IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 31 July 2026, accepted 12 August 2026, date of publication 20 August 2026, date of current version 28 August 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3725528

RESEARCH ARTICLE

An Integrated Lifecycle-Management Pipeline for UAV-Augmented Smart Parking Surveillance Across the Cloud–Edge–End Continuum

ABDELHAK KADOUMA[1], ABDELGHANI MELIANI[2], QING LI[3], AMIT K. SHUKLA[1], AND MOHAMMED ELMUSRATI[1], (Senior Member, IEEE)

[1]University of Vaasa, 65200 Vaasa, Finland
[2]EURECOM, 06410 Biot, France
[3]Åbo Akademi University, 20500 Turku, Finland

Corresponding author: Abdelhak Kadouma (x3307846@student.uwasa.fi)

This work was supported in part by European Union's Horizon Europe Research and Innovation Program though the AC3 Project under Grant 101093129.

ABSTRACT The proliferation of unmanned aerial vehicles (UAVs) and Internet of Things (IoT) devices in urban monitoring creates a need for automated management of heterogeneous distributed computing environments. This paper presents the design, implementation, and experimental validation of a UAV-augmented smart parking surveillance system deployed across a cloud, edge, and end/far-edge computing continuum. The contribution is an integrated lifecycle-management pipeline—rather than a new standalone algorithm—in which an Ontology Semantic Reasoner (OSR) captures application intent and deployment constraints, a lifecycle manager (LiSO) deploys the microservices across cloud, edge, and end/far-edge clusters, a software-defined wide-area network (SD-WAN) controller manages runtime connectivity, and a forecasting-assisted policy maintains service continuity during UAV-triggered service hand-offs. The pipeline is evaluated on a real multi-site testbed spanning IONOS cloud in Germany and EURECOM edge/far-edge infrastructure in France. Deploying deep-learning inference at the far edge reduces total network traffic by at least 82% relative to cloud-centric processing. SD-WAN-based dynamic path selection increases video-analytics throughput from 53.7 to 114.9 frames per second (FPS), a gain of 61.2 FPS, relative to a single static path. A checkpointer operator integrated with LiSO characterizes stateful-migration saving and restoration overhead across state sizes of 10–500 MB and up to five concurrent instances, and the downtime and total migration time are reported across seven heterogeneous migration scenarios. To assess the energy effect of onboard inference, a Jetson-based power-consumption experiment was conducted using a 10-minute idle window and a 20-minute active DeepStream detection window. The active workload increased board input power from 5.307 W to 6.137 W on average, an increment of 0.830 W or 15.6%, indicating that migrating the workload away from the UAV-class node reduces compute-side energy demand. A fine-tuned time-series forecasting model with a tunable sensitivity parameter α drives proactive migration. The results provide experimental evidence for forecasting-assisted, descriptor-driven orchestration in multi-site IoT deployments.

INDEX TERMS Cloud–edge–end continuum, descriptor-driven orchestration, edge computing, forecasting-assisted migration, Internet of Things, Kubernetes, lifecycle management, SD-WAN, smart parking, stateful migration, unmanned aerial vehicle.

The associate editor coordinating the review of this manuscript and approving it for publication was Nurul I. Sarkar.

I. INTRODUCTION

Urban computing infrastructure is undergoing a fundamental transition. Video surveillance systems, once entirely

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

dependent on centralized cloud processing, are being redesigned to exploit a continuum of compute resources spanning public clouds, regional edge servers, and resource-constrained end devices such as Raspberry Pi units and NVIDIA Jetson boards mounted on unmanned aerial vehicles (UAVs). This continuum is commonly described as the cloud–edge–end continuum, in which cloud sites provide global coordination and elastic resources, edge sites host latency-sensitive processing close to data sources, and end devices perform sensing, actuation, and increasingly local inference. Recent work on cloud–edge–end collaboration has further emphasized this architecture as a foundation for distributed artificial intelligence and sixth-generation (6G) networked systems [1]. In this paper, the term *end tier* refers to the UAV-mounted Jetson and other far-edge devices in the UC2 deployment. We retain the term *far edge* when describing the physical deployment tier, while using *cloud–edge–end continuum* to align the architecture with the broader terminology adopted in recent literature.

This shift is driven by three constraints that cloud-centric architectures cannot satisfy simultaneously: *bandwidth*, *latency*, and *resilience*. A large multi-level parking facility equipped with multiple high-resolution cameras and environmental sensors can easily generate tens of megabits per second of video and telemetry data. Transmitting this entire payload to the cloud for analytics is expensive, saturates wireless backhauls, and introduces latency that is incompatible with real-time anomaly detection and traffic management. Edge and end/far-edge processing offer a natural remedy: by executing inference close to the data source, only enriched events and selected frames need to traverse the network, reducing bandwidth demand and improving responsiveness.

UAVs extend this paradigm by acting simultaneously as mobile camera platforms and dynamic far-edge compute nodes. Equipped with NVIDIA Jetson accelerators, UAVs can execute deep-learning models in flight, cover areas that fixed cameras cannot reach, and adapt their patrol patterns to operational conditions. However, integrating UAVs into a production cloud–edge–end continuum introduces system-level challenges that remain insufficiently addressed.

First, UAV-mounted Jetson devices, rack servers at edge sites, and virtual machines in the cloud differ by orders of magnitude in compute, memory, power budget, and connectivity; an orchestration system must abstract these differences while enforcing placement constraints that respect them. Second, UAVs move, change coverage zones, and eventually return to their docking stations, so services running on a UAV must migrate to an edge node without losing state when the UAV leaves its operational area or its battery falls below threshold. Third, far-edge/end nodes connected over fifth-generation (5G) cellular networks or Wi-Fi experience variable throughput and latency, and a multi-site application must continue to meet its service-level agreements (SLAs) even as link conditions fluctuate. Finally, deploying, monitoring, adapting, and migrating containerized microservices across heterogeneous sites requires either large DevOps teams or automated, intent-driven orchestration.

These challenges converge most sharply on the UAV itself. The parking facility under surveillance is geographically static, but the node that observes it is not: the UAV is a mobile, battery-limited far-edge platform whose connectivity varies with position and whose availability is bounded by flight time. On board, the Jetson runs a DeepStream pipeline that processes live camera streams to detect vehicles, persons, and objects, recognize activity, generate annotated frames, and emit event metadata and surveillance alerts. When the UAV approaches a battery threshold, changes coverage zone, returns to its docking station, or suffers link degradation, the inference service must move to another node without discarding the application state it has accumulated—loaded model artifacts, pipeline configuration, stream and tracking metadata, event queues, and cached frames. Preserving this state across relocation allows surveillance to continue with bounded interruption even though the lot under observation never moves.

This paper addresses these challenges through a concrete system: Use Case 2 (UC2) of the European AC3 (Agile and Cognitive Cloud-Edge Continuum) project, a UAV-augmented smart parking surveillance deployment managed by the AC3 Cloud Edge Computing Continuum Manager (CECCM). The CECCM provides an integrated lifecycle-management stack composed of a Graphical User Interface (GUI), an Ontology Semantic Reasoner (OSR), a lifecycle manager (LiSO), Local Management Systems (LMS) for Kubernetes and K3s, and an SD-WAN controller, following the AI-native CECCM architecture developed in the AC3 project [16].

Rather than treating these components as independent tools, this paper presents them as consecutive stages of a single lifecycle loop tailored to UAV-augmented parking surveillance. In the first stage, the OSR captures the surveillance application's intent and constraints and compiles them into an Application Descriptor (AppD). LiSO then consumes the descriptor to deploy the microservices across the cloud, edge, and end/far-edge tiers. For inter-site networking, the SD-WAN controller exposes a northbound API to upper-layer orchestration components, and LiSO invokes this API to request overlay configuration, path-selection policies, and DNAT rules for traffic between sites. At runtime, the monitoring pipeline continuously observes infrastructure and application-level metrics, which feed a forecasting model that anticipates resource saturation or operational risks specific to an airborne far-edge node—imminent battery depletion, a coverage-zone change, or degradation of the UAV's wireless link. When such a risk is predicted, LiSO and the checkpointer operator relocate the affected services, and LiSO's post-migration strategy redirects traffic only after the destination instance is ready. Deployment, connectivity, observation, prediction, migration, and redirection thus form

one continuous control cycle that preserves service continuity during UAV-triggered service hand-offs.

