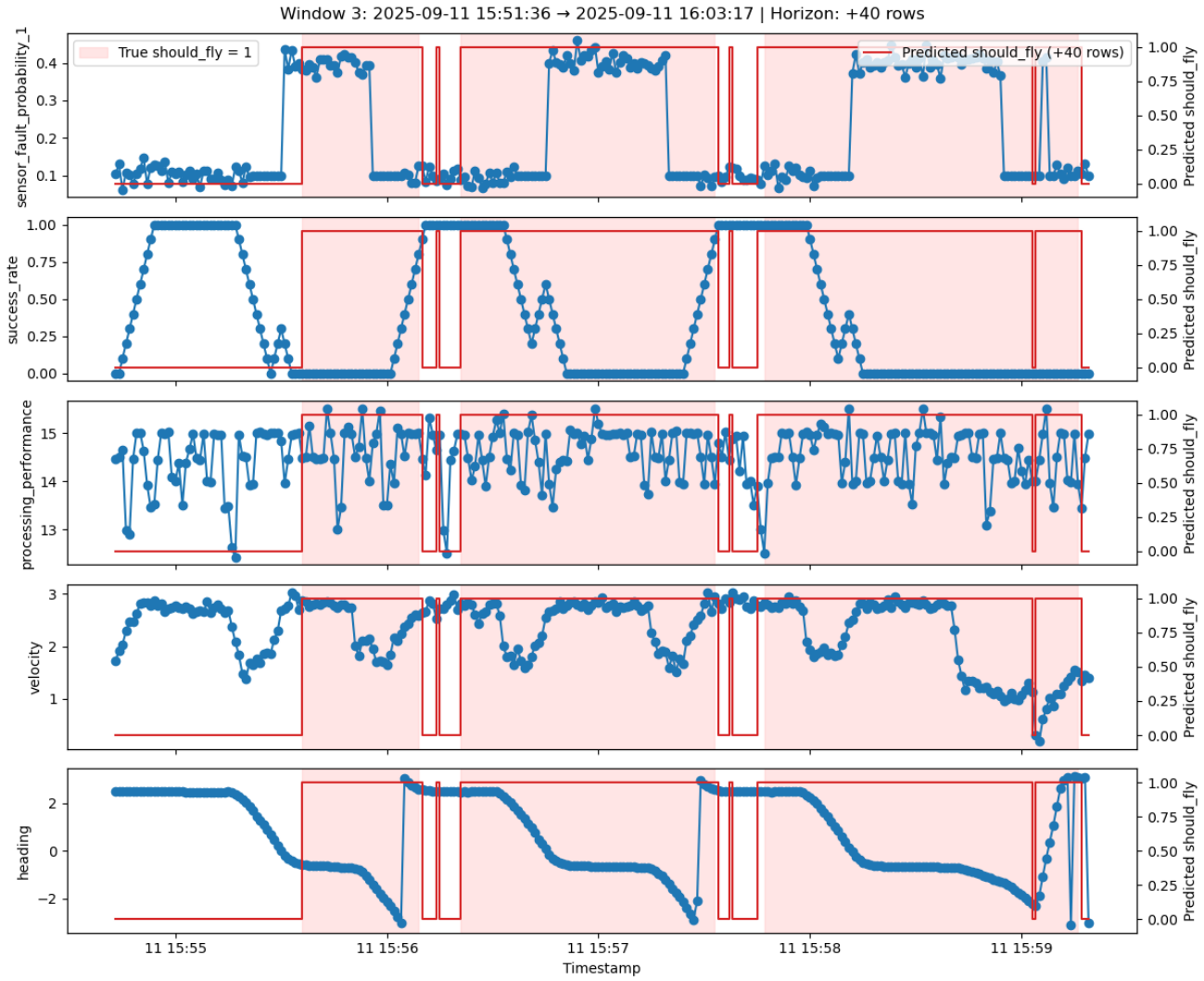

Figure 31: Prediction plot at +40 timesteps

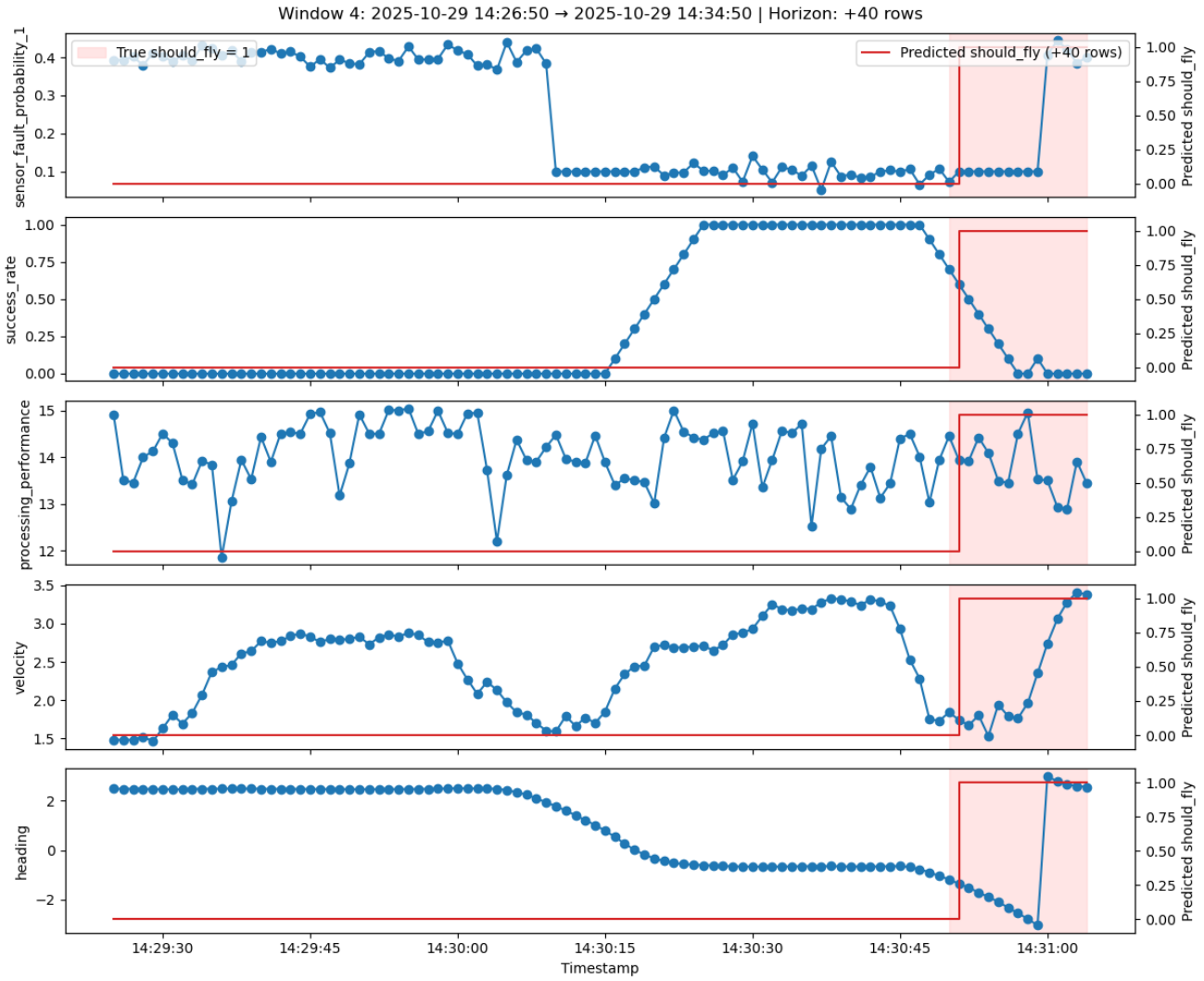

Figure 32: Prediction plot at +40 timesteps

Finally, the model predictions were visualized alongside the original signals around transition events, highlighting how prediction accuracy changes as the horizon increases. From the figures, we see that there is a potential trade-off between +5 and +40 timesteps by having more false positive predictions.

Accuracy was further analysed over a range of horizons (Figure 33), plotting mean and standard deviation across seeds to quantify how predictive power degrades as we forecast further into the future.

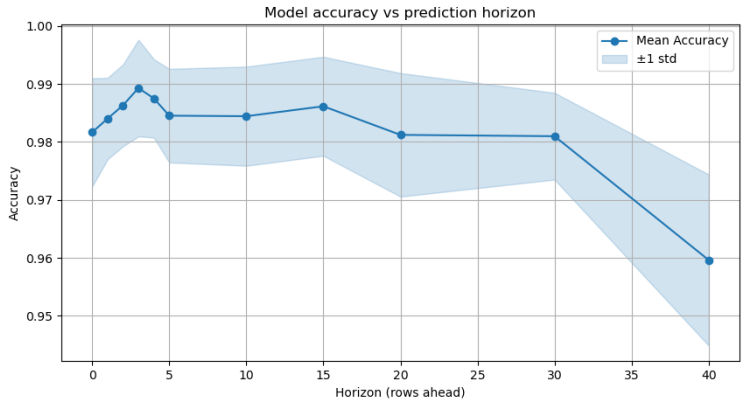

Figure 33: Model accuracy across different prediction horizons

3.3.8 Evaluation

3.3.8.1 Experiment overview

After successful desk setup experiments, the MLSysOps smart agriculture application is ready to be tested under real-world field conditions. The test was conducted before sunset in one of the AUG testbed environments that is described in the field setup (Section 3.3.5.3)

The tractor's starting position is on the western side of the field, facing east, while the drone's starting position is on the northern side of the field. The initial positions are shown in Figure 34. The tractor starts moving forward, having the sun on its back, and the drone remains on land until the tractor requires support.

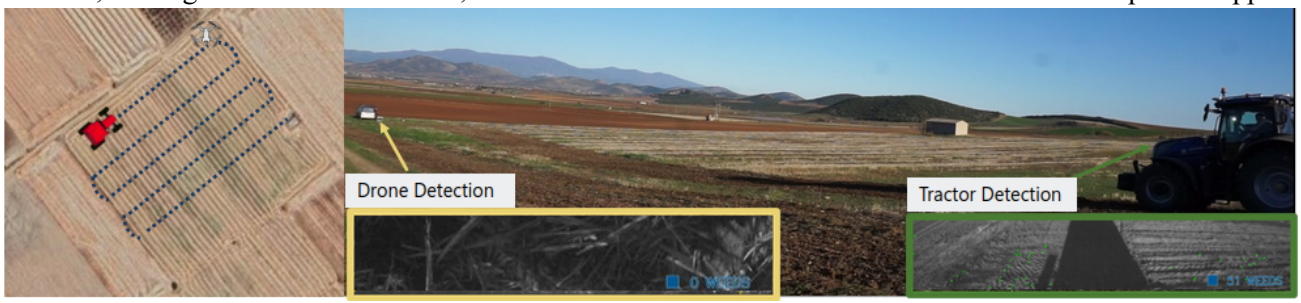

Figure 34: On the left, the tractor's trajectory; on the right, an on-site photograph showing the detection of each component. The tractor is standing on one corner of the field while the drone is on the ground.

As the tractor turns, the sun enters the cameras' field of view, causing a lens flare that degrades the weed-detection performance and triggers the drone take-off (see Figure 35). The tractor then continues forward, makes another turn, and the same sequence is repeated.

Figure 35: On the left, the tractor's trajectory; on the right, an on-site photograph showing the detection of each component. The tractor's camera is exposed to the sun, and the drone is on its mission to maximize weed detection performance.

Since the results can also be evaluated using the video-playback weed detection of AUG, video recording was enabled for both drone and tractor components, for the whole route. More specifically, a single continuous drone flight, rather than repeated take-off and landing cycles controlled by a specific ML-model instance, gives the opportunity to test and validate multiple ML-model or heuristic versions at a later stage. Additionally, having the video recordings of the entire route available and the video playback functionality (which can turn drone engagement on or off at any time) provides the opportunity to test a wide range of scenarios. More specifically, this can be used to subsequently investigate the trade-off between maximizing weed detection performance and minimizing flight time. Moreover, the availability of tractor video recordings does not require the tractor to be connected to a sprayer, allowing a wide range of spraying setups to be explored in a second phase.

Summarizing, three conditions differentiate these field tests from purely randomly selected weed-spraying operations:

1. These tests are performed during a time when light conditions hinder the weed-detection performance of the tractor. Otherwise, the drone cannot meaningfully assist the tractor. Note that farmers typically do operate during these hours.
2. The drone is constantly deployed following the tractor but does not assist it. This, subsequently, helps to fine-tune the ML model to dispatch and return the drone at the optimal time, without having to re-run the experiments for each new version.
3. The control commands to the sprayer are captured/logged, but the tractor is not actually spraying. This is typical for AUG's field tests (where it is important not to alter the field's condition) and does not affect the behaviour of the system. Importantly, this also allows for performing multiple tests on the same field over multiple days.

3.3.8.2 Results and KPI measurements

3.3.8.2.1 Application Metrics

The following metrics are defined to establish a basis for coherent evaluation:

- **Operation time:** The elapsed time from the start of the operation.
- **Safe mode duration:** The total time during which the system operated in a "blind" state due to unreliable weed detection performance, measured from the start of the operation.
- **Safe mode percentage:** The percentage of safe mode duration relative to operation time.

3.3.8.2.2 Tractor Operation

The evaluation baseline is a classic weed-spraying operation with no drone support. The operator drives the tractor in the common "Back-and-Forth" pattern as shown in the previous section. The experiment shown below (see Figure 36) was one arbitrarily picked from many similar operations.

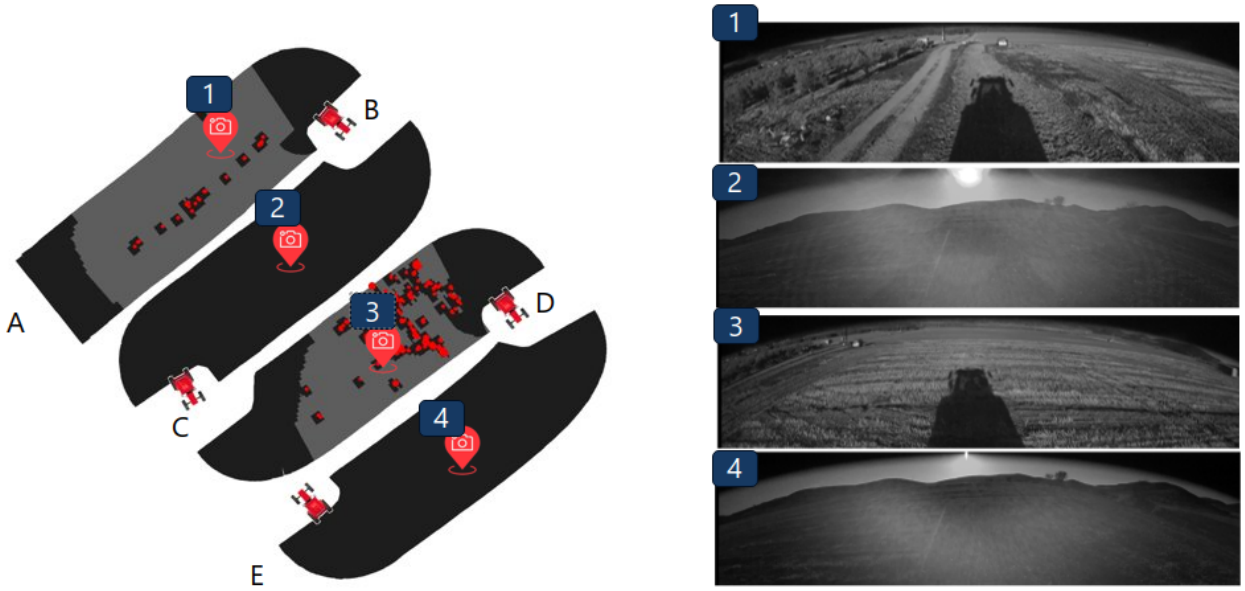

Figure 36: The left image shows a map of the tractor's operation from points A to E. Dark areas indicate sprayed regions, light grey areas indicate unsprayed areas, and red dots are the sprayed weeds. Dark areas without red dots represent safe-mode spraying, which is enabled when weed detection is unreliable. The right image shows example captures from locations 1–4 on the map. Sun creates flare artifacts on paths BC and DE (captures 2 and 4), unlike paths AB and CD (captures 1 and 3).

Table 7: Metrics accumulated up to each point of Figure 36, during tractor operation.

Pt	Operation time (sec)	Safe mode duration (sec)	Safe mode percentage (%)
A	00.00	00.00	-
B	33.34	10.15	30.44
C	69.57	46.37	66.65
D	105.82	60.67	57.33
E	144.2	99.06	68.70

As shown in Table 7, the tractor operates in safe mode for 68.7% of the total operation time. This is due to extended intervals of safe mode operation between points B (t_B = 33.34 sec) and C (t_C= 69.57 sec), as well as between points D (t_D = 105.82 sec) and E (t_E = 144.20 sec). These periods occur because of reduced weed-detection performance caused by flare artifacts, as illustrated in the figure (see Figure 36):

3.3.8.2.3 Drone Engagement and Weed-Detection Performance Improvements

There are three realistic methods to dispatch the drone to assist with weed-detection automatically:

1. **Always-on engagement** - Engage and disengage the drone when the tractor starts and stops its operation. The obvious disadvantage here is that the drone's battery will be drained regardless of whether it assists with weed-detection. In most operations, the drone will not be required as the tractor may not enter safe mode at all (see Figure 37). In the following evaluation, this method is used as a baseline since it shows the

reduction in safe mode time theoretically achievable by an optimal, oracle-like engagement method that can accurately predict when the drone is necessary. It also shows the parts of an operation where the tractor is in safe mode but still cannot be assisted by drone data.

2. **Heuristic-based engagement** - Decide on whether to engage the drone using a heuristic. For instance, a simple one would be to deploy the drone when the tractor enters safe mode (or finds itself in safe mode for an extended period). This would likely be an improvement over method 1, but it is not enough for a practical application. There is a large variety of reasons for which the system enters safe mode (some of which the drone cannot assist with) for varying lengths of time and, even if one were to categorize them, it would still be very hard to hand-implement an algorithm to predict when the system will enter safe mode in the future (a necessary feature since there is a delay from the drone deployment to its arrival in front of the tractor). Even harder would be to implement this in a scalable way that works for any field in any geographical location across the entire year. In the subsequent evaluation, a simple, yet powerful, heuristic-based drone engagement method is used as a comparison to the ML-based one: the drone will be engaged once the tractor is in safe mode, but only while it is holding a relatively steady heading. The reasoning behind the latter part is that during turns, the tractor may enter safe mode for reasons unrelated to its perception of the field and, even if it is perception-related (e.g., blinded by the sun), turns are periods where the state constantly changes and are not expected to last long.