To avoid ambiguity, this paper uses *service hand-off* to denote application-level relocation of a microservice from one compute node to another, for example from a UAV-mounted Jetson to an edge node. This is distinct from radio-layer *handoff* within the same access technology and radio-layer *handover* across different access technologies, such as Wi-Fi and 5G. Architecturally, the CECCM mobility mechanism is designed to be independent of the underlying access technology: it observes mobility-related risks, including battery depletion, coverage-zone changes, and link degradation, and triggers service relocation when continued execution on the UAV becomes unsuitable. In the current implementation, however, the mechanism has been validated only in homogeneous network conditions using 5G connectivity. Therefore, this work demonstrates access-technology-independent service relocation at the orchestration level, but it does not experimentally evaluate heterogeneous radio-access handover.

The main contribution of this work is an integrated lifecycle-management pipeline for UAV-augmented smart parking surveillance. The pipeline connects five stages: intent capture, descriptor-driven deployment, runtime connectivity management, state-migration characterization, and forecasting-assisted service continuity. These stages are not five independent techniques but successive phases of a single workflow, each realized within the AC3 CECCM and validated experimentally on a geographically distributed testbed. Accordingly, the emphasis of this paper is on the system-level integration of these stages into one coherent lifecycle and on its end-to-end experimental validation on real multi-site infrastructure, rather than on a new standalone algorithm. The five stages, and the contribution associated with each, are as follows.

(C1) Intent capture and descriptor generation. At the entry of the pipeline, the OSR translates high-level UAV-surveillance requirements and deployment constraints—expressed by the developer through the GUI—into a machine-executable AppD. This stage establishes the placement, networking, and SLA contract that governs every subsequent stage. It is realized as a complete OSR $\rightarrow$ AppD $\rightarrow$ LiSO path that removes the need for hand-written Kubernetes manifests or manual SD-WAN configuration, enabling zero-touch onboarding across heterogeneous Kubernetes and K3s clusters.

(C2) Descriptor-driven multi-tier deployment. Consuming the descriptor, LiSO deploys the surveillance application across cloud, edge, and far-edge/end clusters spanning IONOS (Germany) and EURECOM (France), placing DeepStream inference on the UAV-mounted Jetson far edge as dictated by the descriptor rather than by manual tuning. This placement reduces total network traffic by at least 82% relative to cloud-centric processing, exceeding the traffic-reduction target defined for the deployment.

(C3) Runtime connectivity management. Once the application is running, LiSO invokes the northbound API exposed by the SD-WAN controller to request overlay configuration, path-selection policies, and DNAT rules. The SD-WAN controller then programs the distributed SD-WAN edge components that enforce inter-site connectivity and dynamic path selection. This runtime stage increases video-analytics throughput from 53.7 to 114.9 FPS, corresponding to a gain of 61.2 FPS relative to a single static path, demonstrating that the network is treated as an actively managed element of the pipeline rather than a fixed transport medium.

(C4) State-migration characterization. To prepare the pipeline for the relocation that UAV mobility makes unavoidable, a checkpointer operator integrated with LiSO is designed for heterogeneous K8s/K3s clusters and used to characterize the overhead of stateful migration—saving time and restoration time—across five state sizes (10–500 MB) and up to five concurrent instances. This stage isolates and quantifies the cost of state capture and restoration that any UAV-triggered service hand-off or resource-driven relocation must absorb.

(C5) Forecasting-assisted service continuity. At the final stage, a time-series forecasting model fine-tuned on real monitoring data, together with a tunable sensitivity parameter α, decides *when* to migrate, so that relocation is initiated ahead of resource saturation or a UAV-triggered service hand-off rather than after a failure. LiSO's post-migration strategy—bringing up the destination instance before the source is decommissioned—then governs the *extent* of service disturbance during the switch. The down-time and total migration time are reported across seven heterogeneous UC2 migration scenarios in Section VIII; the present campaign characterizes LiSO's post-migration behavior and does not claim a separate reactive Kubernetes baseline.

Taken together, these five stages form a single closed pipeline: developer intent is captured (C1), deployed across the continuum (C2), kept connected at runtime (C3), and kept continuous under UAV mobility through quantified state migration (C4) and forecasting-assisted, interruption-aware relocation (C5). The contribution of this paper is therefore the end-to-end integration and experimental validation of this pipeline on real, geographically distributed infrastructure, with a clear separation of concerns between intent capture (OSR/AppD), connectivity management (SD-WAN), migration-overhead characterization (checkpointer operator), migration timing (forecasting model), and interruption reduction (LiSO's post-migration strategy).

The remainder of this paper is organized as follows. Section II surveys related work. Section III defines the UC2 scenario, objectives, stakeholders, and key performance indicators (KPIs). Section IV describes the system architecture and its integration with the CECCM. Section V details the OSR-AppD-LiSO orchestration pipeline. Section VI

presents the stateful migration design and proactive trigger mechanism. Section VII describes the experimental testbed. Section VIII presents and analyzes the experimental results. Section IX discusses practical implications, scalability, and limitations. Section X distills lessons learned, and Section XI concludes the paper.

II. BACKGROUND AND RELATED WORK

The design of applications that span cloud, edge, and end/far-edge resources has attracted substantial research attention, yet the specific combination of UAV-based far-edge inference, automated lifecycle management, stateful migration, and programmable multi-site networking remains, to the best of our knowledge, unaddressed by existing systems. This section surveys the most relevant strands of prior work and positions the present contribution at their intersection.

Foundational work on hierarchical compute architectures established the conceptual basis for distributing workloads across cloud, fog, edge, and end-device tiers. The OpenFog Reference Architecture [2] defined an early reference model for fog computing, while recent work on cloud–edge–end collaboration positions end devices as active participants in distributed intelligence rather than as passive data sources. In particular, Chen et al. [1] discuss foundation-model-based cloud–edge–end collaboration as a framework for 6G native-AI networks, reinforcing the importance of coordinated compute placement across cloud, edge, and end tiers. The UC2 deployment follows the same architectural principle but instantiates the end tier as a UAV-mounted Jetson executing video inference close to the camera stream.

At the orchestration layer, KubeEdge[1] extends the Kubernetes control plane to manage lightweight edge nodes, and K3s[2] provides a resource-efficient Kubernetes distribution suitable for constrained hardware such as Raspberry Pi and Jetson devices. These frameworks are valuable execution environments, but they do not expose semantic intent capture, descriptor-driven placement, integrated SD-WAN control, or forecasting-assisted proactive migration as a single lifecycle-management pipeline. Prior work on unified application profiles for microservices in the cloud–edge continuum provides a complementary semantic modeling foundation for representing application components, consumers, data sources, and lifecycle constraints [4]. The AC3 CECCM builds on K8s and K3s as its underlying local management systems and augments them with an ontology-based application descriptor layer, descriptor-driven placement decisions, forecasting-assisted migration triggering, and integrated SD-WAN control. These capabilities are complementary to the underlying cluster-management foundations and can be combined with them.

The use of UAVs as aerial sensing platforms for surveillance and environmental monitoring has been surveyed comprehensively by Motlagh et al. [3], who identify the integration of UAVs with IoT infrastructure as a promising direction but focus primarily on data collection pipelines rather than on in-flight inference or automated service management. Recent work on joint trajectory and resource optimization for multi-UAV wireless networks [5] demonstrates the feasibility of combining mobility planning with communication co-design, yet assumes static processing endpoints and does not address containerized service orchestration or state-preserving migration. The present system closes these gaps by embedding UAV-mounted far-edge inference directly within an automated lifecycle-management framework that supports stateful migration and programmable multi-site networking.