3. **ML-based engagement** - Use an ML model that is designed to engage or disengage the drone smartly, balancing weed-detection assistance and drone battery usage. The significant advantage here is that this model can be trained to predict long periods of safe mode before they occur, ensuring the drone has enough time to reach the tractor. The model can be subsequently evaluated on the large datasets AUG has acquired from fields all over the world. The effectiveness of this solution, the MLSysOps Smart Agriculture Application UC, is evaluated under real-world field conditions as presented below.

Figure 37: Map visualization of a tractor operation with reliable weed detection performance. The tractor does not enter safe mode. Unlike the operation shown in Figure 36, this one is performed on a different field, in a straight line.

For the tractor operation shown in Figure 37, where the tractor does not enter safe mode, and the weed detection performance is reliable, the ML model described in Section 3.3.7 predicts that the drone does not need to be deployed.

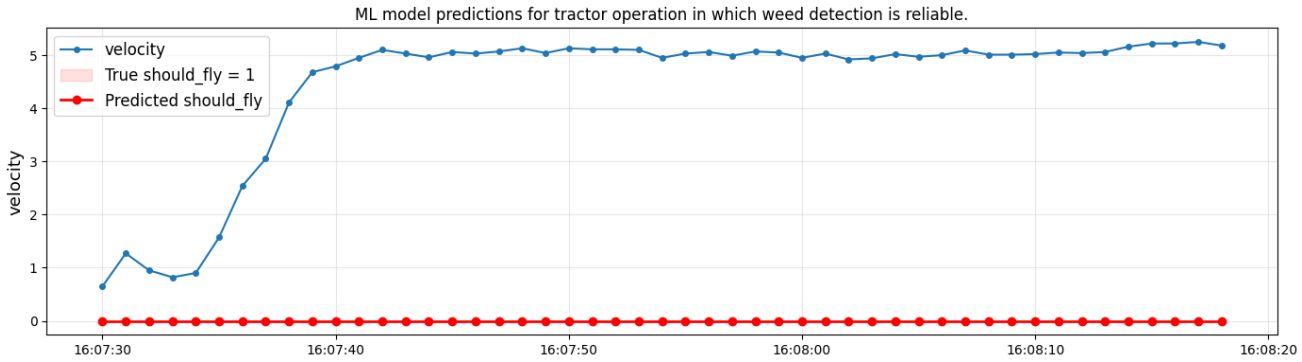

Figure 38: The ML model, as the tractor operates (and velocity increases), predicts correctly that the engagement of the drone is not needed when the weed detection of the tractor is reliable.

For the tractor operation described in the previous section (3.3.8.2.2), the ML model predictions are presented in Figure 39. The model predicts that the drone is required when the tractor is on path BC and when it is about to enter path DE. It is important to note that the ML model sends a “take off” signal some seconds before the tractor enters safe mode, considering the latency and the time for the drone to reach the tractor. The ML model used considers this delay to be 8 seconds, which is an approximation based on drone flight tests.

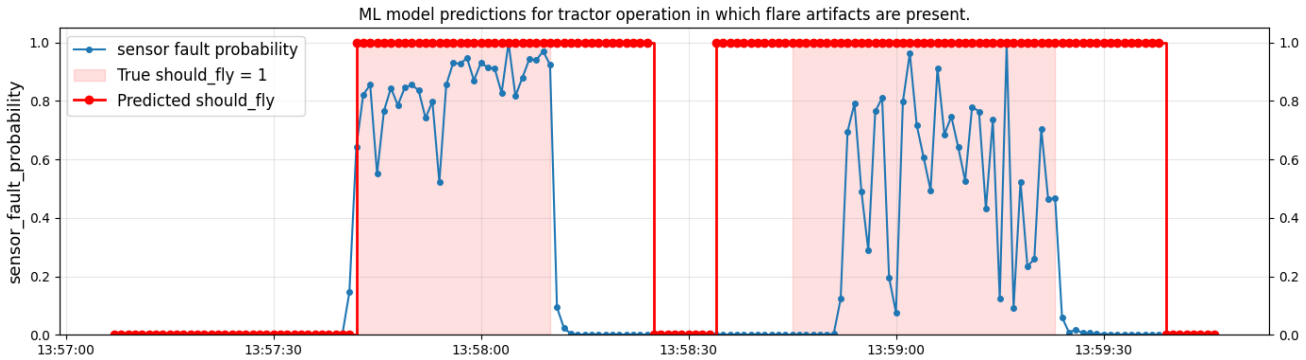

Figure 39: The ML model correctly predicts the need for drone engagement in two situations: (a) when a flare initially introduces artifacts, and (b) several seconds before the tractor enters the sun-facing trajectory.

The contribution of MLSysOps and especially its impact on tractor operation is shown below in Figure 40:

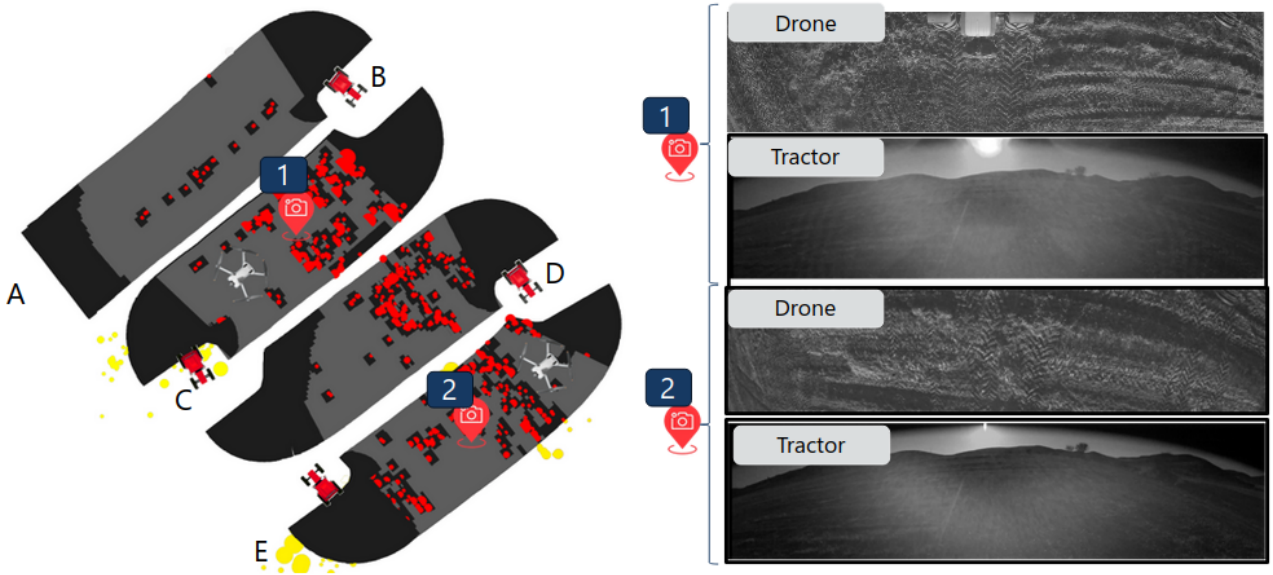

Figure 40: The left image shows a map of the tractor’s and drone’s operation from points A to E. Dark areas indicate sprayed regions, light grey areas indicate unsprayed areas to save herbicide, red dots are the sprayed weeds, and yellow dots are the drone-detected weeds. Dark areas without red dots represent safe-mode spraying, which is enabled when weed detection is unreliable. The right image shows capture from the drone and tractor at locations 1 & 2 on the map. At both sites, the tractor’s cameras exhibit sun flare artifacts, whereas the drone’s cameras do not.

Table 8: Metrics accumulated up to each point of Figure 40, during tractor and drone operation.

Pt	Operation time (sec)	Safe mode duration (sec)	Safe mode percentage (%)
A	00.00	00.00	-

B	33.34	10.15	30.44
C	69.57	24.78 (-21.59)	35.62 (-31.03)
D	105.82	37.03 (-23.64)	34.99 (-22.34)
E	144.2	50.03 (-49.03)	34.69 (-34.01)

As shown in Table 8, the tractor operates in safe mode for 34.69 % of the total operation time. (compared to 68.7% without ML drone engagement approach). This is because the drone cameras are not directly exposed to the sun, allowing for reliable weed detection. As a result, the tractor does not need to enter safe mode.

3.3.8.2.3.1 KPI – Safe Mode Time Reduction

For the evaluation session, the time that the system has satisfactory weed detection performance, thanks to the drone, is improved by **49.48%**.

For a typical 10-hour workday, with 1.5-hour "flare" periods in the morning and evening, safe mode will be enabled for at least 1.5 hours plus ~1 hour for sharp turns or unobserved areas. Based on these, the operation time with satisfactory weed detection performance is estimated as:

- Without MLSysOps (tractor only): 7.5 hours
- With MLSysOps (tractor + drone): 9.0 hours

This corresponds to a **12%** increase in effective operation time when using the drone.

Figure 41 shows the reduction in safe mode duration across the evaluation session that is achieved by the ML-based method for engaging the drone vs the two other drone engagement methods. The *always-on* method sets a theoretical lower bound but is impractical due to battery use. The *heuristic-based* method improves on the tractor alone but has a reaction delay and cannot pre-dispatch the drone for future safe mode periods.

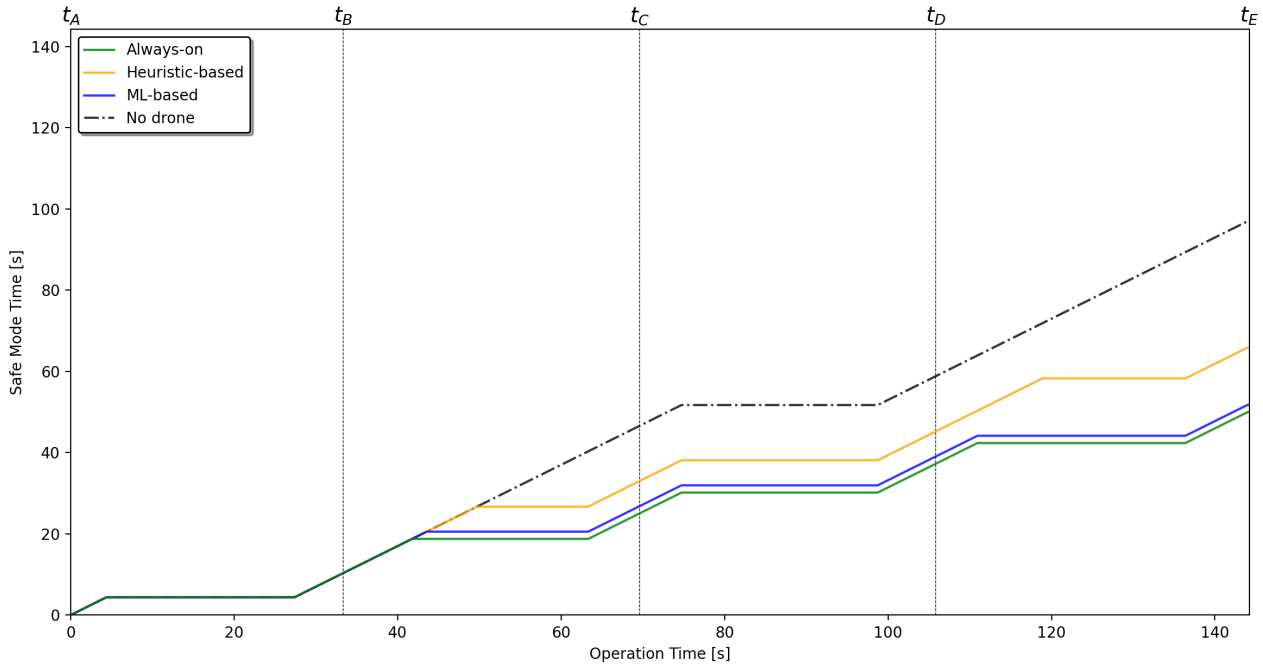

Figure 41: Safe mode time (sec) in accordance with operation time (sec) for different methods: Always-on, Heuristic-based, ML-based, and No drone engagement.

3.3.8.2.3.2 KPI – Drone Battery Usage

While there is no specific KPI metric for the battery consumption of the drone, it is important to note that the ML model attempts to balance off maximum drone support time with minimal flight time. In common terms, this is interpreted as "*fly only when necessary*". This intent is shown in Figure 42 where there are two plots showing the reduction in flight duration (and therefore reduction in battery usage) introduced by the ML model engaging and disengaging the drone.

The plot on the left corresponds to the evaluation test described above. Note that the model correctly identifies that the drone is no longer necessary after t_C and commands it to land. It then predicts the drone will be needed soon after t_D and pre-dispatches it. The plot on the right is from a session that had no issues with its weed detection, and the model correctly inferred that drone engagement was unnecessary.

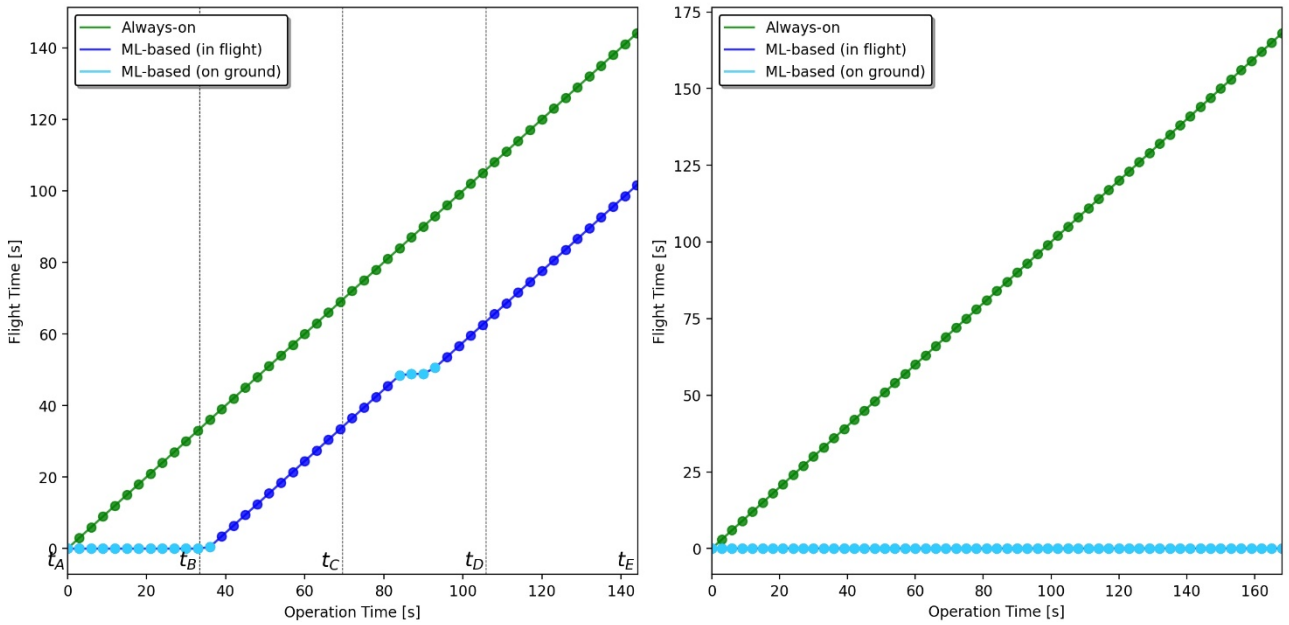

Figure 42: Flight time (sec) in accordance with operation time (sec) for different drone engagement methods: a) Always-on, b) ML-based. The plot on the left corresponds to the test shown in Figure 40, and the plot on the right a distinct test where the ML-model deemed the drone unnecessary.

Under the conditions of the evaluation session and limiting the study only to the part of the day where flare conditions can occur (e.g., near sunset), the ML-based drone engagement policy achieves a notable reduction in flight time compared to the *Always-on* policy. It is more realistic to compare the policy against a naive approach that would fly the drone continuously only during that part of the day, instead of considering the entire day as potential drone engagement time.

In the evaluation session, we managed to reduce the total flight time of the drone from 144 to 101 seconds. But this includes the beginning of the session, where the drone will not be pre-dispatched because the ML model has not yet been fed with the data it requires to make a prediction. Only counting the pre-dispatch-follow-disengage-land cycle (which is expected to be repeated multiple times in a session), we measure a reduction in flight time from 61 to 53 seconds. This corresponds to a reduction of flight time (and thereby battery usage) of 13%.

3.3.8.3 Portability, Generality & Scalability

The application software that is managed by the MLSysOps framework for the smart agriculture use case has been designed with portability as a core principle. A key feature is the shared software architecture between the

tractor and the drone. Both platforms run the same weed detection algorithm. Since the tractor system has already been validated across multiple real-world operations by AUG, this ensures consistent performance and quality when extending the weed detection solution to a drone system. This unified architecture simplifies maintenance and deployment across multiple hardware units.

The ML model that was developed to predict the performance of weed-detection and produce the drone engagement hint is designed so that it can be trained based on different datasets, not only the ones that were collected in the particular field testbed used for MLSysOps projects, but also on other datasets AUG has acquired from farmers worldwide. This allows the ML model to generalize across different crops, regions, and environmental conditions, allowing the system to scale across different geographical regions and farming practices.

Finally, application management can naturally scale across multiple systems. More specifically, multiple drone and tractor pairs in multiple clusters can operate simultaneously using the MLSysOps framework. At the cluster level, the raw (heavy) telemetry of the weed detection application component is consumed (processed by ML) locally on the tractor node. As a result, the data traffic between the tractor node and the cluster-level management logic only includes the performance predictions and drone engagement hints, which is very lightweight with an average traffic of just a few Kbps. In fact, it would be possible to adopt an even more efficient telemetry approach, where the node agent notifies the cluster agent only when the drone engagement hint changes, further reducing the node-to-cluster communication (this optimization was not pursued for debugging reasons, to have a steady periodic telemetry heartbeat between the node and the cluster). Similarly, the data traffic between the drone and the cluster-level logic merely includes the periodic status of the drone controller.