Kubernetes-based orchestration for edge and fog environments has been studied from several complementary perspectives. Brogi et al. [6] model fog application placement as a constraint-satisfaction problem and show that declarative deployment descriptions can guide intelligent, policy-compliant placement decisions. Casalicchio and Perciballi [7] examine the influence of metric selection on container autoscaling, showing that scaling behavior is sensitive to the quality of the monitoring signal. More recent surveys systematize autoscaling techniques for container-based cloud, edge, and fog computing [8] and for cloud-native computing more broadly [9]. These studies classify reactive, proactive, and hybrid scaling mechanisms and highlight the importance of monitoring, prediction, and lifecycle-aware resource management. The present work differs by using forecasting not for replica-count scaling alone, but as a trigger for service relocation in a UAV-assisted, multi-site cloud–edge–end deployment.

Stateful container and service migration in edge environments is widely recognized as an open challenge. Puliafito et al. [10] identify state preservation as a critical problem in fog computing for IoT, noting that checkpoint overhead grows with in-memory state size in ways that are difficult to reconcile with strict latency budgets. Machen et al. [12] address live service migration in mobile edge clouds and demonstrate that minimizing application downtime requires decoupling the traffic-switch event from the underlying data transfer—a principle that directly informs LiSO's post-migration strategy. The enabling kernel mechanism is Checkpoint/Restore in Userspace (CRIU),[3] which provides process-level checkpoint/restore in userspace; its exposure through the Kubernetes forensic checkpointing application programming interface (API)[4] makes it accessible at the pod level within standard cluster management workflows. The checkpointer operator introduced in this

[1] KubeEdge Project, "KubeEdge: Kubernetes native edge computing framework," CNCF, 2024. Available: https://kubeedge.io/. Accessed: Jul. 28, 2026.

[2] Rancher Labs, "K3s: Lightweight Kubernetes," 2023. Available: https://k3s.io/. Accessed: Jul. 28, 2026.

[3] CRIU Project, "CRIU: Checkpoint/restore in userspace," 2023. Available: https://criu.org/. Accessed: Jul. 28, 2026.

[4] Kubernetes Blog, "Forensic container checkpointing (alpha)," Dec. 2022. Available: https://kubernetes.io/blog/2022/12/05/forensic-container-checkpointing-alpha/. Accessed: Jul. 28, 2026.

paper builds on these foundations and extends them to heterogeneous K8s/K3s environments with forecasting-driven, proactive migration triggering.

The application of machine learning to resource management in distributed computing has a well-established track record in both cloud and fog contexts. Gu et al. [13] demonstrate that proactive, prediction-driven resource allocation in fog environments can reduce SLA violations compared with purely reactive strategies, as anticipating workload demand allows the system to act before saturation rather than after. Tuli et al. [15] show that AI-based dynamic scheduling across stochastic edge-cloud environments can outperform heuristic baselines in latency and energy efficiency, although their evaluation is conducted entirely in simulation. The present work advances this line of research by deploying a fine-tuned time-series forecasting model on real monitoring telemetry to drive proactive service relocation in a live, geographically distributed cloud–edge–end testbed.

Table 1 summarizes the positioning of this work relative to closely related systems across five dimensions: UAV/end-tier execution, stateful migration, descriptor-driven orchestration with forecasting-assisted migration, SD-WAN integration, and multi-site experimental validation. To the best of our knowledge, no prior system addresses all five simultaneously, and this gap motivates the contributions of the present paper.

III. UC2: SCENARIO, OBJECTIVES, AND KPIs

A. SCENARIO DESCRIPTION

UC2 targets a large open or multi-level parking facility near a busy urban shopping mall, requiring continuous monitoring of vehicle congestion, security incidents, and environmental conditions. The deployment integrates:

- UAVs equipped with cameras and NVIDIA Jetson Orin (64 GB RAM, GPU) performing on-board video analytics during peak-hour patrol flights;
- fixed surveillance cameras at entry and exit points, connected to Raspberry Pi units for lightweight forwarding;
- environmental sensors on UAVs and fixed cameras measuring temperature, humidity, wind speed, and CO_2 levels;
- a centralized dashboard providing operators with a consolidated view of live streams, analytics results, environmental data, and alerts.

The system must scale to multiple cameras and drones while maintaining real-time responsiveness and service continuity even as UAVs move between coverage zones, return to docking stations, or encounter battery-triggered operational terminations.

B. STAKEHOLDERS

Surveillance operators monitor live streams and receive real-time alerts for unusual vehicle activity, congestion, or environmental anomalies. *System administrators* register devices, configure sensors, and maintain infrastructure health. *Application developers and DevOps teams* define microservices, deployment constraints, and SLAs via the CECCM GUI and OSR, without writing low-level Kubernetes manifests. *Infrastructure providers* operate the CECCM platform and manage its integration with K8s/K3s clusters and SD-WAN networking.

C. OBJECTIVES

The five experimental scenarios that structure the UC2 validation program address the following objectives:

- **O1. Zero-touch deployment:** Define, validate, and deploy the full UC2 application through the GUI, OSR, and LiSO without manual cluster configuration.
- **O2. Traffic reduction:** Quantify the bandwidth reduction achieved by far-edge inference relative to cloud-centric processing.
- **O3. Service continuity:** Validate proactive migration triggered by resource forecasting and quantify service interruption during migration.
- **O4. SD-WAN connectivity:** Demonstrate dynamic overlay path selection for multi-site connectivity and its impact on application throughput.
- **O5. Autonomous adaptation:** Close the monitoring-control loop for zero-touch operations using real-time telemetry.

D. KEY PERFORMANCE INDICATORS

Table 2 lists the KPIs extracted from the AC3 Description of Action (DoA) [17] and the corresponding experimental results achieved in this work. The traffic-reduction KPI is exceeded, while service continuity is validated through LiSO's post-migration behavior and the downtime and total migration-time trends reported in Fig. 9.

IV. SYSTEM ARCHITECTURE AND CECCM INTEGRATION

A. OVERVIEW

Fig. 1 illustrates the end-to-end UC2 architecture. Processing is distributed across three tiers: (i) the *end/far-edge tier*, comprising UAV-mounted Jetson devices performing real-time video inference and sensor acquisition; (ii) the *edge tier*, comprising regional servers at EURECOM hosting latency-sensitive backend logic and data management; and (iii) the *cloud tier*, comprising IONOS virtual machines hosting user-facing services and global orchestration. The CECCM binds these tiers into a unified management domain.

These tiers are operated by the CECCM as one lifecycle loop rather than as a collection of independent modules, and each component occupies a single, well-defined position in that loop. The GUI and OSR capture the developer's intent and generate the AppD; LiSO consumes the descriptor to place the UC2 microservices across the cloud, edge, and end/far-edge clusters; and the LMS and VIM components translate LiSO lifecycle actions into site-specific Kubernetes/K3s operations. For inter-site networking, the SD-WAN controller exposes a northbound API to the CECCM control plane. LiSO invokes this controller-facing API to request

TABLE 1. Comparison with related systems.

System	UAV/End Tier	Stateful Migration	Predictive Adaptation	SD-WAN	Multi-Site Validation
KubeEdge	✓	–	–	–	✓
Brogi *et al.* [6]	–	–	Partial	–	✓
Machen *et al.* [12]	–	✓	–	–	✓
Tuli *et al.* [15]	–	–	✓	–	–
AC3 CECCM (this work)	✓	✓	✓	✓	✓

overlay creation, path selection, and DNAT configuration, while the SD-WAN controller programs the distributed SD-WAN edge components that enforce those connectivity decisions. These overlays carry application traffic, enriched video events, telemetry, and management traffic between the geographically distributed sites.

During operation, the monitoring pipeline streams infrastructure and application metrics into the forecasting model, which predicts when a node will approach saturation or when an airborne end/far-edge node is about to leave service. That prediction drives the checkpointer operator and LiSO to migrate the affected microservices. LiSO's post-migration strategy redirects traffic only after the destination instance is ready. Deployment, connectivity, observation, prediction, migration, and redirection therefore form one continuous control cycle that preserves service continuity during UAV-triggered service hand-offs.

B. MICROSERVICE DECOMPOSITION

The UC2 application is decomposed into five containerized microservices:

TABLE 2. UC2 key performance indicators vs. Achieved results.

KPI	Target	Achieved
Traffic reduction via far-edge processing	$\geq 50\%$	$\geq 82\%$
Service continuity during migration	Preserve application availability	Source instance maintained until destination readiness; downtime and total migration time reported for all 7 scenarios (Fig. 9)

- **Nginx gateway:** Sole externally exposed endpoint; routes HTTPS traffic to backend microservices.
- **Frontend:** Web dashboard providing live video streams enriched with AI annotations, environmental time series, alert logs, and historical data access.
- **Backend:** Handles authentication, device registration, event aggregation, and API exposure to the frontend.
- **Database:** Relational store for user accounts, device metadata, historical sensor measurements, and alert logs.
- **DeepStream:** NVIDIA DeepStream[5] pipeline running on the Jetson GPU for real-time object detection, vehicle classification, and activity recognition.

Fig. 2 shows the logical microservice topology and placement of the UC2 application. The Nginx gateway is the only externally reachable entry point. It routes browser and dashboard requests to the frontend and backend services, while the DeepStream service executes on the UAV-mounted Jetson node and forwards enriched events, metadata, and selected annotated frames to the edge backend and database. This topology makes explicit the descriptor-driven placement policy: public-facing services are placed in the cloud, latency-sensitive backend and data services are placed at the edge, and GPU-accelerated video inference is placed close to the camera stream at the end/far edge.

C. CECCM COMPONENT STACK

The CECCM integrates six functional components to achieve automated lifecycle management:

1) **GUI (Application Gateway):** Browser-based interface through which developers define microservices, resource requirements, placement constraints, and SLAs without writing YAML manifests.
2) **OSR (Ontology Semantic Reasoner):** Validates semantic consistency of application definitions, enforces policy compliance, and generates the AC3 Application Descriptor (AppD)—the primary contract between the application and the CECCM.
3) **LiSO (Lifecycle Manager):** Lifecycle manager that consumes the AppD, translated into a LiSO Network Service Descriptor (NSD), and orchestrates onboarding, instantiation, scaling, migration, and termination across all sites.
4) **LMS (Local Management System):** Site-specific adaptation layer interfacing LiSO with the local container orchestrator: Kubernetes at the IONOS cloud and K3s at the EURECOM edge and end/far-edge sites. The LMS translates LiSO instructions into native Kubernetes/K3s API calls and reports status back.
5) **VIM (Virtual Infrastructure Manager):** Abstracts low-level resource provisioning and image registry operations at each site.
6) **SD-WAN Controller:** Manages overlay networks interconnecting clusters across sites and supports

[5]NVIDIA Corporation, "DeepStream SDK," 2023. Available: https://developer.nvidia.com/deepstream-sdk. Accessed: Jul. 28, 2026.

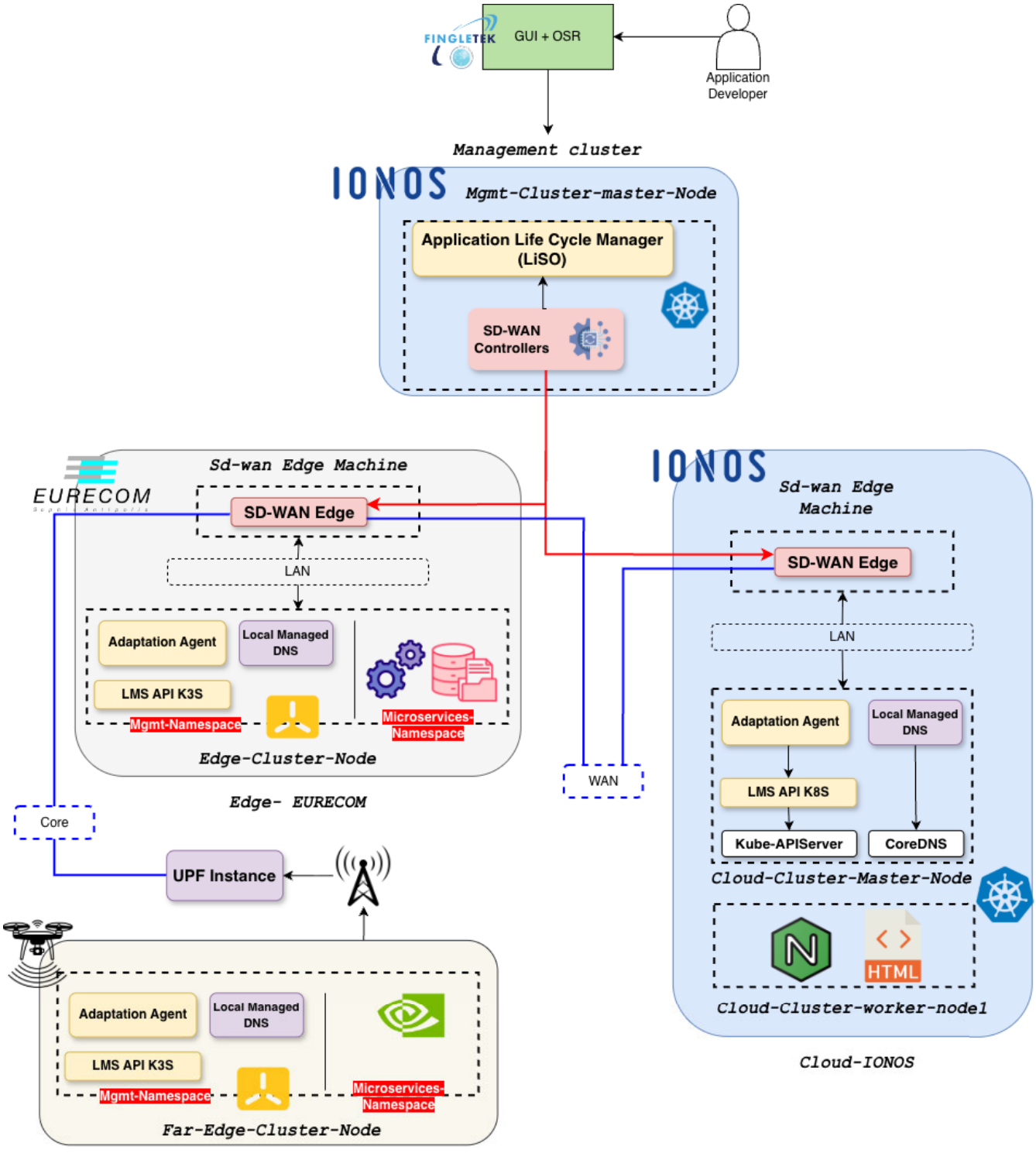


FIGURE 1. UC2 end-to-end architecture using the CECC manager framework. The management cluster hosts the GUI/OSR, LiSO lifecycle manager, and SD-WAN controller. LiSO deploys and manages the UC2 microservices across the IONOS cloud cluster, the EURECOM edge cluster, and the UAV-mounted end/far-edge node. For networking, the SD-WAN controller exposes a northbound API to the CECCM control plane; LiSO uses this API to request overlay, path-selection, and DNAT configuration, while the SD-WAN controller programs the distributed SD-WAN edge components. LMS/VIM and adaptation agents interface with the underlying Kubernetes/K3s clusters.

exposure of selected microservices to the internet through DNAT rules. Consistent with standard SDN/SD-WAN terminology, the controller exposes a northbound API to upper-layer orchestration components. LiSO acts as such an orchestration component: it invokes the SD-WAN controller's northbound API

to request overlay creation, path-selection policies, and DNAT configuration. The controller then translates these requests into configuration actions for the distributed SD-WAN edge components.