3.4 Overall Status Assessment

The *Smart City* use-case has reached a high level of technical maturity and validation, with all core application components (noise processing, image processing, and telemetry forwarder) implemented, integrated, and successfully tested across virtual, lab, and real-world urban environments. The image processing pipeline has been adapted for Arm-based edge nodes (Nvidia Jetson) and validated for high-fidelity vehicle detection. The noise processing component reliably captures acoustic events, while the telemetry forwarder seamlessly aggregates metrics for MLSysOps consumption, ensuring full observability of the edge infrastructure.

The MLSysOps framework's instantiation and agent logic are implemented and fully functional. The hierarchical agent architecture (node, cluster, and continuum) has been validated, with the cluster-level agent successfully orchestrating the activation and deactivation of the image processing component using a predictive FSM-based policy. The system demonstrates robustness to network intermittency and supports intelligent hysteresis logic to prevent rapid power cycling in high-traffic environments.

In total, comprehensive integration and testing were conducted. Virtual simulations allowed for initial stress-testing of the control plane and agent decision logic. Lab setups validated the hardware calibration and acoustic sensitivity in a controlled environment. Real-world deployments in both private parking and public city centre clusters have confirmed full end-to-end operation under varying traffic densities and environmental conditions.

Extensive data collection has been completed, resulting in a rich repository of synchronized noise and video telemetry from multiple nodes over prolonged operational periods. These datasets cover critical system performance metrics (energy, CPU/GPU load) and application of QoS metrics (detection accuracy, latency), providing a robust foundation for adaptive normalization and threshold optimization strategies.

An ML-based activation model has been successfully trained, evaluated, and optimized. The model leverages an LSTM architecture with adaptive P99 normalization to predict traffic events proactively, mitigating hardware latency. Field evaluations demonstrate substantial energy savings in private infrastructure while maintaining a capture rate above 98% when used on high traffic density scenarios.

Overall, the system is functional, validated under real-world conditions, and capable of scaling from single clusters to city-wide deployments. Future work primarily focuses on refining the ML models to enable better performance on low traffic settings and increase robustness for low noise vehicle detections.

The *Smart Agriculture* MLSysOps use-case has reached a high level of technical maturity and validation, with all core application components (tractor weed detection, drone weed detection, drone controller, and sprayer controller) implemented, integrated, and successfully tested across virtual, desk, and real-field environments. The drone controller has been validated both in simulation and on real hardware, supporting the complete operational lifecycle. Weed detection components on both tractor and drone reliably produce application metrics, exchange telemetry, and support weed location transfer. At the same time, the sprayer controller is reused unchanged from production systems, ensuring high technology readiness.

The MLSysOps framework's instantiation and agent logic are implemented and fully functional. The multi-level agent architecture (node, cluster, and continuum) has been validated, with the cluster-level agent successfully controlling drone engagement and application deployment using an FSM-based policy. The system demonstrates robustness to node availability changes and supports dynamic engagement and disengagement of the drone based on predicted weed-detection performance.

In total, comprehensive integration and testing were conducted. Virtual simulations enabled for thorough testing of control flows, telemetry, and orchestration logic. Desk setups validated real hardware, networking,

containerized deployments, and telemetry propagation. Field experiments have confirmed full end-to-end operation under real environmental conditions.

Extensive data collection has been completed, resulting in over 30 GB of synchronized video and telemetry data from multiple field sessions. These datasets cover system, application, and environmental metrics and provide a solid foundation for ML model training and future optimization.

An ML-based drone engagement model has been successfully trained and evaluated. The model accurately predicts future safe-mode conditions well in advance, enabling timely drone deployment. Field evaluations demonstrate a substantial reduction in tractor safe-mode operation, leading to a 12% increase in effective operation, confirming the positive effect of the MLSysOps approach compared to heuristic and baseline.

Overall, the system is functional, validated under real-world conditions, and able to further scale and get optimized. Future work primarily focuses on fully trained ML models with refined engagement policies and expanding large-scale field evaluations, rather than addressing fundamental system gaps.

4 Evaluation of Application-neutral Functionalities

This section presents the evaluation status with respect to the core functional requirements for MLSysOps. The information is provided in a structured way, per requirement group (RG) and for the respective KPIs, giving a short description of the work that was done related to the requirement group to achieve each KPI, the current level of achievement, and the used testbed or simulator. The rationale for such activities of simulator selection and customisation is to match the *Requirement Groups (RGs)* reported in Table 1. We use the following shorthand notation to capture the achievement level: "o" means that the KPI has not been achieved yet; "+" means that the KPI has been partially achieved; "++" means that the KPI has been fully achieved. The testbeds and simulators are presented through identifiers that can be mapped to detailed descriptions in Appendix A.

4.1 Structured System and Application Description

In WP2, work was done to define suitable system infrastructure and application descriptions. These descriptions can be used to capture various system configurations and applications that include multiple components that may also interact with each other using IP-based communication.

4.1.1 RG-KPI 1.1: System infrastructure description can capture at least the infrastructure of MLSysOps application testbeds and research testbeds

The formal infrastructure description is sufficient to capture the system infrastructure of the two application use cases (UBIW, AUG). The concrete descriptions are provided in Sections 3.2 and 3.3. More complex system descriptions have been used to capture a variety of system configurations in the various research testbeds as part of the evaluation experiments that were performed for other KPIs.

Achievement level: ++

Testbeds/Simulators used: TB-APP-1, TB-APP-2, TB-R-5, TB-R-6, TB-R-7

4.1.2 RG-KPI 1.2: Describe application components so that they can be freely placed in at least two different layers of the continuum and linked with RG-KPI 2.1 and RG-KPI 2.2

The formal application description is sufficient to capture applications with multiple different components, and for each component to have different images associated with it, depending on the target platform. Also, the description allows the expression of any deployment restrictions and the desired flexibility regarding the placement of each application component in the continuum, allowing a given component to be pinned to a cluster or a node or leave placement up to MLSysOps, in which case the component can be freely placed (and possibly relocated) in the continuum. As part of the experiments for other KPIs, extensive tests have been performed using heterogeneous system configurations to verify the flexible placement (and relocation) of a given application component in VMs, workstations, and embedded computing nodes (such as RPi and Jetson), based on the application description.

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.2 Application Deployment and Orchestration

A multi-cluster system was setup using Karmada, Kubernetes, and the Far-Edge node gateway of FhP, combining different clusters and widely heterogeneous nodes/hosts that are used to run application components, including VMs, powerful workstations, Jetson, RPi, FPGA SoC Smart-Edge nodes, and Arm-M4 microcontroller Far-Edge nodes. The RPi and Jetson nodes are equipped with a 4G interface to connect to the Internet wirelessly and be able to play the role of mobile nodes. The RPi, Jetson, and workstation are equipped

with a Wi-Fi interface, which is used as an alternative communication channel. Moreover, there are system configurations that include emulated Smart-Edge nodes and Far-Edge nodes, which appear and are used as proper nodes in the respective clusters, together with real nodes. Also, mobile nodes are emulated using the drone simulation environment of UTH (D4.3).

This setup was used to perform a wide range of experiments regarding application deployment and dynamic adaptation at runtime. In addition, extensive tests have been performed for the two application use cases using virtual/emulated nodes. Extensive tests were also done using a combination of emulated nodes and real nodes.

4.2.1 RG-KPI 2.1: Deploy an application with at least three components so that at least one component is placed at the Far-Edge, Smart-Edge, and Edge/Cloud Infrastructure

Tests were performed to confirm the successful deployment of a distributed application composed of at least three distinct components, each placed on a different layer of the computing continuum. Specifically, components of video processing, noise and temperature monitoring applications were deployed. One component was deployed in a virtual machine within the Edge/Cloud layer, responsible for video and image processing tasks. A second component was deployed on a Raspberry Pi Smart-Edge node, handling intermediate video analytics and sensor data aggregation. A third component was deployed on a Kallisto Far-Edge node, performing low-level temperature and noise data acquisition and preprocessing on resource-constrained hardware. These tests confirmed the ability to execute coordinated, multi-component applications across heterogeneous nodes spanning all infrastructure layers, with components communicating and operating seamlessly as part of a unified distributed workflow.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-7

4.2.2 RG-KPI 2.2: Deploy application components on at least two types of Smart-Edge and two types of Far-Edge nodes featuring different CPUs/ Micro Controller Units (MCU) and/or sensors

Tests were conducted to confirm the successful deployment of application components on at least two types of Smart-Edge nodes and two types of Far-Edge nodes, featuring different CPUs, microcontroller units, and sensing capabilities. Application components were deployed on three different Smart-Edge platforms, including Raspberry Pi, NVIDIA Jetson, and FPGA-based SoC systems, as well as on two variations of Kallisto Far-Edge nodes equipped with different sensors and radio interfaces, namely a Kallisto node with a temperature sensor and IEEE 802.15.4 radio, and a Kallisto node with a noise sensor and Wi-Fi radio. The deployed components included computer vision pipelines based on OpenCV for video and image processing, YOLO-based object detection models, image classification CNN models, and lightweight processing components for temperature and noise sensing executed on microcontroller-based devices. These experiments demonstrated the portability and adaptability of application components across heterogeneous Smart-Edge and Far-Edge platforms with diverse computational capabilities, sensors, and communication interfaces.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-7

4.2.3 RG-KPI 2.3: Have a deployment where at least two application components can interact with each other over either 4G/Internet, Wi-Fi, IEEE 802.15.4, or Bluetooth links

Tests have been performed to confirm the successful interaction between application components over the default network path (which is also used for the control and telemetry plane of MLSysOps) over Ethernet and 4G. Tests have been performed to confirm the successful interaction between application components running on Smart-Edge nodes over Wi-Fi using both ad-hoc and infrastructure modes. Furthermore, tests have been

conducted to verify the ability to switch dynamically, at runtime, application traffic between different network interfaces, and more specifically, between Ethernet/4G to Wi-Fi and vice versa. Notably, Far-Edge nodes communicate with the rest of the system over Wi-Fi or 6LoWPAN (via the Far-Edge node gateway), in which case Wi-Fi or 6LoWPAN is used not just for the application traffic but also for the control/telemetry plane.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-7

4.2.4 RG-KPI 2.4: The initial deployment plan is close to optimal, i.e., within 10% vs. a plan produced by an offline/oracle algorithm

Tests have been performed for an indicative application consisting of two interacting components; the client component is pinned on a Smart-Edge (mobile) node, while the service-providing component is allowed to be freely placed on edge nodes or in a VM running in a server cluster. A Reinforcement Learning policy has been developed, based on the Advantage Actor-Critic (A2C) algorithm, to select the host with the minimum recorded latency (based on pre-recorded data). The model accuracy is 91% compared to an oracle policy that always selects the node with the minimum recorded latency.

In larger-scale simulated scenarios, the initial deployment logic is handled by the same sophisticated engines that drive the adaptive behaviour. For example, CLEAR employs a Deep Q-Network (DQN) reinforcement learning agent to optimize serverless execution. The core policy dynamically tunes container "keep-alive" durations on a per-invocation basis, rather than relying on fixed timeouts. Guided by a multi-objective reward function, the agent learns to balance cold start latency against the carbon footprint of idle resources, adapting its strategy in real-time to workload patterns and user-defined preferences for performance versus sustainability. CLEAR employs a trained Deep Q-Network (DQN) agent that makes placement decisions for every function invocation. Since the system's cumulative performance across full trace simulations tracks closely with an optimal Oracle (as detailed in RG-KPI 2.5), the initial deployment decisions—which constitute the start of these traces—are validated as part of this near-optimal trajectory.

The initial (so-called default) deployment is predefined for the two application use cases. No ML is required for this.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, SIM-2

4.2.5 RG-KPI 2.5: When changes in system state and application execution profile are detected, produce/execute an adapted deployment plan close to optimal, i.e., within 10% vs. a plan produced by an offline/oracle algorithm

Tests have been performed for the application described above (two application components, one pinned on a Smart-Edge mobile node and one that can be freely placed in the continuum). In this case, adaptation concerns the placement of the non-anchored component and the redirection of application traffic between 4G and Wi-Fi, with the objective of improving end-to-end latency. Adaptive deployment has been tested using a simple heuristic and a more intelligent data-driven heuristic that learns from previous experience.

In simulated scale-out experiments, we have quantitatively validated the optimality of the system's adaptation using real-world traces. PeakLife utilizes a deep surrogate model with an encoder-decoder architecture to jointly forecast three critical metrics: average CPU utilization, peak CPU utilization, and remaining VM lifetime. This predictive capability drives a proactive migration policy that employs an adaptive utilization threshold tuned to predicted workload burstiness. By applying a cost-based VM selection strategy that filters out short-lived instances, the policy effectively minimizes unnecessary migrations and service level objective (SLO) violations

compared to static heuristics. The PeakLife framework was evaluated against an Oracle strategy that assumes perfect knowledge of future resource requirements. PeakLife's adaptive migration policy matched the Oracle's performance within 7.47% for migration counts and 1.13% for SLO violations, proving the adaptation plan is nearly optimal.

Similarly, the CLEAR framework was tested against the Oracle scheduler over diverse workload traces. The RL agent's adaptive decisions for keep-alive and placement incurred only a 6.18% increase in carbon costs and a 7.2% increase in cold start counts compared to the optimum achieved by the Oracle. These results confirm that the system's dynamic adaptation to changing system states consistently falls within the targeted 10% margin of an optimal offline algorithm.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, SIM-2

4.3 Node-level Resource Usage & Management

A broad spectrum of heterogeneous nodes is currently supported so that they can be used as part of a system slice that is managed by MLSysOps, including server machines, powerful workstations, NVIDIA Jetson, RPi, FPGA SoC, and Arm-M4 microcontroller nodes. Depending on the node type, different settings/configurations are supported so that MLSysOps can exploit them to achieve energy-efficient processing. Also, support for the transparent exploitation of hardware-based acceleration is achieved via vAccel (NUBIS), which allows applications to invoke accelerated function implementations in a portable way across heterogeneous nodes. More details on these experiments are shown in Section 3 of D3.2.

4.3.1 RG-KPI 3.1: Different combinations of power configurations spanning the energy efficiency space are supported for at least one type of Cloud/Edge Infrastructure node, Smart-Edge node, and Far-Edge node

Tests have been performed to confirm the ability to set the CPU/GPU frequency on the datacentre server, workstation, Jetson, and RPi. Tests have been performed to confirm the ability to dynamically set the number of cores that will be used to run a workload/application component on the server machine. The systems encompass a heterogeneous range of architectures, such as x86, ARM, NVIDIA GPUs, and FPGA reconfigurable platforms. Moreover, Far-Edge nodes expose a power-related knob that allows switching between different radio transmission power levels.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-7

4.3.2 RG-KPI 3.2: Offer at least three different function implementations (CPU Only, GPU, Field Programmable Gate Arrays (FPGA)) that are transparently invoked by at least two different application components

Tests have been performed to confirm the end-to-end invocation of a core image processing function (Optical Flow based on OpenCV libraries) supported by two different CPU and GPU implementations. Tests have also been performed to confirm the deployment and execution of application components that invoke this function on four different types of nodes: workstation (x86 CPU and GPU), Jetson (Arm CPU and NVIDIA GPU), Xilinx ZCU102 MPSoC (Arm CPU, FPGA), and RPi (Arm CPU).

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.3.3 RG-KPI 3.3: The performance overhead for the transparent usage of the acceleration hardware should be low (< 5% vs. a hardwired invocation of the respective implementation)

The overhead of vAccel for native execution (no remote plugin/network involved) has been measured to be less than 3%. These preliminary results were shared at a conference, and a more complete evaluation of vAccel has been conducted during the third year of the project, showing clear trade-offs between local execution (no acceleration) and remote execution (both CPU and GPU execution).

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.3.4 RG-KPI 3.4: Offer acceleration support for at least one high-level ML framework (TensorFlow or PyTorch) for inference and training

We have initial vAccel support for basic Tensorflow operations (SessionLoad, SessionRun) and Torch's jitLoadForward operation. We are working on the implementation of various models with Torch (Bert, Resnet, MobileNet, etc.). We have abstracted both Tensorflow and Torch APIs to a simple model_load(), model_run() flow, and, thus, it can encompass any similar high-level framework that follows this abstraction. The mechanism and its evaluation are released as open source through the vAccel software framework.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-6 NUBIS

4.3.5 RG-KPI 3.5: The initial node level and local application configuration are close to optimal, within 10% vs. a configuration produced by an offline/oracle algorithm

We have developed a custom reinforcement learning (RL) algorithm designed to identify the most efficient execution configuration by analysing neural network characteristics and available system resources. Specifically, we applied this approach to optimize core allocation on a multi-core CPU for Convolutional Neural Network (CNN) inference. Our experiments with this offline-trained model demonstrate high precision; it achieves performance levels identical to or comparable to manually optimized configurations, with minimal deviation.

In addition, we have developed a novel management framework driven by a custom RL agent that dynamically optimizes FPGA configurations. By leveraging real-time telemetry—including system utilization, power consumption, and application performance—the agent determines the optimal task-to-hardware mapping for incoming CNN inference tasks. Experimental evaluation on the Xilinx Zynq UltraScale+ MPSoC ZCU102 demonstrates that the framework achieves, on average, 95% of the optimal energy efficiency across several CNN models.

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.3.6 RG-KPI 3.6: When changes in the local node state and application execution profile are detected, the adapted configuration falls within a 10% margin vs. a plan produced by an offline/oracle algorithm

The same RL models referred to in RG-KPI 3.5 are deployed online and continuously monitor the system using real-time telemetry data. When changes in the system state or workload requirements are detected (for example, a new CNN model is deployed for inference), the RL agent dynamically selects an FPGA configuration plan that is near optimal for this new workload. Evaluation results confirm that these adaptive configuration decisions consistently keep system efficiency within a 10% margin of the best-known efficient configuration.

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.4 Storage

The following KPIs have been identified for the MLSysOps Storage system. All KPIs have been validated in the UTH testbed.

4.4.1 RG-KPI 4.1: Integrate at least 20 cloud storage locations across at least 4 commercial cloud providers

The basic functionality of an S3-compatible object storage service has been completed; it supports (among others) the following S3 endpoints:

- Create Bucket
- List bucket contents
- Delete bucket
- Create object
- Retrieve object
- Delete object

A notable detail is how heterogeneous storage resources are represented and made available for the end user. The storage service introduces the concept of a "Storage Policy", which contains the erasure coding configuration (e.g., total and redundant number of fragments), as well as the storage location identifiers where the fragments should be placed. When a bucket is created, a Storage Policy is attached to it, which defines how objects in the bucket are encoded and where their fragments are distributed. The listed storage locations may include any combination of cloud and edge resources.