D. DEPLOYMENT MAPPING ACROSS THE CONTINUUM

The CECCM enforces the placement rationale encoded in the AppD. The Nginx gateway and frontend are placed in the IONOS cloud cluster because they require public internet reachability and have no hard latency dependency on the sensors. The backend and database are placed in the EURECOM edge cluster to reduce latency to the end/far-edge data sources while retaining datacenter-grade reliability. DeepStream is placed on the UAV-mounted Jetson end/far-edge cluster because it requires GPU access and colocation with the camera input, thereby avoiding the backhaul of raw high-definition video.

This deployment mapping is shown in Fig. 1 at the infrastructure level and in Fig. 2 at the microservice topology level.

E. MONITORING PIPELINE

A distributed metrics collection pipeline provides end-to-end observability. Local collectors at each site gather metrics from Prometheus, including CPU, GPU, memory, disk I/O, and network throughput, as well as application-level KPIs such as video FPS, end-to-end latency, frame encoding quality, and sensor readings. Collected metrics are forwarded via Apache Kafka to a central aggregator and persisted in InfluxDB for time-series analysis. Grafana dashboards provide real-time operator visibility. Critically, this monitoring data is the input to LiSO's forecasting-driven adaptation mechanisms, closing the observe-decide-act loop.

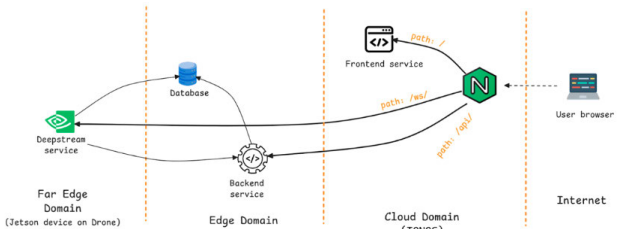


FIGURE 2. Microservice topology and placement of the UC2 application. Nginx acts as the externally reachable gateway, the frontend is placed in the cloud domain, backend and database services are placed in the edge domain, and DeepStream inference is executed on the UAV-mounted end/far-edge Jetson node.

V. DESCRIPTOR-DRIVEN ORCHESTRATION: OSR–AppD–LiSO PIPELINE

A. APPLICATION DESCRIPTOR SEMANTICS

The OSR is the principal interface through which developers describe UC2 to the CECCM. It combines an ontology-based knowledge representation with a semantic reasoner that validates application definitions against infrastructure policies and generates a structured AppD. This semantic layer is necessary because UC2 is not a homogeneous workload but a set of microservices with conflicting placement constraints that must be satisfied jointly. DeepStream requires a GPU and proximity to the cameras, fixing it to the UAV far edge; the backend and database require edge placement for low-latency access to far-edge data and durable persistence; the Nginx gateway and frontend require cloud placement with public internet reachability; and the SD-WAN overlays must be configured to interconnect this distributed service graph across sites. Expressing such interdependent constraints as hand-written Kubernetes manifests and SD-WAN rules is fragile and error-prone, since a single misplaced service or mis-specified route can violate a latency-critical path or expose an internal service. By encoding these constraints once in the AppD, the OSR allows the CECCM to enforce them automatically and consistently across the cloud, edge, and far-edge clusters. The AppD contains four semantically distinct sections:

1) **Global metadata:** Application name, version, owner, and lifecycle policies.
2) **Microservice configurations:** Per-service entries specifying container image, environment variables, resource requests and limits (CPU, memory, GPU), microservice-level SLAs, and eligible placement tiers (cloud/edge/far-edge).
3) **Networking graph:** Directed graph of logical connections between microservices, specifying protocol, port, traffic volume estimates, and per-link SLA attributes (latency, bandwidth, availability).
4) **Application-level SLAs:** End-to-end latency bounds, overall availability targets, and scaling policies.

Figs. 3–5 illustrate the three principal descriptor sections.

```
ApplicationName: "Surveillance System"
Version: "1.0.0"

Microservices_configuration:

Networking_graph:

Global_SLA:
  ServiceAvailability: "99.9%"
  MaxLatency: "500 ms"
  MaxResponseTime: "Low"
  DataThroughput: "High"
```

FIGURE 3. High-level structure of the AppD generated by the OSR, capturing microservice definitions, networking dependencies, and SLA requirements.

The semantic richness of the AppD is essential for descriptor-driven and policy-constrained placement: the networking graph exposes latency-critical paths (e.g., DeepStream → Backend) that LiSO must co-locate or

```
Microservices_configuration:
  - MicroserviceName: "frontend"
    Version: "1.0"
    Image: "capy8ra/ac3-uc2-frontend:latest"
    ID: "frontend"
    Dependencies:
      - "backend"
      - "deepstream"
    ResourceRequirements:
      Cpu: "1 vCPU"
      Memory: "2Gi"
    MicroservicesSLAs:
      ServiceAvailability: "99.9%"
      MaxResponseTime: "Low"
      DataThroughput: "High"
    ReplicaCount: "1"
    EnvironmentVariables:
      - Name: "VITE_USER_SERVICE_URL"
        Value: "http://172.21.16.156:30033/api/v1"
      - Name: "VITE_USER_SERVICE_BASE_URL"
        Value: "http://172.21.16.156:30033"
      - Name: "VITE_REACT_APP_SITE_KEY"
        Value: "XXX"
    apiEndpoint: "http://frontend.fingletek.com"
    apiPort: "5173"
    Protocol: "HTTP/REST"
    InternetAccess: "true"

    GeographicalArea:
      Region: "London-west"
      LocationType: "cloud"
```

FIGURE 4. Microservice configuration entry for the frontend service: image metadata, environment variables, resource constraints, placement eligibility (cloud-only), and per-service SLA.

minimize-latency route, while placement eligibility constraints prevent the CECCM from deploying the Nginx gateway on a resource-constrained Jetson device.

B. OSR-TO-LiSO TRANSLATION

The OSR exposes a REST API to LiSO. An internal translator at the OSR converts each AppD into a LiSO-native NSD that preserves microservice structure, networking topology, and SLAs but adapts to LiSO's internal representation. This translation is transparent to application developers and allows LiSO to remain decoupled from the semantic vocabulary of the GUI and OSR.

C. ONBOARDING AND INSTANTIATION WORKFLOW

Upon receiving the NSD, LiSO orchestrates a two-phase workflow.

Onboarding:

1) LiSO decomposes the NSD into per-microservice package descriptors.
2) For each microservice, LiSO requests the VIM Manager to onboard the service at the target site.
3) The VIM pushes the container image to the local registry and creates a namespace (K8s) or project (K3s) for the application.

```
Networking_graph:
  - Source: "backend"
    Destination: "db"
    Protocol: "TCP"
    Port: "5432"
    ConnectionSLAs:
      Latency: "Less than 500 ms"
      Availability: "99.9%"
      Bandwidth: "High"
      ErrorRate: "Less than 1%"
  - Source: "frontend"
    Destination: "deepstream"
    Protocol: "TCP"
    Port: "8585"
    ConnectionSLAs:
      Latency: "Less than 500 ms"
      Availability: "99.9%"
      Bandwidth: "High"
      ErrorRate: "Less than 1%"
  - Source: "frontend"
    Destination: "backend"
    Protocol: "TCP"
    Port: "8000"
    ConnectionSLAs:
      Latency: "Less than 500 ms"
      Availability: "99.9%"
      Bandwidth: "High"
      ErrorRate: "Less than 1%"
```

FIGURE 5. Networking graph excerpt specifying microservice interactions, protocols, ports, and per-link SLA attributes. This graph drives LiSO's placement and SD-WAN routing decisions.

Instantiation:

1) LiSO generates a unique instance ID, enabling multiple independent co-existing instances.
2) The VIM creates all required K8s/K3s objects: Deployments, Services, ConfigMaps, and PersistentVolumeClaims.
3) LiSO polls the LMS APIs until all pods reach `Running` state and records exposed endpoints (IP addresses, ports).
4) In parallel, LiSO invokes the northbound API exposed by the SD-WAN controller to request inter-cluster overlay creation and Destination Network Address Translation (DNAT) rules for internet-exposed services. The SD-WAN controller then applies the corresponding configuration to the SD-WAN edge components.

The entire workflow operates without manual intervention, fulfilling the zero-touch objective (O1).

VI. STATEFUL MIGRATION: DESIGN AND MECHANISM

A. ROLE IN UC2

In UC2, the parking facility under surveillance is stationary, but the far-edge node that observes it is not. The UAV is a mobile, battery-limited platform with a bounded flight window and position-dependent wireless connectivity, so the DeepStream inference service it hosts must periodically relocate to another far-edge or edge node. We refer to this operation as *service hand-off* or *service relocation*: an application-level transfer of execution from one compute node to another. This differs from radio-layer handoff within the same access technology and from radio-layer handover across access technologies, such as Wi-Fi-to-5G mobility. The CECCM is architecturally designed to remain independent of the underlying access technology and to react to observed connectivity, battery, coverage, or resource conditions rather than controlling radio mobility directly. In the present implementation, this mechanism has been tested only under homogeneous 5G connectivity; heterogeneous access handover is therefore outside the experimental scope of this paper. Relocation is driven by conditions intrinsic to airborne operation—reaching a battery threshold, changing coverage zone, executing a scheduled return to the docking station, suffering link degradation, or saturating the on-board compute—any of which can render the UAV unsuitable as a host even though the lot it monitors never moves.

The migration mechanism evaluated here preserves this CPU-side application and process state; full GPU execution-context migration of the active DeepStream inference engine is outside the scope of this work. To characterize the overhead of transferring such state without depending on a specific model build, the evaluation in Section VIII uses controlled in-memory state-size proxies spanning 10–500 MB. This range is a deliberate characterization and stress envelope rather than a claim about the live footprint of any single service: 50–200 MB is representative of lighter-to-moderate model, pipeline, and application artifacts, whereas 500 MB serves as an upper-bound stress case reflecting larger model artifacts, buffered metadata, cached frames, and multi-stream operation.

Within this setting, the three mechanisms of the migration stage play distinct and complementary roles. The forecasting model decides *when* to migrate, initiating relocation ahead of a predicted battery, connectivity, or saturation event rather than after the UAV has already left service. The checkpointer operator determines the *cost* of moving the service by capturing and restoring the accumulated state, thereby setting the state-transfer overhead. LiSO's post-migration strategy governs the *interruption* experienced during the switch, instantiating the destination instance and redirecting traffic only once it is ready. Acting together, these mechanisms ensure that operators continue receiving vehicle detections, annotated frames, and alerts even when the UAV leaves service. Accordingly, LiSO supports both stateless migration, used for the frontend and Nginx gateway, and stateful migration, used for services whose accumulated state must survive relocation; resource hotspots or load spikes at an edge server can likewise trigger relocation of backend services between clusters.

B. CHECKPOINTER OPERATOR DESIGN

A *checkpointer operator* is designed and implemented using the Kubernetes Operator pattern,[6] compatible with both full K8s and K3s. The operator automates a three-stage workflow:

1) **Checkpoint:** The operator issues a CRIU-based checkpoint request to all containers in the target pod, capturing memory pages, CPU register state, open file descriptors, and relevant filesystem data to a local directory.
2) **Image build:** The operator constructs a new Open Container Initiative (OCI)-compliant container image by layering the checkpoint archive atop the original image layers and compressing the result into registry-pushable format.
3) **Push and pull:** The operator pushes the checkpointed image to a container registry accessible by both source and destination clusters. LiSO then instructs the destination VIM to pull the image and start the container, which resumes execution from the saved state.

The total time across these three stages defines the *saving time*—the period during which the service is at risk of data loss. The *restoration time* measures the duration from image pull completion to full service resumption at the destination.

C. LiSO INTEGRATION AND POST-MIGRATION STRATEGY

LiSO integrates with the checkpointer operator via a REST API exposed at the management plane. Critically, LiSO implements a *post-migration strategy*: it first instantiates a new microservice instance at the destination, waits for it to become fully operational and integrated into the application workflow, and only then decommissions the source instance. This approach bounds downtime to the brief interval between the new instance becoming ready and traffic being redirected—regardless of state size or migration complexity.

D. PROACTIVE MIGRATION VIA TIME-SERIES FORECASTING

Migration decisions are driven by LiSO's forecasting-based decision engine, which uses a fine-tuned time-series forecasting model [19] to predict future resource consumption (CPU, GPU, memory) at each node. The model follows a two-phase training strategy:

[6] A. E. Meliani, "Migrator operator," GitHub repository, 2024. Available: https://github.com/abdelghanimeliani/migrator_operator. Accessed: Jul. 28, 2026.

1) **Pre-training:** An initial model is trained on the TimeTrack dataset [18]—a collection of high-granularity monitoring time series collected from an OpenAirInterface (OAI) CI/CD cluster—providing initial generalization for forecasting-based orchestration.
2) **Fine-tuning:** The model is continuously fine-tuned on real-time monitoring data from each target node, adapting to infrastructure-specific resource consumption patterns.

Fig. 6 illustrates this retraining architecture. A migration decision is triggered for affected applications on a node when the forecasted resource usage, adjusted by the sensitivity parameter α, exceeds a predefined threshold θ:

$$\hat{r}(t+\Delta) > \alpha\theta, \tag{1}$$

where $\hat{r}(t+\Delta)$ is the forecasted resource usage Δ steps ahead.

The parameter α offers a principled trade-off (Fig. 7):

- *Low α:* Lowers the effective threshold, resulting in more aggressive migration triggering—fewer missed migrations but more unnecessary migrations (increased overhead).
- *High α:* Raises the effective threshold, resulting in fewer unnecessary migrations but a higher risk of missed migrations (potential performance degradation).

This allows operators to calibrate the policy according to application priorities: responsiveness-critical deployments prefer low α; resource-constrained deployments prefer high α.

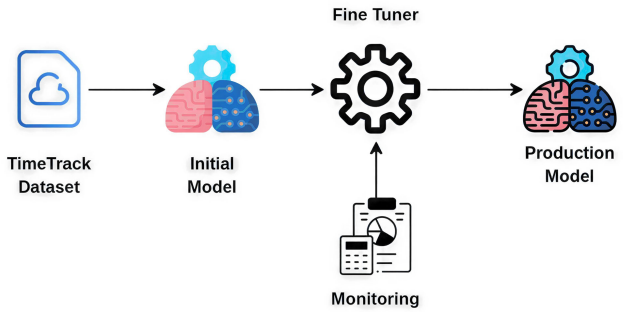


FIGURE 6. Fine-tuning pipeline for the proactive migration model: initial training on general Timetrack data, followed by fine-tuning on real-time monitoring telemetry for each infrastructure node.

VII. EXPERIMENTAL SETUP

A. UC2 PRODUCTION TESTBED

The UC2 application is deployed across two geographically separate regions.

IONOS cloud (Germany):

- *Management cluster:* Single-node K8s, 4 CPU, 8 GB RAM, 20 GB storage. Hosts LiSO and the SD-WAN controller.
- *SD-WAN edge cluster:* Single-node K8s, 4 CPU, 8 GB RAM, 20 GB storage.

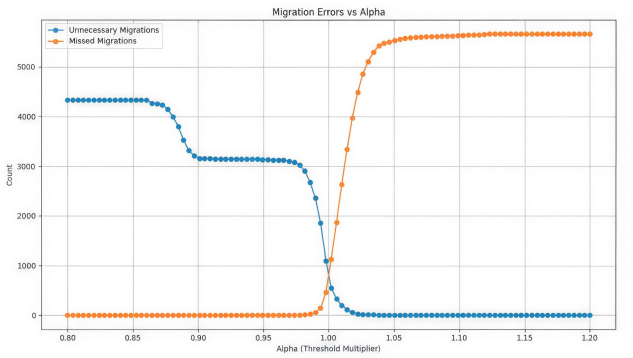


FIGURE 7. Migration error rates as a function of α. Lower α protects application performance (fewer missed migrations); higher α reduces migration overhead (fewer unnecessary migrations). Operators select α according to their SLA priorities.