Regarding cloud locations, we have completed integrations with 110 regions of 13 commercial cloud providers from which users can choose. Edge storage can be included by deploying MinIO[13] instances in the MLSysOps slice and sharing their credentials with the storage service. The storage gateway is already able to use these resources.

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.4.2 RG-KPI 4.2: Share availability and performance measurements with the (ML-driven) policies within 60 minutes of the data transfer.

The storage system records the performance (latency and speed) of every fragment upload and download that occurred in an MLSysOps-enabled bucket synchronously to an InfluxDB table. This information is immediately available to the ML component after the up/download completed.

Achievement level: ++

Testbeds/Simulators used: TB-R-5

[13] https://min.io/docs/minio/linux/operations/install-deploy-manage/deploy-minio-single-node-single-drive.html

4.4.3 RG-KPI 4.3: Record availability and performance of all cloud and edge storage locations at least once every 6 hours.

This KPI was achieved through an automated, continuously running measurement pipeline. Every six hours, a scheduled cron job enumerates all configured cloud and edge storage locations and creates measurement tasks for the upcoming six-hour window. For each storage location, a uniformly random execution time within that window is generated. This design intentionally avoids fixed measurement times, preventing systematic bias and ensuring that performance and availability are observed under varying network and load conditions throughout the day.

The generated tasks are placed into a task queue with their predefined launch times. When dispatched, each task performs a standardized set of uploads and download tests against the target storage location using randomly generated files of 1 MB, 10 MB, and 50 MB. For every transfer, the file size and elapsed time in milliseconds are recorded, allowing throughput to be derived, while failures are explicitly logged to build an availability time series. All results are persisted in a BigQuery table, enabling full SQL-based analysis, aggregation, and long-term tracking of storage availability and performance across all locations.

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.4.4 RG-KPI 4.4: Share file access events with the (ML-driven) policies within 15 minutes of the event.

While RG-KPI 4.2 focuses on telemetry describing the performance characteristics of cloud and edge storage locations (e.g., upload and download throughput), RG-KPI 4.4 addresses the timely collection of usage telemetry for buckets where adaptive storage policies are enabled. Specifically, we record object download events to inform the ML components about changes in traffic volume and access patterns, such as shifts in geographic distribution or request frequency that may warrant policy adaptation.

Each download event is recorded immediately after the object contents have been successfully served, using a background task within the request handler to avoid impacting request latency. Events are written to a BigQuery table and include the event timestamp, bucket and object identifiers, and the location of the gateway that handled the request. Rather than recording IP-based user locations, we log the gateway location to preserve user privacy. Gateways are exposed via a single service URL and use anycast DNS, ensuring that requests are routed to the geographically closest gateway. As a result, the recorded gateway location provides an accurate approximation of the request origin without introducing privacy concerns, while still enabling timely (well within 15 minutes) and effective feedback to the ML-driven policy engine.

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.4.5 RG-KPI 4.5: Realize the storage representation changes decided by the (ML-driven) policies in a maximum of 15 minutes per MB of data affected.

We have decomposed this complex problem into 3 individual parts that work together to identify when and how a bucket's Storage Policy should be changed in response to significant changes either in the storage location performance or bucket traffic:

- Predict bucket traffic origin and magnitude based on past traffic and other factors like time of day
- Predict storage location performance based on past performance and other factors like time of day, type (cloud or edge), and cloud provider
- Generate alternative Storage Policies, choose the most suitable one, and decide whether the migration from the current to the new policy is worth migrating

We have developed an ML-based solution that proposes the optimal Storage Policy for a bucket. A Policy is suggested based on the traffic distribution in the examined time window (currently past 15 minutes), as well as high-level restrictions (transfer speed and latency targets), geo-fence (e.g., only store fragments in whitelisted countries), and an arbitrary mix of optimization goals between cheapest transfer, geographical proximity to traffic origins and storage locations that have the highest bandwidth to the traffic origins.

We have extended the mechanism of changing the storage representation of objects in a bucket when a new optimal Storage Policy differs from the one currently applied to the bucket. The final subsystem can change a bucket between any two Storage Policies, including changing erasure coding configuration and/or the fragment locations. The process is guaranteed to generate the least amount of data transfer when carrying out the bucket migration. From the user's perspective, this migration is a transparent atomic background process that either fully succeeds or makes no changes. The bucket stays fully operational during the migration, and the new object representations are atomically swapped at the end by updating the object metadata in a single database transaction after all data movement has taken place.

Achievement level: ++

Testbeds/Simulators used: TB-R-5

4.5 Trust

We have validated on the UTH testbed. Besides, we use Raspberry Pi 5 (8GB) as the simulator hardware since it includes an onboard power monitoring sensor, which means we can measure power without adding any external measurement hardware. For storage, we use a SanDisk High/Max Endurance microSD card as a solid and reliable choice for sustained operation. Finally, we use two units so we can validate multi-node scenarios, such as connectivity and interaction between devices.

4.5.1 RG-KPI 5.1: Reputation/credit calculation is performed in real-time, within a few milliseconds. The calculation aims to consume <=5% of the energy consumption during normal application execution. The bandwidth cost is similar to the cost of a normal application communication message (with or without authentication, the bandwidth performance remains similar). Furthermore, resource and application allocation and policy adjustments related to credit should be performed on the same scale as above.

To fulfil this RG-KPI with an effective and deployable trust evaluation mechanism, we implemented an online anomaly-detection-driven trust scoring module based on a lightweight, fully connected autoencoder. Telemetry features are normalised using Min–Max parameters estimated from aggregated training data that concatenates traces collected under multiple workload regimes (idle/medium/high CPU crossed with idle/medium/high memory), reducing sensitivity to benign load changes and improving robustness under dynamic edge conditions. The autoencoder compresses the input feature vector through successive linear layers down to 0.1× of the original feature size. It reconstructs it, and the per-sample anomaly score is computed as the reconstruction error (RMSE) between the input and the reconstruction. The threshold is derived from the training error distribution using a percentile-based cut-off. The anomaly score is mapped to an anomaly probability via a sigmoid function and converted to an instantaneous benign signal, which is then smoothed with exponential smoothing to obtain a bounded trust score T(t) in [0,1] that evolves gradually and is resilient to noise spikes.

In our edge testbed (two Raspberry Pi 5 nodes), the end-to-end trust score update latency is 7.4 ms (median) and 9.9 ms (90^{th} percentile) per telemetry window, meeting the "few milliseconds" requirement, and the end-to-end reaction time from a score change to an enforced allocation/policy update is 9.8 ms (median) and 15.5 ms (90^{th} percentile). We also profiled the additional energy overhead introduced by feature extraction, plus autoencoder inference, plus trust score update, and observed 3.6% average overhead with 4.7% worst-case

overhead versus baseline execution, remaining below the 5% target. Trust dissemination is performed using compact, event-driven delta updates (plus low-rate heartbeat), keeping added bandwidth close to normal control signalling. The observed added bandwidth is 6.2% relative to baseline traffic, which is comparable to a typical application control message and consistent with the KPI requirement.

Achievement level: ++

Testbeds/Simulators used: TB-R-8

4.5.2 RG-KPI 5.2: The authentication should be 100% accurate, i.e., once an entity passes the authentication, it should be 100% what it claims to be.

To fulfil this RG-KPI, we implemented a Zero-Trust Architecture using Istio Service Mesh, ensuring encrypted, authenticated, and protected communication from the Smart-Edge layer to the Edge and Cloud infrastructures. Istio employs mutual Transport Layer Security (mTLS) to encrypt all inter-service traffic, verifying the identities of both sender and receiver to prevent unauthorised access and ensure data confidentiality.

Istio dynamically manages identities by integrating with Certificate Authorities (CA) to issue and rotate cryptographic certificates. Access controls are enforced through fine-grained policies, defining which services can interact and under what conditions. As new nodes or services are introduced, Istio automatically integrates them into the secure communication network, maintaining adaptability without compromising security. Additionally, Istio's monitoring tools enable real-time observation and detection of anomalies, further enhancing security.

Achievement level: ++

Testbeds/Simulators used: TB-R-8

4.5.3 RG-KPI 5.3: The time for adapting the encryption/decryption to the trust level of nodes will be a few milliseconds, and there should not be more than 5% energy cost vs normal application execution. The extra bandwidth the encryption brings will be restricted by the 15-20% expansion of the original communication data.

To satisfy this RG-KPI, we implemented trust-aware encryption adaptation on top of Istio by coupling MLSysOps trust scores with dynamic mesh security policy selection. Depending on the trust level, traffic is routed through different enforcement profiles (e.g., strict mTLS and restrictive policies for low-trust nodes while maintaining standard secure communication for normal operation). Trust changes are propagated to the mesh control layer such that the new profile becomes effective within 5.1 ms (median) and 9.7 ms (90th percentile) from a trust score update, satisfying the "few milliseconds" adaptation requirement. Figure 43 shows the results for the 90th percentile of the added latency in milliseconds compared with the baseline. The added latency is below 1 millisecond, as expected by the KPI. Using the same benchmarking setup as our latency experiments (HTTP/1.1 payload, 1000 requests per second, 1/2/4/8/16/32/64 concurrent connections with mutual TLS enabled), the 90^{th} percentile added latency compared to the baseline remains 0.86 ms, aligned with the KPI expectation. The energy overhead attributable to trust-adaptive security enforcement is 3.1% on average (4.4% worst-case) compared to normal application execution, remaining within the 5% limit. In comparison, the measured bandwidth expansion introduced by encryption/policy overhead is 14.2% for typical payload sizes and 17.8% in the worst case for small payloads, staying within the 15–20% expansion bound.

Achievement level: ++

Testbeds/Simulators used: TB-R-8

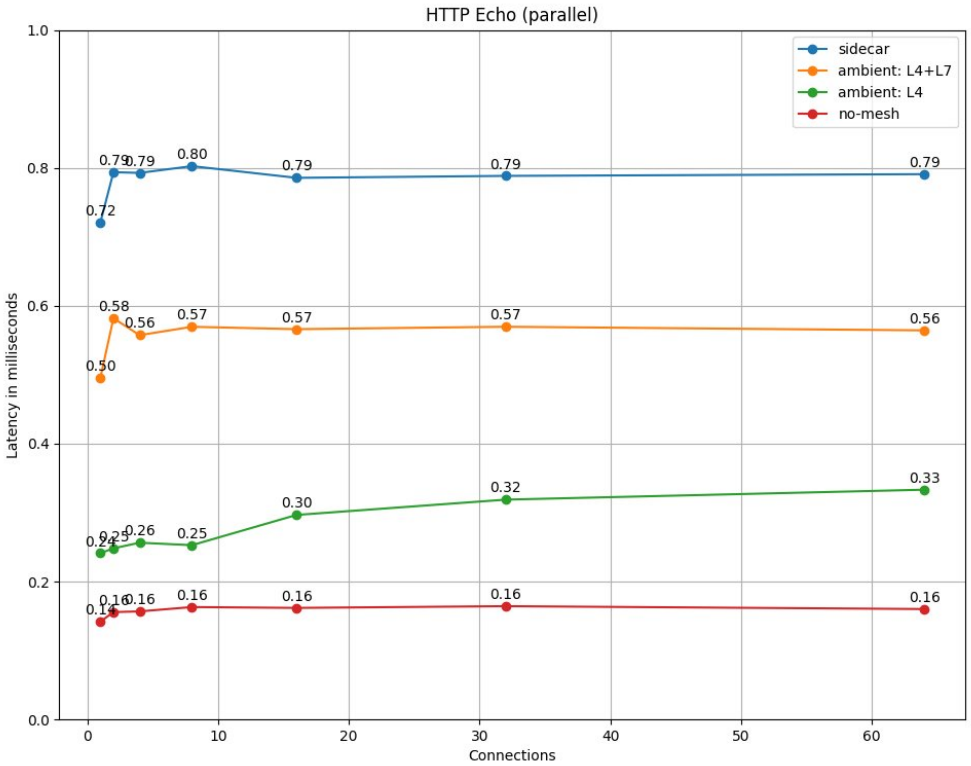

Figure 43: Latency in milliseconds vs different Istio modes.

4.6 Wireless Network Management and Security at the Edge

The following KPIs have been identified for the MLSysOps network management and security system. They specify the need to manage node connections and support application deployment and adaptation based on real-time, ML-driven evaluations of security-related parameters. This capability is important for maintaining the integrity of 5G networks in the presence of potential malicious activities, such as a power-based jamming attack. To address these requirements, we implemented an ML-driven framework designed to monitor, assess, and adapt connections and deployments based on security insights derived from targeted feature analysis. The framework has been thoroughly evaluated using predefined KPIs to ensure its effectiveness and reliability.

4.6.1 RG-KPI 6.1: Manage a network with at least 3 edge nodes (one mobile Smart-Edge node and some fixed Smart-Edge nodes) with latency kept below 10 milliseconds and Packet Delivery Ratio (PDR) higher than 90%.

The framework was evaluated in a network setup comprising at least three edge nodes, including one mobile Smart-Edge node and multiple fixed Smart-Edge nodes, which compose our 5G network environment. The 5G Core (5GC) was implemented using the open-source Open5GS project. Two Nuand bladeRF 2.0 micro SDRs served as the physical layer transceivers. One SDR was configured to operate as the gNB, while the second was dedicated to acting as a jammer. A COTS OnePlus Nord 2T 5G smartphone was used as the UE, ensuring that the data collected from the device side accurately reflects the behaviour and logging mechanisms of a real-world consumer product. The system was designed to maintain strict performance metrics, ensuring end-to-end latency remained below 10 milliseconds, and Packet Delivery Ratio (PDR) exceeded 90%. This was established through trace monitoring from the Next Generation Node B (gNB) and User Equipment (UE) side. Simulation results demonstrate that the framework consistently met the latency and PDR requirements across various workloads

and mobility scenarios, validating its robustness and adaptability in managing Smart-Edge networks under stringent performance constraints.

Achievement Level: ++

Testbeds/Simulators used: TB-R-2

4.6.2 RG-KPI 6.2: Manage a network with at least 3 edge nodes (one mobile Smart-Edge node and some fixed Smart-Edge nodes) with anomalies detected with an accuracy higher than 90% and detection time lower than 100 milliseconds.

During the primary analysis phase, we examined several approaches to meet this RG-KPI effectively. Initially, we hypothesized a detection framework that applies to multiple wireless technologies through analysis of general network metrics for anomalies and then identifies attacks based on predefined patterns. However, this approach presented key challenges: the dependence on generalised metrics limited the adaptability to specific wireless scenarios and the ability to detect nuanced attacks such as power-based jamming. To overcome the limitations, we developed a more specialised feature extraction model tailored to the 5G technology environment. In this approach, we focused on extracting targeted security-related features from the 5G UE and gNB logs and traces. These features included metrics such as Channel Quality Indicator (CQI), Modulation and Coding Scheme (MCS), PDR, and Uplink/Downlink power level parameters indicative of jamming and signal interference.

The extracted features were given as input to several ML models trained specifically to detect power-based jamming attacks. By utilising these targeted, granular features, the models were trained to distinguish between normal fluctuations and security anomalies with a high degree of accuracy.

As shown in Figure 44 a comparative evaluation of four ML models, SVM, KNN, Gradient Boosting Decision Tree (GBDT), and Random Forest (RF), was conducted. All tested models successfully met the KPI's accuracy target of >90%. The GBDT model demonstrated strong performance by achieving the highest Accuracy at 0.96, and Precision of 0.99, while the highest Recall of 0.99 was achieved by the SVM model. Notably, the GBDT, SVM, and RnF models all registered a strong ROC_AUC score of 0.95. This performance demonstrates the practical applicability of these models in real-world scenarios, particularly in detecting jamming attacks with fewer false positives and false negatives.

In addition to accuracy, the model's processing time was evaluated to meet the KPI's detection time requirement of <100 ms. Figure 45 presents the distribution of the required processing time, broken down into 'Data Preprocessing Time', 'Inference Time', and 'Total Time'. The results demonstrate that the Total Time for detection is consistently and significantly below the 100 ms KPI target. This efficiency confirms the model's suitability for real-time or near-real-time operation, thereby satisfying the second condition of the RG-KPI.

Achievement Level: ++

Testbeds/Simulators used: TB-R-2

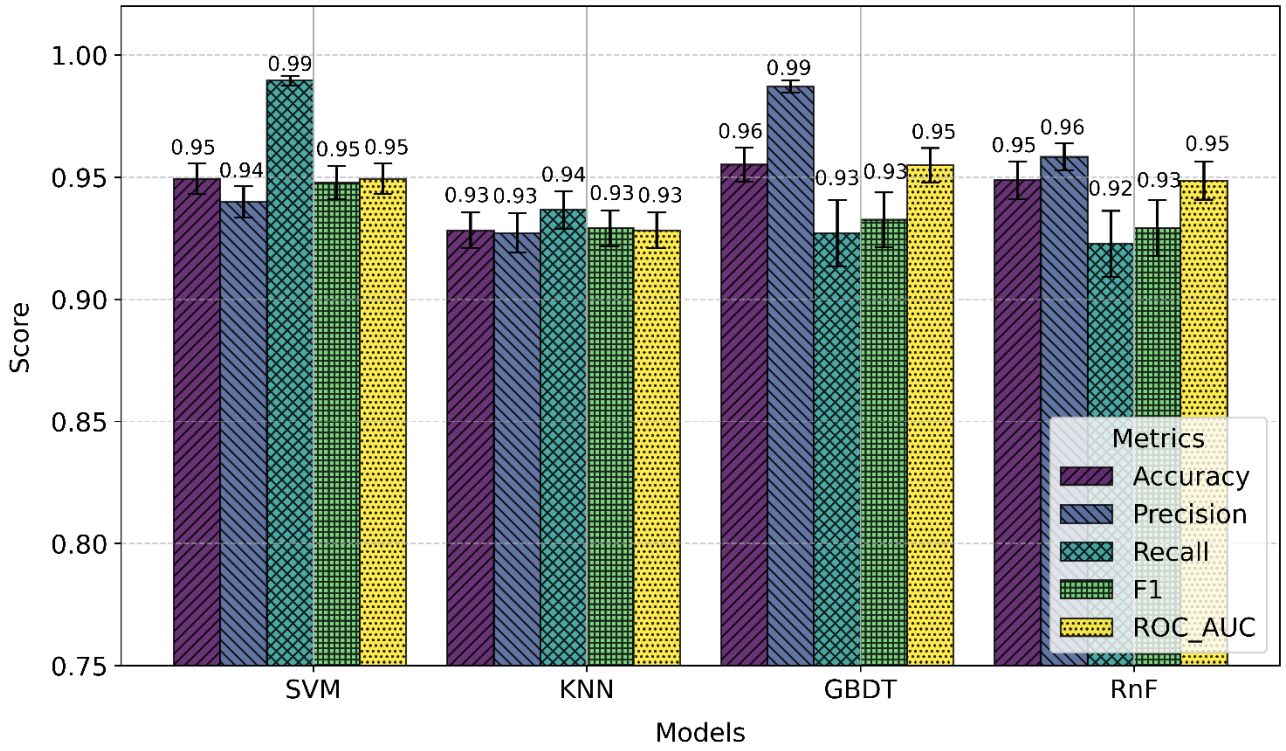

Figure 44: Performance comparison of Machine Learning models for jamming attack detection.