- *Application cloud cluster:* Two-node K8s. Control plane: 4 CPU, 8 GB RAM, 64 GB storage; worker node hosts Nginx and the frontend.

EURECOM (France):

- *Edge cluster:* Single-node K3s, 20 CPU, 64 GB RAM, 480 GB storage. Hosts backend and database. SD-WAN edge virtual machine (VM): 2 CPU, 4 GB RAM.
- *Far-edge cluster:* Single-node K3s on NVIDIA Jetson Orin, 64 GB RAM, 64 GB storage, GPU. Hosts DeepStream. Connected to the rest of the infrastructure via a 5G base station deployed at EURECOM, representing UAV wireless connectivity.

B. JETSON POWER-CONSUMPTION EVALUATION

Direct UAV flight-autonomy measurements require access to the physical UAV platform, battery instrumentation, propulsion system, and coordinated flight testing with the external deployment partner. Since such access was not available during this revision, we evaluated the compute-side energy impact of onboard inference using an NVIDIA Jetson Orin Nano Developer Kit configured in 15 W `nvpmodel` mode. The Jetson platform emulates the UAV end/far-edge computing environment used to host the DeepStream workload.

The experiment compares two operating phases. The first phase represents the idle condition after stopping the application containers. The second phase represents the *pre-migration* condition, in which the DeepStream workload runs on the Jetson and performs onboard video analytics. Power was measured using the Jetson onboard INA3221 sensor through `tegrastats`, focusing on the `VDD_IN` rail. This rail reports DC input power at the development board and therefore captures compute-module power draw, not total UAV flight power or wall-outlet AC power.

Two long measurement windows were collected to improve reliability: a 10-minute idle window and a 20-minute active DeepStream detection window. The idle window produced 597 samples over approximately 600 s, and the

active window produced 1,193 samples over approximately 1,199 s, giving approximately 99.5% sample coverage. During the active window, the USB webcam provided 640 × 480 YUY2 video at 30 FPS, the pipeline converted the stream to NVMM/NV12, `nvstreammux` operated at 1280× 720 with batch size 1, and the DeepStream `nvinfer` primary detector ran with `interval=0`, meaning inference was performed on every frame. A `fakesink` was used to avoid display and on-screen-display overhead.

The original full DeepStream Compose service could not sustain the requested 20-minute run because it failed after approximately 2 min 8 s with a CUDA error in the OSD/tracker path. A reduced pipeline using a precompiled engine also failed after approximately 15 min 25 s because the engine plan had been built for a different device model. For the valid 20-minute measurement, the primary detector TensorRT engine was rebuilt locally on the Jetson from the supplied model and calibration data. The resulting stable workload is therefore a valid sustained USB-webcam DeepStream primary-detection workload, but it intentionally excludes the unstable tracker, secondary classifiers, Python frame-copying, WebSocket/database processing, and OSD. Consequently, the measurement is reported as compute-side inference power rather than full application power.

C. MIGRATION EVALUATION TESTBED

A dedicated dual-cluster virtual testbed is used for systematic migration evaluation:

- *Cluster A* (emulating an edge server): Vanilla Kubernetes 1.26.0 on Fedora 37 (kernel 6.0.7), 2 CPU cores, 4 GB RAM.
- *Cluster B* (emulating a far-edge node): K3s 1.28.8 on Fedora 37 (kernel 6.0.7), 2 CPU cores, 4 GB RAM.
- *Host machine:* Intel Core i7-1365U (13th generation), 32 GB Low Power Double Data Rate 5 (LPDDR5) memory at 6400 MHz.
- *Registry:* Local container registry shared by both clusters to eliminate external network variability.

Stateful services are modeled using a SQLite in-memory database container, with state sizes of 10, 50, 100, 200, and 500 MB. Experiments vary the number of concurrent container instances from 1 to 5. Each configuration is repeated 100 times. Median values are reported to reduce sensitivity to initialization artifacts and timing outliers. Where raw repeated measurements are available, variability is shown using interquartile ranges (IQRs), which are appropriate for timing data that may be skewed.

D. SD-WAN EVALUATION TESTBED

Two edge VMs (2 GB RAM, 1 CPU each) and one SD-WAN controller VM (4 GB RAM, 2 CPU) are deployed using the same host machine as the migration testbed. Wide-area network (WAN) characteristics are emulated using the `netem` traffic control tool.[7] Two distinct overlay paths (Overlay A and Overlay B) are configured with different latency and jitter profiles. Video-analytics FPS is measured over 120-second evaluation windows under each SD-WAN configuration.

VIII. RESULTS AND EVALUATION

This section presents and analyzes results from the five UC2 validation scenarios and the additional Jetson power-consumption experiment conducted to estimate the compute-side energy impact of onboard DeepStream inference.

A. UC2 SCENARIO 1: ZERO-TOUCH APPLICATION DEFINITION AND DEPLOYMENT

The complete UC2 application was defined through the CECCM GUI, validated and compiled into an AppD by the OSR, translated into an NSD, and deployed by LiSO across IONOS cloud and EURECOM edge/far-edge clusters—without any manual cluster configuration or direct Kubernetes API interaction by the developer. All five microservices reached `Running` state across their respective clusters. Automated SD-WAN configuration was performed through the controller's northbound API: LiSO requested inter-site overlay and DNAT configuration, and the SD-WAN controller applied the corresponding rules to the distributed SD-WAN edge components. This exposed the Nginx endpoint and dashboard to the internet. The deployment was further validated end-to-end by accessing the live surveillance dashboard from an external browser and confirming that video streaming, AI-annotated frames, and environmental sensor data were operational.

Significance: This result demonstrates that the OSR-AppD-LiSO pipeline successfully abstracts multi-cluster heterogeneity. Developers interact exclusively with high-level intent; all cluster-specific configuration is synthesized automatically, substantially reducing deployment effort and the potential for human error.

B. UC2 SCENARIO 2: TRAFFIC REDUCTION VIA END/FAR-EDGE INFERENCE

Two deployment configurations are compared over 5-minute measurement windows corresponding to the maximum flight time of the experimental UAV:

- **Case 01 (cloud-only):** DeepStream inference runs in the IONOS cloud; raw video frames are transmitted over the network from the far-edge camera to the cloud for processing.
- **Case 02 (end/far-edge inference):** DeepStream runs on the UAV-mounted Jetson; only enriched events and selected annotated frames are forwarded to the edge backend.

Fig. 8 presents received traffic (RX), transmitted traffic (TX), total traffic, and the resulting traffic gain for both cases.

[7]Linux traffic control documentation: https://lartc.org/. Accessed: Jul. 28, 2026.

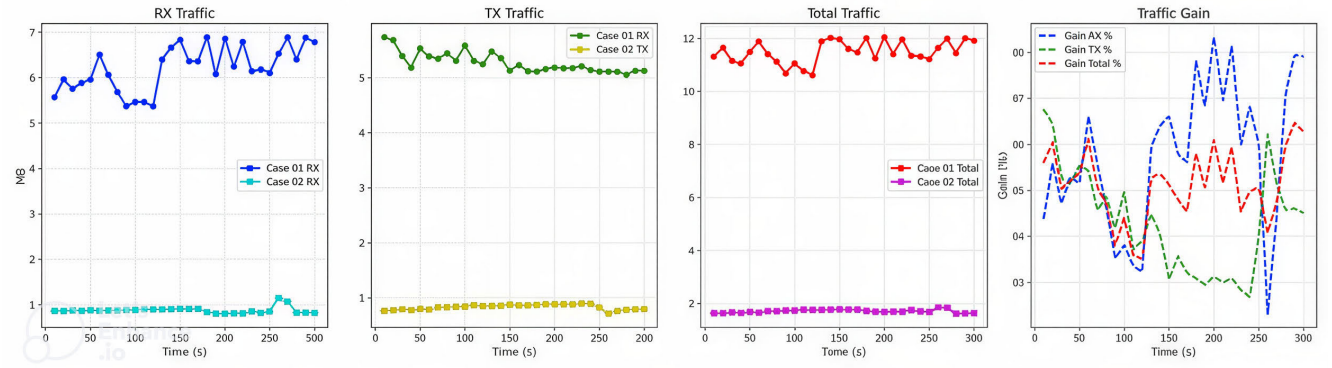


FIGURE 8. Comparison of network traffic over a 5-minute flight window for cloud-only (Case 01) versus far-edge inference (Case 02) deployments. From left to right: received traffic (RX) at the drone, transmitted traffic (TX) from the drone, total network traffic, and traffic gain. End/far-edge inference reduces total traffic by at least 82%.