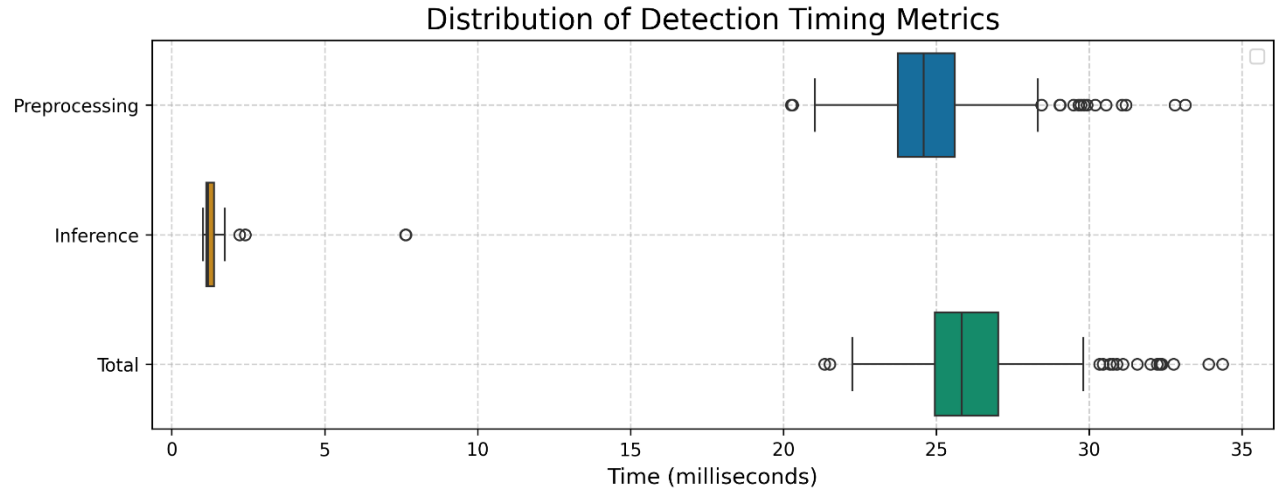

Figure 45: Distribution of online detection timing, including Data Preprocessing, Inference, and Total Time.

4.7 5G Network Management

The 5G network management identified a set of 4 requirements that relate to RG-KPI 7.1. The following section identifies the work done to achieve this RG-KPI according to each requirement that links to it.

4.7.1 RG-KPI 7.1: Latency is improved with the autonomic changes in the connectivity path between two or more interacting application components (core network NF)[14]

In the scope of R7.1 (Automate the deployment and configuration of the Network Function (NF) User Plane Functions (UPFs) at the edge (as a VM or container according to the core network implementation), providing a 5G network and edge datacentre with resource inventory capabilities), we thoroughly explored multiple strategies to address the RG-KPI 7.1 effectively. The first hypothesis we considered involved having the target datacentres without pre-deployed UPFs, with the intention of deploying UPFs on demand. However, this approach presented significant limitations: it would not allow us to gather parameters related to the resource consumption of the UPF itself, which is critical for accurate evaluation by the machine learning algorithm. Additionally, we assessed this scenario as unrealistic in practical terms, as Operators typically have edge UPFs already in place to maintain seamless network function and performance.

Given these limitations, we shifted our focus to a second hypothesis, which entailed configuring the UPF by utilising network slicing capabilities. Through this approach, we not only satisfied the requirement but also enabled continuous monitoring of resource inventory capabilities via an integrated agent. This configuration automatically implemented a 5G network slicing setup as outlined in requirement R7.4, ensuring adaptability and resource efficiency across the network. This second approach offers the significant advantage of being future proof, as most metrics collected by our implemented agent are aligned with 3GPP specifications for counters in the 5G Standalone (SA) core network. Although most vendors have yet to implement these counters fully, having the UPFs pre-deployed enables future utilisation of these 3GPP-defined counters as they become available. This foresight ensures that our system can readily adapt to forthcoming industry standards and vendor implementations, enhancing both compatibility and network observability over time.

Regarding the UPF switch, along with its data path change (R7.2: Changes to the 5G core network led to zero or very small downtime), we achieved it by modifying the network slice allocated to the end user. Following this modification, we issued a detach request to the User Equipment (UE) to prompt the use of the newly configured slice. Currently, Athonet does not implement the "re-attach required" cause, meaning we must rely on the UE to initiate a new connection.

To manage this, we utilised a Teltonika device, leveraging the ability to check the connection status every minute and reestablish it if a connection is unavailable. This approach effectively minimised downtime, keeping it below one minute.

We investigated this area by deliberately selecting the network-initiated deregistration (detach) request as the preferred mechanism to manage UE compatibility issues in multi-slice scenarios. This choice was driven by the current limited maturity of commercial UEs in supporting simultaneous or dynamic handling of multiple 5G network slices, which was observed to be inconsistent and often vendor dependent. Based on the procedures defined in Section 5.5.2.3.4 of 3GPP TS 24.501, we analysed the use of deregistration requests—specifically those requiring re-registration—as a controlled and deterministic method to enforce slice re-selection by the UE. By explicitly modifying the allowed slice configuration and triggering a detach, the UE is forced to re-attach to the network under updated slice selection conditions, effectively compensating for the immature multi-slice capabilities currently exhibited by UEs.

In the scope of R7.3 (Identify metrics from the 5G network and infrastructure data of available nodes in the continuum for target deployment to place the local data breakout for User Plane traffic optimally), we identified

[14] The original KPI referred, as an example, to the latency between Milan and London. In our environment, considering the existing latency conditions, we achieved an improvement of approximately 10 ms by taking into account a UPF deployment located within a territory such as Italy.

all the necessary metrics. We designed an agent to retrieve them directly from the network. Initially, we considered using metrics specified by 3GPP standards from the core network; however, we opted to deploy an external agent instead. This decision was driven by the fact that many core network vendors have not yet implemented these counters, making an external agent a more immediate and reliable solution. This approach allowed us to ensure comprehensive data collection without dependency on vendor-specific implementations, enhancing the consistency and accuracy of our monitoring capabilities.

Regarding the dynamic adjustment of the network slice allocated to the UE, related to R7.4 (Enable gNBs to be connected to a non-co-located, centralised UPF and switch among UPFs if ML models deem this to offer better performance), we developed, as outlined above, an API to interface with the Athonet core network, specifically targeting the Session Management Function (SMF). This approach effectively triggers a change in the data path within the user plane, which then aligns with the UPF designated in the network slice configuration. Additionally, using the same API, we issue a detach request to the UE, prompting it to reconnect to the network and thereby refresh its network slice configuration as specified (also refer to requirement R7.2). This process ensures seamless adaptability in data path routing and enhances alignment with network slice configurations.

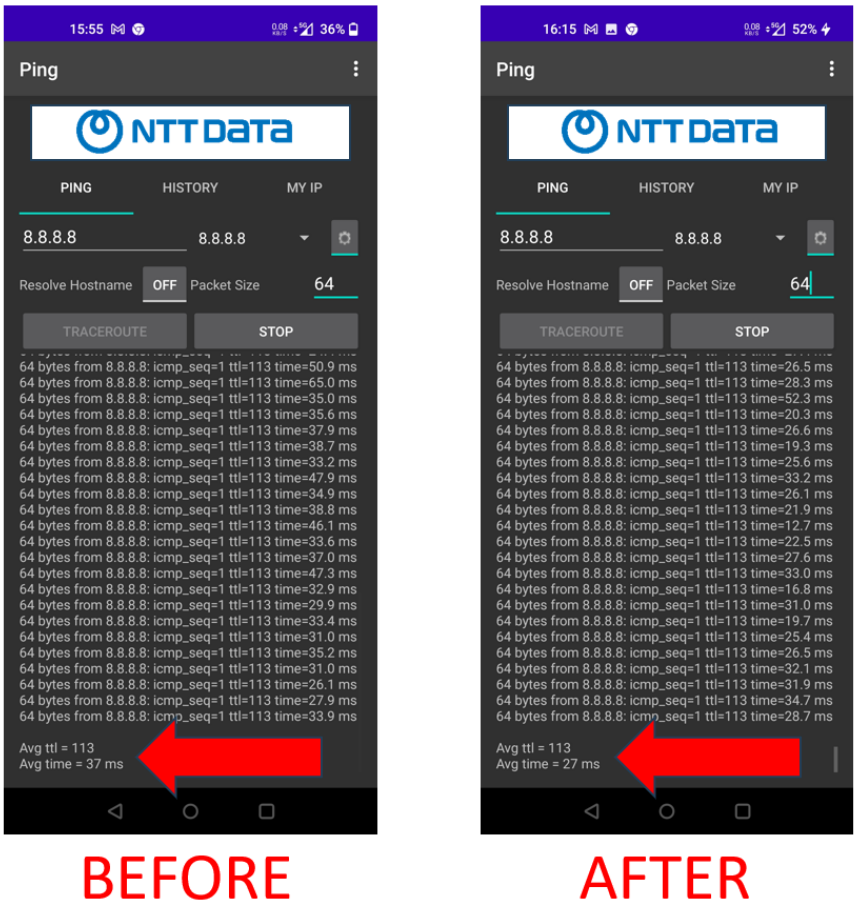

Figure 46: Improvement of 10 ms in user experience.

These developments allowed us to achieve a reduced local data breakout latency successfully, and the RG-KPI has been verified. Figure 46 shows an improvement of 10 ms in user experience.

Achievement Level: ++

Testbeds/Simulators used: TB-R-1

4.8 Optical Networking in the Datacentre

The following are the RG-KPIs for the optical network functionality within MLSysOps:

4.8.1 RG-KPI 8.1: The network management policy dynamically switches between at least two network topologies (Fat-Tree, 3D-Torus) depending on the arriving workload mix, excluding switching layers that are not required and switching off the relevant devices.

The network management algorithms have been developed, and the topology shift has been carried out for different mixtures of computations. Initially, the algorithms were verified in a simulation that used exact models. The simulator was extended with datacentre-in-the-loop capability, where the algorithms configure a small-scale cluster instead of the simulated cluster design. The jobs are launched and retired by leveraging the Slurm job scheduler, and an appropriate service handles the configuration of Optical Circuit Switches. As a result, packet switches that are excluded are switched off.

Achievement level: ++

Testbeds/Simulators used: TB-R-3

4.8.2 RG-KPI 8.2: The network management policy dynamically changes at least one parameter of the physical network (e.g., bandwidth steering) (R8.1).

The Bandwidth Steering approach is demonstrated directly in the real small-scale datacentre setup and not in simulation mode. The reason is that it requires the Adaptive Routing functionality of the Infiniband transport to take advantage of making available multiple physical paths towards the same destination. Therefore, this functionality is not simulated but directly demonstrated by our "full bisection bandwidth" algorithm that extends allocation of lanes to remove bandwidth tapering that is introduced by physical topology when it is required by the application.

Achievement status: ++

Testbeds/Simulators used: TB-R-3

4.8.3 RG-KPI 8.3: Use at most 2 switching layers to execute application workloads involving Deep Learning Recommendation Models (DRLM) and Large Language Models (LLM). (R8.1, R8.2)

For the current characterized workload mix, which consists of both DLRM and LLM, the simulations show that every communication path runs with at most 2 switching layers. We have extended our simulations to cover the most demanding collective operation (i.e., all-to-all) that underpins any modern AI workload. We managed to use at most 2 switching layers for this one by leveraging a ring-based all-to-all algorithm.

Achievement level: ++

Testbeds/Simulators used: TB-R-3

4.8.4 RG-KPI 8.4: The network power consumption is reduced by 30% due to the power down of the network elements of the bypassed layer. (R8.1)

Proven by simulation by counting the number of packet switch devices that are replaced by the OCS switches and factoring in the power consumption difference of each device class (packet switches rate ~2000 W while OCS switches only 12W). The simulation provides detailed savings, which depend on the placement algorithms used. We exceeded the goal of reducing 30% the network power consumption and still achieved the same performance for a variety of workloads by removing the spine core layer of the network and replacing it with an optical switch core network that connects leaf switches. That leads to removing 60% of the switches from the network core and replacing them with the very low power optical counterparts, which reduces network power to 60% which is well beyond the original goal.

Achievement level: ++

Testbeds/Simulators used: TB-R-3

4.8.5 RG-KPI 8.5: The network port-to-port latency is reduced by 25% with the use of a switching layer bypass. (R8.1, R8.2)

A latency study has been carried out, but at the transport level, the results did not make any important difference, at least for Infiniband transport. We found that each switching layer introduces 400 ns RTT latency in the path. In full-fat tree topologies, the worst case, paths that go through the core will result in 3 switching layer crossings that account for 1.2 μs. With OCS, we only have one switching layer plus 50 ns for the optical switch. In other words, the network NIC port-to-NIC port latency improvement offered by OCS-enabled topologies ranges from a factor of 2x-3x. Nevertheless, we found that packet networks like InfiniBand are designed to hide latencies of several microseconds, so the described benefit is not really reflected in the transport performance and to the application after all, despite that is there. We expect that memory bus-type connections like NVLink will exhibit better performance using OCS-based topologies because it is very latency sensitive, but this fabric investigation could not be carried out by our simulator and our testbed.

Achievement level: +

Testbeds/Simulators used: TB-R-3

4.8.6 RG-KPI 8.6: Arriving workloads are expected to be 100% accommodated (job scheduling and topology reconfiguration on an AI Cluster) based on the decisions generated by the MLSysOps framework. (R8.1, R8.2)

To ensure that arriving workloads are fully accommodated, an ML-based prediction mechanism was developed to support informed job scheduling and topology-aware resource management. The machine learning model was trained using ground-truth data generated with an NVIDIA proprietary simulator, which was used to model realistic cluster and network behaviour. At runtime, the model consumes the current cluster state and predicts whether an incoming job can be successfully placed and executed immediately, with particular emphasis on network-related resources whose availability is not deterministic prior to job placement, such as the optical uplinks required for inter-node communication. Based on these predictions, the scheduling decisions made by the MLSysOps framework dynamically adjust the execution order of incoming jobs, deferring those that are unlikely to be accommodated at the current time while prioritizing jobs that can be executed with the available resources. This predictive, feedback-driven scheduling approach reduces the time jobs spend waiting in the queue due to temporary resource unavailability.

Achievement status: ++

Testbeds/Simulators used: SIM-4

4.9 Energy-efficient and Green Computing in Datacentres

Work is currently being done on the development of ML-based policies for tackling node/VM management and task placement/migration within a datacentre. Model training and evaluation are performed using a simulated system via the CloudSim/CloudSimPlus simulators (with some extensions; see D4.3 “Final Version of System Simulators”), while the workload is generated based on public traces. At this point, the optimisation target is to reduce the number of active servers, save energy, reduce the number of migrations, and reduce the cost of SLA violations, thereby also reducing the cost of operation for the datacentre provider.

In the third year of the project, we investigated continuum-level scheduling to support task placement and management over multiple datacentres with different energy profiles; the energy consumed in each datacentre / each time period has a different carbon intensity depending on the mix of brown and green energy being produced in the given country or geographical location.

4.9.1 *RG-KPI 9.1[15]: Implement carbon-aware orchestration that dynamically adapts to spatio-temporal variations in carbon intensity*

We have validated the capability to exploit available green energy through the CLEAR-global ML-based policy. Specifically, CLEAR-global extends the CLEAR reinforcement learning model by expanding the action space to include cross-region placement decisions. The policy integrates spatio-temporal carbon intensity signals and inter-region network latency directly into its reward formulation. This allows the agent to dynamically route function invocations to geographical regions with "greener" energy grids, effectively trading off calculated network delays for significant reductions in carbon emissions. Evaluation on real-world traces confirms the system successfully shifts consumption to "greener" time windows and regions, achieving a 14.9% reduction in total carbon emissions.

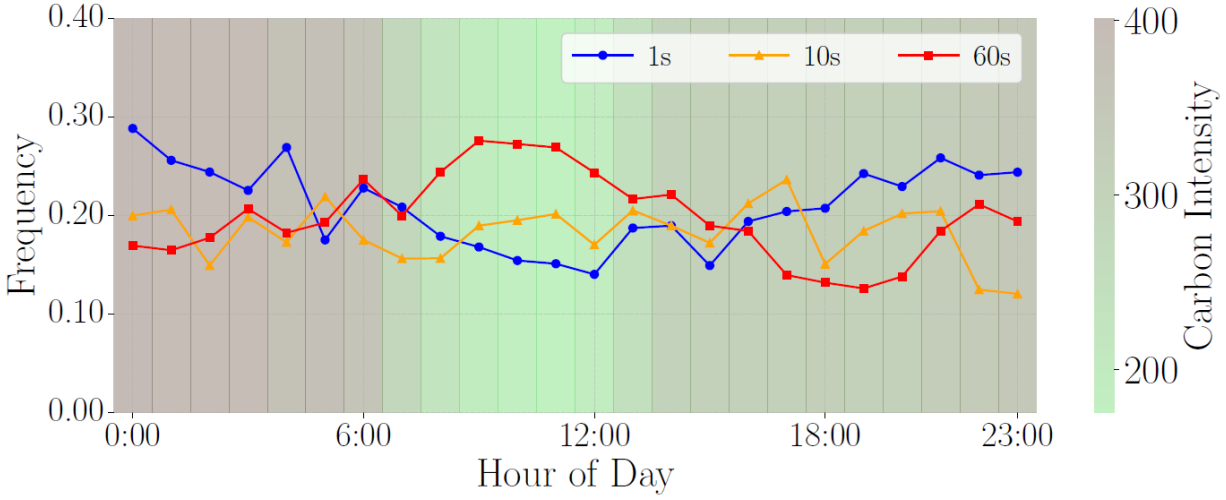

Figure 47: Temporal distribution of keep-alive durations by CLEAR. Longer durations are favoured during low-carbon hours, reflecting carbon-aware adaptation.

This exploitation is also quantitatively demonstrated by the agent's learned behaviour as depicted in Figure 47 during low-carbon hours (e.g., 08:00–12:00), the system automatically prioritizes resource-intensive actions (selecting longer keep-alive durations like 60s) to maximize utility when green energy is available. Conversely, during peak carbon intensity, it shifts to shorter durations to minimize brown energy waste. This temporal and spatial shifting confirms that the mechanism for maximizing green energy usage is fully operational.

Achievement status: ++

Testbed used: SIM-2

4.9.2 *RG-KPI 9.2 Achieve operators' costs within 5% of those of adaptive, non-ML state-of-the-art policies*

Our evaluation demonstrates that the ML-based policies achieve operational efficiencies that match or exceed state-of-the-art heuristics. PeakLife was compared against the Local Regression Minimum Migration

[15] RG-KPI 9.1 has been reformulated from "Exploit at least 75% of the green energy available to datacentres" to "Implement carbon-aware orchestration that dynamically adapts to spatio-temporal variations in carbon intensity". The original formulation refers to a metric and threshold which is hard/unrealistic to quantify given that datacentres are not connected to separate Green and Brown power supply sources but draw whichever energy is available from the grid. What varies (in time and by location) is the mix of Green and Brown energy sources used in electricity production. The new KPI captures this aspect in a clearer way and focuses on the capability developed in the project (spatio-temporal adaptation).

(LRMMT[16]) heuristic. PeakLife reduced the average number of migrations (a key operational cost driver) to 1,050 compared to 1,484 for LRMMT, a reduction of ~29% while simultaneously lowering SLO violations. Additionally, the CLEAR framework was compared against a state-of-the-art Dynamic PSO-based (DPSO[17]) metaheuristic. CLEAR achieved comparable total carbon emissions (407,19g CO2 vs 395,12g CO2) but with significantly better responsiveness (17.85 s latency vs 23.40 s). Moreover, CLEAR's inference overhead was 4600x faster than DPSO (0.96 s vs 4.52 s for the workload), making it a far more cost-effective solution for real-time deployment.

Achievement status: ++

Testbed used: SIM-2

4.10 Machine Learning

The ML functionality in MLSysOps is delivered via the MLConnector. This manages the full ML model lifecycle, from search and discovery of models, requesting the activation of an ML model (leading to the deployment of the corresponding containerised model), to invoking the ML model and requesting its deactivation (leading to the removal of the respective container). This design ensures that ML models are cleanly decoupled from the agents that utilise these models, which allows agents to switch between different ML models flexibly at runtime.

4.10.1 RG-KPI 10.1: At least two different models are used interchangeably without modifying the underlying resource management and configuration mechanisms.

The containerized deployment of ML services through the usual deployment path of MLSysOps ensures we can deploy multiple models without modifying the underlying resource management and configuration mechanisms. The dynamic model engagement and invocation are supported via the ML Connector, and the dynamic change between different ML models has been tested without issues, using 2 different versions (with minor changes) of the ML models developed for each use case.

Achievement level: ++

Testbed used: TB-R-5

4.10.2 RG-KPI 10.2[18]: The effectiveness of continual learning shall improve system performance whenever model drifting is detected (i.e., it should at least bring the system to the performance levels before model drifting).

We have designed and implemented efficient state representations for Deep Reinforcement Learning (DRL) agents managing VMs in cloud systems, focusing on faster and more robust training. After exploring the state-of-the-art (autoencoders, VQ-VAE, and graph embeddings), we employ advanced representation learning and

[16] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. 2012. Energy-aware resource allocation heuristics for efficient management of data centres for cloud computing. *Future generation computer systems* 28, 5 (2012), 755–768.

[17] Y. Jiang, R. B. Roy, B. Li, and D. Tiwari, "Ecolife: Carbon-aware serverless function scheduling for sustainable computing," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, 2024, pp. 1–15.

[18] RG-KPI 10.2 has been reformulated from "The effectiveness of continual learning shall improve system performance whenever model drifting is detected." to "The effectiveness of continual learning shall improve system performance whenever model drifting is detected (i.e., it should at least bring the system to the performance levels before model drifting)". This reformulation was done to clarify the metric used for evaluation.

feature extraction techniques to compress states into powerful latent representations. These dimensionality reduction methods create compact yet meaningful views of the environment, reducing the computational complexity of the state space and significantly accelerating training while preserving performance. Our approach incorporates domain adaptation and knowledge transfer, allowing agents to adapt to environments with varying infrastructure sizes. By leveraging knowledge distillation and domain transfer strategies, we ensure that agents retain and reuse knowledge efficiently across dynamic cloud systems, fostering adaptability and scalability. This approach was tested for VM job allocation using CloudSim. However, the approach is applicable to other mechanisms.

Achievement level: ++

Testbed used: SIM-2

4.10.3 RG-KPI 10.3: Performance isolation will be achieved between the running application and the ML model (training/inference). Application QoS targets are met, although resources are used for applications and model re-training / continual learning mechanisms.

When an agent activates an ML model through the MLConnector, the model is deployed as a special containerised application that exposes a REST endpoint used for inference. This way, ML inference is decoupled from the agent and application execution while exploiting the flexible deployment capability of the MLSysOps framework. The same approach was followed for the model drift detection and training processes. Support for scheduling the inference and training processes so as not to impact application performance was completed during the third year of the project.

Achievement level: ++

Testbed used: TB-R-5

4.10.4 RG-KPI 10.4[19]: Explanation mechanisms provide sufficient justifications in terms of feature importance for decisions made by an ML-based mechanism.

To enhance the transparency and interpretability of the model's actions, we utilise two powerful model-agnostic explainable methods: Local Interpretable Model-Agnostic Explanations (LIME[20]) and SHapley Additive exPlanations (SHAP[21]). These methods provide complementary insights into model decisions, with LIME focusing on local interpretability for individual predictions and SHAP offering global interpretability to assess the contribution of each feature across the entire dataset. These explainability mechanisms are integrated with the MLConnector.

Achievement level: ++

Testbed used: TB-R-5

---

[19] RG-KPI 10.4 has been reformulated from "Explanation mechanisms provide sufficient justifications for every decision made by an ML-based mechanism" to "Explanation mechanisms provide sufficient justifications in terms of feature importance for decisions made by an ML-based mechanism". This reformulation was done to clarify the metric used for evaluation.

20 Ribeiro, M.T., Singh, S. and Guestrin, C., 2016, August. " Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1135-1144).

21 Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 4768–4777.

4.10.5 RG-KPI 10.5: System and running applications continue working even when ML-driven decision-making is deactivated.

ML solutions are provided to agents via the MLConnector. This solution is decoupled from the rest of the application, ensuring that applications continue working even when ML-driven decision-making is deactivated, using conventional (e.g., rule-based) policies instead. The ability of agents to switch between ML-based and conventional operations has been tested using the ML models developed for the application use cases without issues.

Achievement level: ++

Testbed used: TB-R-5

4.10.6 RG-KPI 10.6[22]: All relevant telemetry data is successfully captured and processed using data-centric AI techniques to produce summaries of good quality (i.e., the resulting quality will not negatively affect model performance) that can be used as experience memory and made available for further analysis or utilisation.

The efficient state representations we developed for managing VMs in cloud systems employ data-centric AI techniques by collecting and storing useful data samples that are integral to the retraining of the agents. Although the collected information is state-specific, states are represented in a generalizable and transferable tree structure that makes them useful for retraining agents in different state-spaces. For example, once an agent is trained to assign VMs to a datacentre of 10 hosts, data collected whenever the agent is in the observation phase, are recorded using the representation tree we proposed to be used when the number of the hosts change. This approach is generalizable to other use-cases since we proposed a generic algorithm for defining the representation tree.

The UTH research testbed, comprising a diverse set of platforms and connectivity technologies—including RPi Jetson device, datacentre servers, and workstations, interconnected via 4G, Wi-Fi, and Ethernet—provided a wide range of real telemetry data and supported the execution of heterogeneous workloads. This diversity proved invaluable for experimenting with and developing ML models for system-level management decisions, enabling realistic system characterization and modelling.

Testbed used: TB-R-5, SIM-2

Achievement level: ++

[22] RG-KPI 10.6 has been reformulated from “Explanation All of the relevant telemetry data is successfully captured and processed, using data-centric AI techniques, to produce summaries of good quality that can be used as experience memory and be made available for further analysis or utilization.” to “All relevant telemetry data is successfully captured and processed using data-centric AI techniques to produce summaries of good quality (i.e., the resulting quality will not negatively affect model performance) that can be used as experience memory and made available for further analysis or utilisation”. This reformulation was done to clarify the metric used for evaluation.

5 Evaluation vs Project Level KPIs

This section outlines the evaluation status of the project KPIs identified for MLSysOps in the project proposal. The information is organised in the same structure as in Section 4, with a sub-section for each project objective and its corresponding KPIs. For each KPI, we provide a summary of the work done to achieve it and indicate the current level of achievement: "o" for not achieved, "+" for partially achieved, and "++" for fully achieved (adopting the same notation that was already used in Section 4). The testbeds and simulators are presented using identifiers that can be mapped to the corresponding descriptions provided in Appendix A.

5.1 Deliver An Open AI-ready, Agent-based Framework for Holistic, Trustworthy, Scalable, and Adaptive System Operation Across the Heterogeneous Cloud-Edge Continuum

5.1.1 P-KPI 1.1 At least 2 real-world applications transparently combine services from cloud and edge layers and/or different infrastructure providers

The Smart Agriculture use-case combines edge and cloud services through the MLSysOps framework. Application components for weed detection and drone control are executed on edge nodes deployed on the tractor and a drone. At the same time, system monitoring, orchestration, and decision-making logic are handled by MLSysOps agents operating on the physical nodes as well as at the cluster and continuum levels on cloud-hosted VMs (Google). The system integrates the edge and cloud layers through a continuous telemetry exchange and remote control, reflecting the operating conditions at runtime. This integration has been validated through real-field experiments using real hardware, confirming end-to-end operation across the system's infrastructure.

Similarly, the Smart City use-case demonstrates this capability by orchestrating services across a heterogeneous edge-datacentre continuum in two distinct real-world testbeds. The application components for noise sensing and image processing execute on diverse edge nodes (Nvidia Jetson devices) located in both a private infrastructure (UBIW headquarters) and a public city environment (Aveiro city centre). These edge services are transparently combined with orchestration, decision-making agents, and telemetry aggregation hosted on virtualized infrastructure within UBIW's local datacentre. The system seamlessly manages the interplay between local edge activation logic and centralized datacentre monitoring, validating the framework's ability to unify services across geographically distributed infrastructure layers and distinct operational domains (private vs. public).

Achievement level: ++

Testbeds/Simulators used: TB-APP-1, TB-APP-2

5.1.2 P-KPI 1.2 Deployment and orchestration on at least 4 families of devices spanning from cloud to Far-Edge

Tests have been performed to confirm the proper deployment and orchestration of application components on the following types of nodes/devices: VMs (on typical cloud-class servers), workstations, Jetson, RPi, FPGA SoC (standalone Smart-Edge nodes), and Arm-M4 microcontroller devices (Far-Edge nodes).

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-6, TB-R-7

5.1.3 P-KPI 1.3 Support for at least 2 different cloud/edge resource provisioning /allocation/orchestration frameworks

Tests have been performed to confirm the proper operation of the framework, interfaced with Karmada at the continuum level and with Kubernetes at the cluster level, to support the flexible deployment and orchestration of distributed applications based on the formal application description both across and within clusters. The MLSysOps framework is based on generic containers, rendering the system compatible with any orchestration framework that supports containers. Consequently, while Karmada and Kubernetes (K8s/K3s) constitute the preferred deployment framework, Docker Swarm remains a supported alternative for cluster and node agents. Moreover, we have implemented interfaces to Amazon Web Services enabling the integration of public cloud infrastructure to the resource slices available to MLSysOps-supported applications.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-6

5.1.4 P-KPI 1.4 Support at least 2 AI policies for any managed resource without changes to the management mechanisms

Different ML-based policies have been developed for various management aspects. The ability to employ different ML models without changing the underlying mechanisms is one of the key design features of the MLSysOps framework. This has been confirmed experimentally through agents that dynamically activate and invoke different proxy ML models via the MLConnector, also offering the option to switch between ML-based and conventional heuristic-based operations at the cluster level.

Achievement level: ++

Testbed used: SIM-2, TB-R-5

5.1.5 P-KPI 1.5 Support systems with at least 250 nodes across the continuum (validated through simulation)

The real-world application use cases and the research testbeds only have a few physical nodes. Additional nodes can be emulated using VMs on server machines and workstations (see drone simulation environment of UTH). Truly large-scale experiments will be performed using system simulators such as CloudSim (UTH), CloudSimPlus (UCD), and EdgeCloudSim (UNICAL). At this point, initial simulations are being performed focusing on datacentres and edge-based systems; with EdgeCloudSim we setup simulations with up to 1000 mobile nodes, about 30 edge datacentres/5G Base Stations, and 1 cloud datacentre. On CloudSim, we have worked with datacentres with 320 server nodes, and on CloudSimPlus, initial experiments include 1 cloud datacentre with up to 32 hosts. Last, we supported cross-datacentre management/task scheduling, where the scale is expected to reach the planned target.

Achievement level: ++

Testbeds/simulators used: SIM-1, SIM-2

5.1.6 P-KPI 1.6 Sufficiently lightweight agent implementation, targeting memory-limited (less than 1GB) Smart-Edge devices and agent coordination overhead of less than 10% of application data traffic

The current implementation of the MLSysOps agents is relying on the SPADE framework. It has been further containerized and Kubernetes-enabled to ensure reproducible and scalable deployment across the Cloud–Edge Computing Continuum. All agents are deployed as DaemonSets, allowing a consistent presence at the node, cluster, and continuum levels, while maintaining a lightweight runtime footprint suitable for resource-constrained environments.

The memory footprint of the agents was experimentally evaluated using memory usage values reported by the Kubernetes kubelet for each MLSysOps agent pod. These values are based on container-level memory usage derived from Linux control groups (cgroups). Measurements were collected during steady-state operation across the different continuum layers.

As shown in Table 9, the continuum-level agent deployed on a cloud VM exhibits a memory usage of approximately 154.56 MB, confirming that global coordination functionality can be supported with a modest memory footprint.

At the cluster level, MLSysOps cluster agents deployed on different clusters demonstrate memory footprints of approximately 199.34 MB (cluster clus2) and 151.22 MB (cluster clus1). These results indicate that cluster-level coordination logic remains lightweight, even when managing multiple node-level agents.

At the node level, MLSysOps agents deployed on individual Smart-Edge nodes show memory consumption values ranging from 143.36 MB to 221.29 MB, depending on the node platform and workload characteristics. Specifically, measured footprints include 143.36 MB (Raspberry Pi 4), 178.58 MB (Jetson Nano), and 221.29 MB (Raspberry Pi 3). All values remain well below the 1 GB memory constraint typically associated with memory-limited Smart-Edge devices, demonstrating that node-level intelligence can be executed efficiently within containerized environments.

Regarding coordination overhead, agent communication remains minimal across all layers, on the order of a few kB/s. This traffic mainly consists of heartbeats, telemetry updates, and status notifications exchanged between node-level and cluster-level agents. The relative impact of this coordination overhead depends on the specific application traffic requirements: for data-intensive applications, it is negligible, whereas for applications generating little or no traffic, the coordination overhead will naturally dominate.

Table 9 Memory footprint of MLSysOps agents across the Cloud–Edge continuum

Agent	Name	Node Type	Memory (MB)
Continuum	cloudvm	VM	154.56
Cluster	clus1	VM	151.22
Node	n-node	Jetson Nano	178.58
Node	r3-node	Raspberry Pi 3	221.29
Cluster	clus2	VM	199.34
Node	r4node	Raspberry Pi 4	143.36

Achievement level: ++

Testbeds/Simulators used: TB-R-4

5.2 Develop an AI Architecture Supporting Explainable, Efficiently Retrainable ML Models for End-To-End Autonomic System Operation in the Cloud-Edge Continuum

5.2.1 P-KPI 2.1 Explainable ML models with decision quality within 5% of traditional state-of-the-art resource management algorithms.

Explainability mechanisms are provided via two powerful model-agnostic explainable methods: Local Interpretable Model-Agnostic Explanations (LIME) and SHapley Additive exPlanations (SHAP). This approach is generalizable to any ML model. Several models have been developed, and results show that they perform better than state-of-the-art solutions. With the use of the MLConnector, other mechanisms can integrate explainability and measure the impact of the ML model and the quality of decisions made.

Achievement level: ++

Testbed used: TB-R-5

5.2.2 P-KPI 2.2[23] Eliminate the effect of model drifting without any perceivable reduction (less than or equal to 1%) in the QoS of applications in the presence of system slack. & P-KPI 2.3 Reduce the effect of model drifting at a QoS penalty of less than 5% for running applications in the presence of resource pressure.

Given the identical strategies employed for P-KPI 2.2 and P-KPI 2.3, their status analysis has been consolidated.

We have implemented an approach based on Deep RL that incorporates domain adaptation and knowledge transfer, allowing agents to adapt to environments with varying infrastructure sizes. By leveraging knowledge distillation and domain transfer strategies, we ensure that agents retain and reuse knowledge efficiently across dynamic cloud systems, fostering adaptability and scalability and hence reducing the effect of model drift. This approach is tested for VM job allocation using CloudSim+. However,, it can be generalized to other mechanisms and platforms. The developed Deep RL-based agents were trained with QoS metrics as baselines in order to make sure that there is no significant QoS penalty (i.e., more than 5%).

Achievement level: ++

Testbed used: SIM-2

5.3 Enable Efficient, Flexible, and Isolated Execution Across the Heterogeneous Continuum

5.3.1 P-KPI 3.1[24] Package and deploy in the continuum applications consisting of at least 10 components, transparently and on-demand, targeting 4 execution enclaves (VMs, microVMs, unikernels, containers) at comparable latency with state-of-the-art standalone enclave generation methods

[23] P-KPI 2.2 has been reformulated from “Eliminate the effect of model drifting without any perceivable reduction in the QoS of applications in the presence of system slack” to “Eliminate the effect of model drifting without any perceivable reduction (less or equal to 1%) in the QoS of applications in the presence of system slack”. This reformulation was done to clarify the metric used for evaluation.

[24] P-KPI 3.1 has been reformulated from “Package and deploy in the continuum applications consisting of at least 10 components, transparently and on-demand targeting 4 different execution enclaves Virtual Machines (VM), microVMs, unikernels, containers, at comparable latency with state-of-the-art standalone enclave generation methods” to “Package and deploy in the continuum applications consisting of at least 10 components, transparently and on-demand, targeting 4 execution enclaves (VMs, microVMs, unikernels, containers) at comparable latency with state-of-the-art standalone enclave generation methods (package should not exceed 60% of the generic packaging time and deployment should not exceed 10% of the generic deployment time)”. This reformulation was done to clarify the metric used for evaluation.

(package should not exceed 60% of the generic packaging time and deployment should not exceed 10% of the generic deployment time).

We use cloud-native tools for packaging applications into OCI images and deploy these images using open-source container runtimes (where we actively contribute with code and maintenance effort) that allow sandboxing applications into microVMs. In addition, we built two components that extend this functionality to more exotic execution environments, such as unikernels: (a) bima, a packaging tool that builds an OCI image from a unikernel binary, along with specific annotations that facilitate its deployment, (b) urunc, a lightweight container runtime, CRI-compatible (k8s), able to spawn unikernels. Preliminary numbers show a 30% decrease in spawn/tear-down time of a container compared to generic container runtimes (runc) and more than 2x speed-up compared to sandboxed container runtimes (kata-containers). Detailed performance measurements are reported in D3.3.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-6

5.3.2 P-KPI 3.2[25] Support Arm Cortex-M family devices as legitimate application deployment and resource orchestration targets.

The mechanisms that introduce Far-Edge devices as legitimate resources for deployment and resource orchestration are developed (details on "D3.3 Final Version of AI-ready MLSysOps Framework") and have also been successfully tested through the current integrated version of the framework. They support the Arm Cortex-M family, namely Kallisto, an Arm Cortex-M4 low-power device.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-7

5.3.3 P-KPI 3.3 Support resource-constrained devices (Class 1 as defined by IETF RFC 7228) over constrained networks (as defined by IETF RFC 7228) and reduce reprogramming time by at least 20% compared to a full firmware update.

The minimum compatibility for embServe (Far-Edge framework for application component deployment) with support for indicative application components is class 3 resource-constrained devices. There is a trade-off between the support for Class 1 resource-constrained devices and the functionality and complexity of the application components supported by embServe and the Far-Edge devices. We opted for functionality and complexity while still supporting resource-constrained devices, according to IETF RFC 7228.

With embServe, we have reduced the Far-Edge nodes' reprogramming time by 99% compared to the full firmware update[26] (0.109s for application component deployment vs 27.72s for full firmware update) using Wi-Fi. When considering resource-constrained networks, specifically 6LowPan, we have also reduced the

[25] P-KPI 3.3 has been reformulated from "Support Advanced Reduced Instruction Set Computer (RISC) Machine (Arm) Cortex-M family devices as legitimate targets for application deployment and resource orchestration" to "Support Arm Cortex-M family devices as legitimate application deployment and resource orchestration targets". This reformulation was done to correct the KPI specification, which incorrectly associated two different HW architectures (RISC and Arm) with a family of devices that is Arm-based.

[26] J. Oliveira, F. Sousa and L. Almeida, "embServe: Embedded Services for Constrained Devices," *2023 IEEE 19th International Conference on Factory Communication Systems (WFCS)*, Pavia, Italy, 2023, pp. 1-8, Doi: 10.1109/WFCS57264.2023.10144123

reprogramming time by 99% compared to the full firmware update (1.048s for application component deployment vs 502s for full firmware update).

Achievement level: ++

Testbeds/Simulators used: TB-R-7

5.3.4 P-KPI 3.4 Enable the use of at least three (3) execution devices (cloud GPU accelerators, edge CPUs, edge GPU accelerators) by a single application binary.

We build and enhance vAccel, a hardware acceleration framework that enables the decoupling of function calls from their respective hardware-specific implementations. To this end, we enable compute-intensive functions (e.g., OpenCV's Optical Flow) to be executed on diverse hardware accelerators using vAccel. Using the vAccel API, the application can transparently perform an optical flow operation with (a) no acceleration (CPU only), (b) GPU-enabled acceleration (CUDA, NVIDIA RTX), (c) edge GPU acceleration (CUDA, NVIDIA Jetson Orin AGX).

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-6

5.4 Support Green, Resource-Efficient, and Trustworthy System Operation while Satisfying Application QoS/QoE Requirements

5.4.1 P-KPI 4.1 Reduce the IT energy footprint of 2 real-world use cases by at least 20%.

For the Smart Agriculture use-case, the MLSysOps framework improves energy-related efficiency by dynamically controlling drone engagement based on predicted weed-detection performance. Energy consumption was approximated using the flight time of the drone. Evaluation based on field experiments shows that the ML-based engagement policy can strongly reduce unnecessary drone flight time compared to an always-on approach while maintaining almost as good application performance.

In the specific evaluation session, the ML-based engagement policy reduces the drone's energy consumption by approximately 13%. If the entire day is considered, this reduction will increase to over 86% (since the ML-model will make sure the drone is passive when it's unneeded). The exact savings depend on various unpredictable factors such as the field's dimensions, and the tractor's speed.

For the Smart City use-case, the MLSysOps framework enables energy-related efficiency gains by dynamically managing the power state of the Computer Vision (CV) component on smart lampposts. Measured via real-time node telemetry, the performance results vary significantly based on the deployment context. In the private testbed, the framework achieved average energy savings of ~21%, with peak reductions reaching the theoretical hardware limit of 40%. In the public testbed (City Centre), the system prioritized safety-critical reliability, resulting in more conservative energy savings of ~3-5% to ensure a near-perfect capture rate of >99%, validating the framework's ability to adapt its energy-saving to the high traffic profile of the urban environment.

Achievement level: +

Testbeds/Simulators used: TB-APP-1, TB-APP-2

5.4.2 P-KPI 4.2[27] Reduce the total operational carbon footprint of cloud/edge workloads by at least 15% compared to standard baselines.

The CLEAR-global policy reduced total carbon emissions by 14.9% (reaching 85.1% of the baseline) by exploiting cross-region green energy availability. The PeakLife policy reduced unnecessary VM migrations by 41.33%, directly lowering the operational energy and carbon overhead associated with inefficient resource management. The combined effect of these policies quantitatively meets the target reduction.

Achievement level: ++

Testbeds/simulators used: SIM-2, TB-R-5

5.4.3 P-KPI 4.3[28] Reduce network power consumption by 30% by exploiting optical circuit switching, which reduces the required number of power-hungry electrical packet switches without introducing oversubscribed or longer network paths that can affect application performance.

MLNX OCS-based network cluster simulations have shown that large-scale communication collective operations (e.g., all-reduce, scatter-all-gather, etc.), which underpin the AI training large-scale communications on AI clusters, can be flexibly supported by only a leaf layer of packet switches without either the spine or core layers of Fat tree. The approach leverages optical switches to interconnect the leaf-layer packet switches dynamically, so the final physical topology reflects the application connectivity requirements. Managing to remove the core layer of packet switches as well to support the main collective operation of all-to-all reduces the overall power consumption by more than 30%. To provide a quick example, in the mainstream topology of a full-fat tree, we need 1024 packet switches at the core layer, 2048 for the spine layer, and 2048 for the leaf layer. For an average packet switch consumption of 2000 W, this totals to a 10240 KW power requirement. Replacing the core layer and spine layer with Optical Circuit switch devices of the same 64-port radix, we are trading 3072 packet switches that consume 6155 KW for 2048 Optical circuit switches that consume 24,5 KW. The total network power of the datacentre is reduced approximately 60%, 2x better than the goal.

Achievement level: ++

Testbeds/simulators used: TB-R-3

---

[27] P-KPI 4.2 has been reformulated from “Execute cloud and edge workloads at least 50% on green energy, assuming green energy availability during 30% of the day on average in each datacentre location” to “Reduce the total operational carbon footprint of cloud/edge workloads by at least 15% compared to standard baselines. The original formulation refers to a metric and threshold which is hard/unrealistic to quantify given that datacentres are not connected to separate Green and Brown power supply sources but draw whichever energy is available from the grid. What varies (in time and by location) is the mix of Green and Brown energy sources used in electricity production. Moreover, whether the original formulation of the KPI could be achieved or not depends highly on workload characteristics, namely whether the workload consists of batch jobs which have relaxed deadlines and can be delayed, or interactive / real-time jobs which have more stringent deadlines. The new KPI focuses on the essence / end goal, which is the reduction of the carbon footprint, rather than the means/method to achieve this (how much green energy to use).

[28] P-KPI 4.3 has been reformulated from “Reduce network power consumption by 30% by exploiting optical circuit switching without increasing application-perceived latency” to “Reduce network power consumption by 30% by exploiting optical circuit switching, which reduces the required number of power-hungry electrical packet switches without introducing oversubscribed or longer network paths that can affect application performance”. This reformulation was done to clarify the metric used for evaluation.

5.4.4 P-KPI 4.4 Far-Edge network latency to gateway less than 10ms and packet failures lower than 10^{-7} with a device density of 100 devices per gateway.

We have prepared the Far-Edge gateway (NextGenGW) to handle more than 100 devices and evaluated its capability to manage application component deployment with such device density. Latency was measured considering the overhead NextGenGW interoperability feature adds to communication, disregarding the network overhead because that depends on network setup and on the number of hops between the Far-Edge device and the Node where NextGenGW is running. Considering this, with a cluster of 100 devices, the NextGenGW latency for 50 simultaneous read requests on the LwM2M object 'Device' of 50 different devices is 247 microseconds. The tests were performed with the NextGenGW running on an Intel NUC 13 Pro equipped with an Intel Core i3-1315U CPU, 16GB of RAM, and a 256GB NVMe SSD, running Ubuntu 22.04 LTS. We selected the LwM2M object 'Device' for the tests because it is the largest object in Kallisto, comprising 17 properties.

Achievement level: ++

Testbeds/Simulators used: TB-R-7

5.4.5 P-KPI 4.5 True positive rate of network anomaly detection higher than 90% and permitted false positive rate lower than 5% for critical applications.

The system leverages machine learning (ML) algorithms to analyse features extracted from network logs, such as Signal-to-Noise-Interference Ratio (SNIR), Received Signal Strength Indicator (RSSI), Packet Delivery Ratio (PDR), and bit rate, to identify patterns indicative of network anomalies. The ML-driven detection module achieved a true positive rate greater than 90% for detecting network anomalies while maintaining a false positive rate below 5% for critical applications. These results validate the model's capability to reliably detect jamming attacks while minimising false alarms, ensuring actionable insights that trigger adaptation processes without unnecessary disruptions. A more sophisticated machine/deep learning model has been developed and trained on larger and more diverse datasets. The aim was to improve the accuracy and efficiency of real-time or near real-time anomaly and attack detection, optimising the system's performance in practical scenarios. More specifically a Generative Adversarial Network (GAN) has been proposed and we implemented GANSec, published at ESORICS conference 2025. The main advantages of GANSec concern its generalisation capability. Indeed, models trained exclusively on GANSec-generated data significantly outperformed all baseline methods on the unseen data.

Achievement level: +

Testbeds/simulators used: TB-R-2

5.4.6 P-KPI 4.6 Reduced local data breakout latency at the edge by 10 milliseconds on a fully cloudified standalone 5G.

The P-KPI 4.6 focused on automating the deployment and configuration of the User Plane Function (UPF) at the network edge, which was addressed through a thorough evaluation of different strategies. The first approach considered involved deploying UPFs on demand in datacentres without pre-deployed UPFs. However, this solution presented significant limitations. It did not allow the collection of critical parameters related to UPF resource consumption, which are essential for the accurate functioning of machine learning algorithms. Additionally, this strategy was deemed impractical, as operators typically have pre-deployed UPFs at the edge to ensure the continuity of network functions and performance.

Given these challenges, we shifted our focus to a second approach, which leveraged the network's slicing capabilities to configure pre-deployed UPFs. This approach not only met the project's requirements but also enabled continuous resource monitoring through an integrated agent. The implemented configuration automated

the setup of 5G network slicing, as outlined in Requirement R7.4, ensuring efficient resource utilisation and network adaptability. A further advantage of this solution was its forward-looking design: most of the metrics collected by the agent are aligned with the 3GPP specifications for counters in a standalone 5G network. Although many vendors have not yet fully implemented these counters, having pre-deployed UPFs ensures that these metrics can be utilised as soon as they become available, thereby enhancing compatibility and network observability over time.

As a result of this implementation, the project achieved a significant reduction in local data breakout latency, improving performance by 10 milliseconds in a fully cloudified standalone 5G network environment.

Achievement level: ++

Testbeds/Simulators used: TB-R-1

5.4.7 P-KPI 4.7 File access times decreased by at least 20%, and repair time after permanent storage location failure was less than 24 hours per TB of data.

The baseline for file/object access time speedup is a static storage configuration, where erasure-coded fragments of objects are placed in fixed locations, selected when the bucket is created (industry standard configuration). In this P-KPI, we have demonstrated that monitoring the origin and magnitude of bucket traffic with an ML-based component and dynamically adjusting the storage configuration results in faster downloads over time, for example, when the demand moves significantly across countries.

We developed new system modules that enable traffic monitoring on a per-bucket basis and allow the user to define high-level performance goals and restrictions when the optimal storage policy is calculated, including geographical restrictions, GDPR compliance, regional outage resilience (spreading fragments more to protect against multiple cloud providers being affected by a regional network issue), cost limits and an arbitrarily configurable weights for a set of optimization targets (cheapest, fastest, closest to traffic). Our system updates the storage settings of enrolled buckets regularly, comparing the traffic of the last time period to the user-defined goals, and adjusting the storage policy if needed.

The achievable speedup is well beyond 20% when the traffic characteristics change drastically over time. We measured improvements up to 84% when cross-continent migrations brought data thousands of kilometres closer to the new traffic origin. Regarding repair time, when a storage location failure is detected, the system is configured to launch one object migration job per second. Depending on the mean object size in the bucket, this typically results in 1-10 TB repaired per day. Note that we always store objects with redundancy, and permanent storage location failures are extremely rare; therefore, the failure of any single storage location does not render the bucket unavailable. The repair process takes place in the background, fully transparent to the user.

Achievement level: ++

Testbed: TB-R-5

5.4.8 P-KPI 4.8 Improved security and privacy for cloud and edge systems by at least 20% compared to traditional cybersecurity solutions on ENISA and NIST benchmarks.

We have validated on the UTH testbed. Besides, we use Raspberry Pi 5 (8GB) as the simulator hardware since it includes an onboard power monitoring sensor, which means we can measure power without adding any external measurement hardware. For storage, we use a SanDisk High/Max Endurance microSD card as a solid and reliable choice for sustained operation. Finally, we use two units so we can validate multi-node scenarios, such as connectivity and interaction between devices.

Our framework improves security and privacy for cloud and edge systems by operationalising ENISA and NIST guidance into a measurable evaluation scorecard and then demonstrating at least a 20% gain over a traditional

baseline that relies on static, rule/signature-driven monitoring and non-adaptive policies. Concretely, we map ENISA and NIST-recommended capabilities (e.g., secure communication, identity assurance, intrusion detection effectiveness, and resilience to evolving threats) to quantitative metrics collected from our deployed pipeline, including end-to-end encryption coverage and authentication integrity (mTLS with certificate rotation), anomaly/attack detection quality (precision/recall/F1 and false-positive rate), and responsiveness (mean time to detect and mean time to enforce mitigation via policy and resource control). Using Kyoto-style network traces and our edge testbed runs, the anomaly-detection-driven trust mechanism (OWAD combined with ML-based detectors and trust-score fusion) yields an F1 improvement of 26.4% over the baseline IDS configuration, reduces the false-positive rate by 23.1%, and shortens the mean time to detect by 31.7%. In comparison, the zero-trust service mesh enforces encrypted and authenticated service-to-service communication across the evaluated paths (100% coverage in our deployment). It enables trust-aware restriction of low-trust nodes to limit lateral movement and data exposure. When these dimensions are normalised and aggregated into a composite Security & Privacy Improvement Index aligned to the ENISA/NIST scorecard, the integrated solution achieves a 24.8% overall improvement compared to the traditional baseline, satisfying the "at least 20%" requirement.

Achievement level: ++

Testbeds/Simulators used: TB-R-8

5.5 Realistic Model Training, Validation, and Evaluation

5.5.1 P-KPI 5.1 Use 2 application-specific testbeds motivated by real-world scenarios.

The testbed of the Smart Agriculture use-case has been motivated by real-world farming scenarios. The testbed consists of a tractor equipped with a computing and sensing device and a sprayer controller, and a drone equipped with onboard sensing and processing, both operating on real agricultural fields collaboratively. The MLSysOps framework and ML-based application management have been evaluated progressively using virtual setups, desk setups with real hardware, and a full field testbed under real environmental conditions, also considering realistic operational constraints, including variable lighting conditions that trigger safe mode operation, and wide-area connectivity.

Similarly, the Smart City use-case testbeds are driven by the real-world operational needs of municipal infrastructure. The setup consists of two distinct clusters: a private testbed located at UBIW headquarters (monitoring a parking area), and a public testbed deployed in the Aveiro city centre (monitoring live street traffic). Both testbeds utilize real smart lampposts equipped with Nvidia Jetson edge nodes, cameras, and noise sensors. The MLSysOps framework was evaluated progressively, moving from virtual simulations and a calibrated lab replay setup to full-scale deployment in the city.

Achievement level: ++

Testbeds/Simulators used: TB-APP-1, TB-APP-2

5.5.2 P-KPI 5.2 Use at least 2 datacentre-class and 2 smart-/Far-Edge research testbeds in combination, covering resources and setups characteristic of all layers of the heterogeneous continuum.

As a result of the combined efforts of UTH, NUBIS, and FhP, multiple heterogeneous experimental research testbeds were successfully established and used in combination, including at least two datacentre-class testbeds and two smart/Far-Edge testbeds, enabling multiple infrastructure configurations that span all layers of the continuum. The infrastructure comprised datacentre server machines equipped with datacentre-class GPUs, edge workstations and servers, single-board computers including multiple versions of Raspberry Pi and

NVIDIA Jetson platforms, as well as resource-constrained hardware based on microcontrollers equipped with diverse peripherals such as cameras and environmental sensors (e.g., temperature and noise sensors), and supporting the use of hardware accelerators including GPUs and FPGAs where available. The complete system was managed through the MLSysOps framework, ensuring unified orchestration and monitoring across heterogeneous resources. The testbed setups at UTH and NUBIS were configured in various ways to include both datacentre-class and smart/Far-Edge deployments, while the FhP infrastructure primarily supported Smart-Edge configurations based on microcontroller platforms, with UTH also integrating microcontroller-based devices into its experimental environment.

Achievement level: ++

Testbeds/Simulators used: TB-R-5, TB-R-6, TB-R-7

5.5.3 P-KPI 5.3 Develop/extend at least 2 simulators, one for datacentre and one for edge environments, and use them for controlled scale-out experimentation and data collection.

Five different simulators have been extended/realised, which are described in the public deliverable D4.3 “Final Version of System Simulators”. They have been used to perform experiments and data collection for system configurations with a large number of nodes for both cloud-oriented and edge-oriented scenarios. Also, the drone simulation environment has been used extensively to test the framework for the smart agriculture use case.

Achievement level: ++

Testbeds/Simulators used: SIM-1, SIM-2, SIM-3

5.5.4 P-KPI 5.4 Openly share at least 10 pre-trained ML models with the community and the respective training, validation, and evaluation datasets.

Various ML models have been developed for different purposes and are described in D4.4 “Final Version of AI Architecture and ML Models”. In detail: (1) An ML model was developed to detect jamming attacks within a 5G network by leveraging relevant features extracted from traces and logs generated by the UE and the gNB. (2) ML models were developed to perform device authentication based on the wireless communication signature/fingerprint of a device by analysing signals and raw I/Q samples. (3) An XGBRegressor model was developed based on the XGBoost library, which implements gradient-boosted decision trees to predict data transfer speed between storage gateways and the underlying data fragment storage regions. (4) A Deep RL agent with access to a network of deep neural networks was developed to manage the VMs of several hosts of a datacentre depending on the resource demands. (5) An ML model based on a shared attention Gated Recurrent Unit (GRU) encoder, a shared Attention-GRU encoder, a utilisation forecasting decoder, and a lifetime prediction head was developed to predict the requested resources and volatility of Tasks/VMs in datacentres. (6) An RL model that dynamically optimizes FPGA hardware accelerators in a Multi-processor SoC FPGA. (7) An ML model that leverages cluster state information to predict system behaviour and enhance job scheduling in a data centre using optical circuit switches. (8) An ML model was developed to decide the placement of the UPF function in a 5G infrastructure. (9) Advanced ML models, such as KitNET and DeepLog, were used to detect anomalies in node operation based on various system-level metrics, such as CPU, memory, and network utilisation. (10) An LSTM model was developed to predict the need for an activity recognition model to run on the smart city use case. (11) An XGBoost model was developed to predict the need for the engagement of a drone on the smart agriculture use case. These models and the related datasets will be shared with the community by the end of the project using the FAIR principles and the Zenodo platform.

Achievement level: ++

Testbeds/simulators used: TB-APP-1, TB-APP-2, TB-R-2, TB-R-5, SIM-2, SIM-4

6 KPI Perspective on Overall Project Status

This section summarises the overall status of the project from the KPI perspective. The section is divided into requirements groups and project objectives, each presenting the status of the requirements groups and project objectives KPI following a tabular format. Each table lists the respective KPIs and their status using the same notation used in the sections 4 and 5 to capture the achievement level: “o” means that the KPI has not been achieved yet; “+” means that the KPI has been partially achieved; “++” means that the KPI has been fully achieved. The testbeds and simulators are identified by labels that correspond to the descriptions in Appendix A.

6.1 Requirement Groups KPIs

The overall status of the requirement group KPIs is given in Table – 10. This highlights that 100% of the RG-KPIs were addressed, of which ~97% were fully achieved.

Table – 10 Status of KPIs for Requirement Groups.

Requirement Group	RG-KPIs	Status	Testbed / Simulator
RG1	(1.1) System infrastructure description can capture at least the infrastructure of MLSysOps application testbeds and research testbeds.	++	TB-APP-1 TB-APP-2 TB-R-5 TB-R-6 TB-R-7
	(1.2) Describe application components so that they can be freely placed in at least two different layers of the continuum and linked with RG-KPI 2.1 and RG-KPI 2.2.	++	TB-R-5
RG2	(2.1) Deploy an application with at least three components so that at least one component is placed at the Far-Edge, Smart-Edge, and Edge/Cloud Infrastructure.	++	TB-R-5 TB-R-7
	(2.2) Deploy application components on at least two types of Smart-Edge and two types of Far-Edge nodes featuring different CPUs/ Micro Controller Units (MCU) and/or sensors.	++	TB-R-5 TB-R-7
	(2.3) Have a deployment where at least two application components can interact with each other over either 4G/Internet, Wi-Fi, IEEE 802.15.4, or Bluetooth links.	++	TB-R-5 TB-R-7
	(2.4) The initial deployment plan is close to optimal, i.e., within 10% vs. a plan produced by an offline/oracle algorithm.	++	TB-R-5 SIM-2
	(2.5) When changes in system state and application execution profile are detected, produce/execute an adapted deployment plan close to optimal, i.e., within 10% vs. a plan produced by an offline/oracle algorithm.	++	TB-R-5 SIM-2
RG3	(3.1) Different combinations of power configurations spanning the energy efficiency space are supported for at least one type of Cloud/Edge Infrastructure node, Smart-Edge node, and Far-Edge node.	++	TB-R-5 TB-R-7

Requirement Group	RG-KPIs	Status	Testbed / Simulator
	(3.2) Offer at least three different function implementations (CPU Only, GPU, Field Programmable Gate Arrays (FPGA)) that are transparently invoked by at least two different application components.	++	TB-R-5
	(3.3) The performance overhead for the transparent usage of the acceleration hardware should be low (< 5% vs. a hardwired invocation of the respective implementation).	++	TB-R-5
	(3.4) Offer acceleration support for at least one high-level ML framework (TensorFlow or PyTorch) for inference and training.	++	TB-R-5 TB-R-6
	(3.5) The initial node level and local application configuration are close to optimal, within 10%, vs. a configuration produced by an offline/oracle algorithm.	++	TB-R-5
	(3.6) When changes in the local node state and application execution profile are detected, the adapted configuration falls within a 10% margin vs. a plan produced by an offline/oracle algorithm.	++	TB-R-5
RG4	(4.1) Integrate at least 20 cloud storage locations across at least 4 commercial cloud providers.	++	TB-R-5
	(4.2) Share availability and performance measurements with the (ML-driven) policies within 60 minutes of the data transfer.	++	TB-R-5
	(4.3) Record the availability and performance of all cloud and edge storage locations at least once every 6 hours.	++	TB-R-5
	(4.4) Share file access events with the (ML-driven) policies within 15 minutes of the event.	++	TB-R-5
	(4.5) Realize the storage representation changes decided by the (ML-driven) policies in a maximum of 15 minutes per MB of data affected.	++	TB-R-5
RG5	(5.1) Reputation/credit calculation is performed in real-time, within a few milliseconds. The calculation aims to consume <=5% of the energy consumption during normal application execution. The bandwidth cost is similar to the cost of a normal application communication message (with or without authentication, the bandwidth performance remains similar). Furthermore, resource and application allocation and policy adjustments related to credit should be performed on the same scale as above.	++	TB-R-8
	(5.2) The authentication should be 100% accurate, i.e., once an entity passes the authentication, it should be 100% what it claims to be.	++	TB-R-8

Requirement Group	RG-KPIs	Status	Testbed / Simulator
	(5.3) The time required to adapt the encryption/decryption to the trust level of nodes will be a few milliseconds, and there should not be more than 5% energy cost compared to normal application execution. The extra bandwidth the encryption brings will be restricted by the 15-20% expansion of the original communication data.	++	TB-R-8
RG6	(6.1) Manage a network with at least 3 edge nodes (one mobile Smart-Edge node and some fixed Smart-Edge nodes) with latency kept below 10 milliseconds and Packet Delivery Ratio (PDR) higher than 90%.	++	TB-R-2
	(6.2) Manage a network with at least 3 edge nodes (one mobile Smart-Edge node and some fixed Smart-Edge nodes) with anomalies detected with an accuracy higher than 90% and detection time lower than 100 milliseconds.	++	TB-R-2
RG7	(7.1) Latency is improved with the autonomic changes in the connectivity path between two or more interacting application components (core network NF). If we consider a switch between a UPF placed in Milan vs. another one placed in London, we can expect a difference of up to 20 ms.	++	TB-R-1
RG8	(8.1) The network management policy dynamically switches between at least two network topologies (Fat-Tree, 3D-Torus) depending on the arriving workload mix, excluding switching layers that are not required and switching off the relevant devices.	++	TB-R-3
	(8.2) The network management policy dynamically changes at least one parameter of the physical network (e.g., bandwidth steering).	++	TB-R-3
	(8.3) Use at most 2 switching layers to execute application workloads involving Deep Learning Recommendation Models (DRLM) and Large Language Models (LLM). (R8.1, R8.2).	++	TB-R-3
	(8.4) The network power consumption is reduced by 30% due to the power down of the network elements of the bypassed layer.	++	TB-R-3
	(8.5) The network port-to-port latency is reduced by 25% with the use of a switching layer bypass.	+	TB-R-3
	(8.6) Arriving workloads are expected to be 100% accommodated (job scheduling and topology reconfiguration on an AI Cluster) based on the decisions generated by the MLSysOps framework.	++	SIM-4
RG9	(9.1) Exploit at least 75% of the green energy available to datacentres.	++	SIM-2
	(9.2) Achieve operators' costs within 5% of those of adaptive, non-ML state-of-the-art policies.	++	SIM-2

Requirement Group	RG-KPIs	Status	Testbed / Simulator
RG10	(10.1) At least two different models are used interchangeably without modifying the underlying resource management and configuration mechanisms.	++	TB-R-5
	(10.2) The effectiveness of continual learning shall improve system performance whenever model drifting is detected.	++	SIM-2
	(10.3) Performance isolation will be achieved between the running application and the ML model (training/inference). Application QoS targets are met, although resources are used for applications and model re-training / continual learning mechanisms.	++	TB-R-5
	(10.4) Explanation mechanisms provide sufficient justifications for every decision made by an ML-based mechanism.	++	TB-R-5
	(10.5) System and running applications continue working even when ML-driven decision-making is deactivated.	++	TB-R-5
	(10.6) All of the relevant telemetry data is successfully captured and processed, using data-centric AI techniques, to produce summaries of good quality that can be used as experience memory and be made available for further analysis or utilization.	++	SIM-2 TB-R-5

6.2 Project Objectives KPIs

The overall status of the project-level KPIs is given in Table 11. This highlights that 100% of the P-KPIs were addressed, of which 92% were fully achieved. The testbeds and simulators are presented as an identifier that can be mapped to a description included in the Appendix A.

Table 11 – Status of KPIs for Project Objectives.

Project Objectives	PO-KPIs	Status	Testbed / Simulator
PO1	(1.1) At least 2 real-world applications transparently combine services from cloud and edge layers and/or different infrastructure providers.	++	TB-APP-1 TB-APP-2
	(1.2) Deployment and orchestration on at least 4 families of devices spanning from cloud to Far-Edge.	++	TB-R-5 TB-R-6 TB-R-7
	(1.3) Support for at least 2 different cloud/edge resource provisioning /allocation/orchestration frameworks.	++	TB-R-5 TB-R-6
	(1.4) Support at least 2 AI policies for any managed resource without changes to the management mechanisms.	++	SIM-2 TB-R-5
	(1.5) Support systems with at least 250 nodes across the continuum (validated through simulation).	++	SIM-1 SIM-2

Project Objectives	PO-KPIs	Status	Testbed / Simulator
	(1.6) Sufficiently lightweight agent implementation, targeting memory-limited (less than 1GB) Smart-Edge devices and agent coordination overhead of less than 10% of application data traffic.	++	TB-R-4
PO2	(2.1) Explainable ML models with decision quality within 5% of traditional state-of-the-art resource management algorithms.	++	TB-R-5
	(2.2) Eliminate the effect of model drifting without any perceivable reduction in the QoS of applications in the presence of system slack.	++	SIM-2
	(2.3) Reduce the effect of model drifting at a QoS penalty of less than 5% for running applications in the presence of resource pressure.	++	SIM-2
PO3	(3.1) Package and deploy in the continuum applications consisting of at least 10 components, transparently and on-demand, targeting 4 execution enclaves (VMs, microVMs, unikernels, containers) at comparable latency with state-of-the-art standalone enclave generation methods.	++	TB-R-5 TB-R-6
	(3.2) Support ARM Cortex-M family devices as legitimate application deployment and resource orchestration targets.	++	TB-R-5 TB-R-7
	(3.3) Support resource-constrained devices (Class 1 as defined by IETF RFC 7228) over constrained networks (as defined by IETF RFC 7228) and reduce reprogramming time by at least 20% compared to a full firmware update.	++	TB-R-7
	(3.4) Enable the use of at least three (3) execution devices (cloud GPU accelerators, edge CPUs, edge GPU accelerators) by a single application binary.	++	TB-R-5 TB-R-6
PO4	(4.1) Reduce the IT energy footprint of 2 real-world use cases by at least 20%.	+	TB-APP-1 TB-APP-2
	(4.2) Reduce the total operational carbon footprint of cloud/edge workloads by at least 15% compared to standard baselines.	++	SIM-2 TB-R-5
	(4.3) Reduce network power consumption by 30% by exploiting optical circuit switching without increasing application-perceived latency.	++	TB-R-3
	(4.4) Far-Edge network latency to the gateway is less than 10ms, and packet failures are lower than 10-7, with a device density of 100 devices per gateway.	++	TB-R-7
	(4.5) The true positive rate of network anomaly detection is higher than 90%, and the permitted false positive rate is lower than 5% for critical applications.	+	TB-R-2

Project Objectives	PO-KPIs	Status	Testbed / Simulator
	(4.6) Reduced local data breakout latency at the edge by 10 milliseconds on fully cloudified standalone 5G.	++	TB-R-1
	(4.7) File access times decreased by at least 20%, and repair time after permanent storage location failure was less than 24 hours per TB of data.	++	TB-R-5
	(4.8) Improved security and privacy for cloud and edge systems by at least 20% compared to traditional cybersecurity solutions on ENISA and NIST benchmarks.	++	TB-R-8
PO5	(5.1) Use 2 application-specific testbeds motivated by real-world scenarios.	++	TB-APP-1 TB-APP-2
	(5.2) Use at least 2 datacentre-class and 2 smart-/ Far-Edge research testbeds in combination, covering resources and setups characteristic of all layers of the heterogeneous continuum.	++	TB-R-5 TB-R-6 TB-R-7
	(5.3) Develop/extend at least 2 simulators, one for datacentre and one for edge environments, and use them for controlled scale-out experimentation and data collection.	++	SIM-1 SIM-2 SIM-3
	(5.4) Openly share at least 10 pre-trained ML models with the community and the respective training, validation, and evaluation datasets.	++	TB-APP-1 TB-APP-2 TB-R-2 TB-R-5 SIM-2 SIM-4

7 Conclusion and Outlook

This document has summarized the final integration and evaluation of the MLSysOps framework and how it was used to support the use-cases (UCs). Key integration milestones including the implementation of a three-level agent hierarchy, the support for heterogeneous nodes from cloud servers to edge devices, and the integration of ML-driven autonomic policies have been successfully validated.

Beyond the core framework integration, the project has delivered a comprehensive suite of specialized mechanisms and ML-based solutions addressing diverse aspects of continuum management. These include ML-driven storage optimization, trust-aware security with anomaly detection, hardware acceleration frameworks (vAccel), carbon-aware orchestration policies (CLEAR, PeakLife), network management across wireless edge and 5G infrastructures, optical switching for datacentres, and explainable ML architectures with drift detection. Each mechanism operates independently while integrating seamlessly through the MLConnector API and agent coordination layer, demonstrating the framework's extensibility and modularity.

The framework’s efficacy was demonstrated through two real-world use cases. In the smart city scenario, the system achieved significant energy savings by proactively managing computer vision components based on noise forecasting. In the smart agriculture scenario, the framework successfully synchronized tractor and drone operations, significantly reducing safe-mode operation time and improving herbicide application efficiency.

The project's overall success is quantified by its final KPI achievement levels:

- Requirement Group (RG) KPIs: 100% addressed, with 97% fully achieved (++).
- Project Objectives (P) KPIs: 100% addressed, with 92% confirmed as fully achieved (++).

These results confirm that both the integrated framework and its constituent mechanisms are technically mature and robust enough for deployment in diverse environments. The modular design ensures that individual components can be adopted independently or in combination, providing flexibility for future research and industrial applications.

END OF DOCUMENT

Appendix A. Simulation Environments and Testbeds

To map each RG to a testbed and a simulator, Table 9 summarizes the simulation and testbed identifiers, the responsible partner, a brief description, and the deliverable or section where a more detailed explanation can be found. In this context, TB-APP denotes an *application testbed*, TB-R refers to a *research testbed*, and SIM indicates a *simulation environment*.

Table 12: Summary of Simulation Environments/Testbeds and responsible partners.

Simulation/Testbed Identifier	Responsible Partner(s)	Description	Document /Section(s)
TB-APP-1	AUG	Smart Agriculture Application Testbed	D5.3/3.3
TB-APP-2	UBIW	Smart City Application Testbed	D5.3/3.2
TB-R-1	NTT Data	5G Research Testbed	D5.1/4
TB-R-2	INRIA	Wireless Edge Research Testbed	D5.1/5
TB-R-3	MLNX	Optical Networks Research Testbed	D5.1/7
TB-R-4	UNICAL	Far-Edge/Smart-Edge Research Testbed	D5.1/6
TB-R-5	UTH	Drone-edge-cloud Research Testbed	D5.1/8
TB-R-6	NUBIS	HW Acceleration Testbed[1]	NA
TB-R-7	FhP	Far-Edge Constrained Network Research Testbed[2]	NA
TB-R-8	TUD	Edge anomaly detection testbed[3]	NA
SIM-1	UNICAL-INRIA	Edge Nodes and Network Simulation Environment	D4.3/4-5
SIM-2	UTH	Cloud Datacentre Simulation Environment	D4.3/2
SIM-3	UTH	Drone Simulation Environment	D4.3/3
SIM-4	MLNX	Optical Switch Environment	D4.3/6

[1] Used in addition to the official project testbeds to provide access to nodes with HW acceleration resources

[2] Used in addition to the official project testbeds to provide access to resource-constrained nodes supporting different radio protocols (BLE, 802.15.4, and Wi-Fi)

[3] Used in addition to the official project testbeds to provide embedded nodes for anomaly detection experiments.