1) RECEIVED TRAFFIC (RX)

Under Case 01, the drone receives over 5 MB even in best-case scenarios (required to download model updates, receive control commands, and stream configuration). Under Case 02, RX consistently remains below 1 MB, since the Jetson processes frames locally and requires only lightweight control and event sink communication.

2) TRANSMITTED TRAFFIC (TX)

The TX reduction is even more pronounced. Under Case 01, raw H.264/AVC-encoded video at the streaming resolution occupies the majority of TX; under Case 02, only annotated frames and event metadata are transmitted, reducing TX to a small fraction of the Case 01 baseline.

3) OVERALL TRAFFIC GAIN

The total traffic reduction achieved by end/far-edge inference in Case 02 relative to Case 01 is at least **82%**, substantially exceeding the AC3 DoA KPI of 50%.

4) ANALYSIS

This result confirms that collocating inference with the video source at the far edge is the dominant design decision for bandwidth management in UAV-based surveillance. In first-order network-budget terms, an 82% traffic reduction means that each UAV uses approximately 18% of the backhaul required by the cloud-centric baseline. Therefore, the same backhaul budget could support substantially more UAV video streams, subject to other constraints such as 5G radio capacity, scheduling overhead, edge compute availability, and control-plane scalability. Importantly, this result is obtained on Jetson Orin hardware over a 5G-connected far-edge setup, making the measurement more representative of practical UAV-based deployments than a simulation-only evaluation.

C. COMPUTE-SIDE ENERGY IMPACT OF ONBOARD DeepStream INFERENCE

To assess the practical energy impact of executing video analytics on the UAV-class end/far-edge node, we measured Jetson board input power during a 10-minute idle window and a 20-minute active DeepStream detection window. Table 3 summarizes the `VDD_IN` rail measurements.

TABLE 3. Jetson board input power during idle and DeepStream detection.

Condition	Samples	Mean (W)	Median (W)	Std. (W)	P5–P95 (W)
Idle, containers stopped	597	5.307	5.263	0.526	4.784–6.518
USB webcam + DeepStream detector	1193	6.137	6.050	0.539	5.612–7.380

The active DeepStream detection workload increased mean board input power from 5.307 W to 6.137 W, an increase of 0.830 W or 15.6%. The median power increased from 5.263 W to 6.050 W, an increase of 0.787 W or 15.0%. The measured energy over the requested windows was approximately 0.884 Wh for the 10-minute idle window and 2.046 Wh for the 20-minute active detection window.

These results confirm that executing DeepStream inference onboard the Jetson increases compute-side energy consumption, and that migrating the workload away from the UAV-class node reduces the board-level power demand. The effect should be interpreted as compute-module energy impact, not as a direct measurement of full UAV battery autonomy. Total flight autonomy also depends on propulsion, avionics, payload weight, camera power, wireless transmission, battery capacity, and flight dynamics. Nevertheless, reducing the onboard compute workload is beneficial when the UAV approaches a battery threshold, because it reduces one controllable component of the total energy budget.

During the valid 20-minute active window, the DeepStream primary detector remained stable with restart count 0. GPU activity was observed through `GR3D_FREQ`, with mean utilization of 21.3%, median utilization of 24%, and a maximum of 40%. This confirms sustained inference activity during the power-measurement window.

D. UC2 SCENARIO 3: SERVICE CONTINUITY THROUGH PROACTIVE MIGRATION

1) MIGRATION CORRECTNESS AND SERVICE INTERRUPTION

We define *service interruption* (downtime) as the interval during which the application cannot serve requests while traffic is switched from the source instance to the destination instance. We distinguish it from *total migration time*, which spans the complete relocation procedure, from checkpoint initiation through destination readiness to source decommissioning. The two quantities are produced by different mechanisms: the forecasting model decides *when* migration is initiated, the checkpointer operator determines the state-transfer overhead that contributes to total migration time, and LiSO's post-migration strategy determines the downtime experienced during the traffic switch. Rather than characterizing this downtime with a subjective term such as "minimal," we define it explicitly and report its observed per-scenario behavior from Fig. 9.

The seven UC2 migration scenarios are integrated end-to-end validation cases rather than repeated microbenchmarks. Fig. 9 therefore reports the observed downtime and total migration time for each scenario. Variability is reported for the repeated checkpointing experiments in Figs. 10–12, where each configuration is repeated 100 times.

Fig. 9 reports both the downtime and the total migration time for seven heterogeneous migration scenarios, each relocating a different subset of UC2 microservices from the edge to the cloud: M1 (Frontend only), M2 (Backend only), M3 (Frontend + Backend), M4 (DeepStream only), M5 (Backend + DeepStream), M6 (Frontend + DeepStream), and M7 (all three).

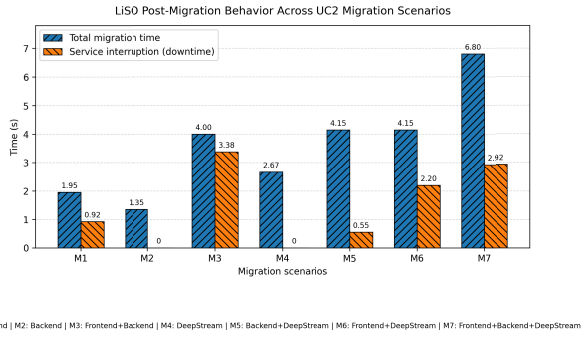


FIGURE 9. Service interruption (downtime) and total migration time across seven UC2 migration scenarios. The x-axis identifies the migration scenarios M1–M7, and the y-axis reports time in seconds. M1 migrates the frontend, M2 the backend, M3 the frontend and backend, M4 DeepStream, M5 the backend and DeepStream, M6 the frontend and DeepStream, and M7 all three services. Downtime is governed by LiSO's post-migration strategy and remains limited across all scenarios, while total migration time increases with migration scope and DeepStream involvement.

a: DOWNTIME BEHAVIOR

Across the seven scenarios, the downtime observed in Fig. 9 remains limited and does not grow in proportion to the migration scope. This follows directly from LiSO's post-migration strategy: the source instance continues serving traffic until the destination instance is operational and responsive, so the downtime is bounded by the traffic-switchover interval rather than by the cost of capturing and transferring state.

b: TOTAL MIGRATION TIME BEHAVIOR

In contrast, the total migration time in Fig. 9 increases with the number of migrated microservices and is highest when DeepStream is involved (scenarios M4–M7), because DeepStream is the heaviest migrated component and is associated with model artifacts, pipeline configuration, and buffered metadata, unlike the lighter frontend and backend services. Since the source and destination instances run concurrently during this period (dual-instance mode), the longer total migration time is absorbed while the application continues to be served and therefore does not translate into longer downtime.

c: SCOPE OF THIS VALIDATION

The present UC2 campaign characterizes the behavior of LiSO's post-migration strategy and the relationship between downtime and total migration time across the seven scenarios. It does not include a separate reactive Kubernetes baseline (such as delete/recreate or eviction/restart). The results are therefore reported as a characterization of LiSO post-migration behavior, without claiming a controlled comparison against Kubernetes-native recovery; this scope constraint is stated explicitly in Section IX.

2) STATEFUL MIGRATION CHARACTERIZATION

To characterize the state-transfer overhead that any such relocation must absorb, the checkpointer operator is evaluated on a controlled in-memory state-size proxy rather than on the GPU execution context of the DeepStream engine, whose migration is outside the scope of this work. Figs. 10–12 report the resulting trends rather than isolated values.

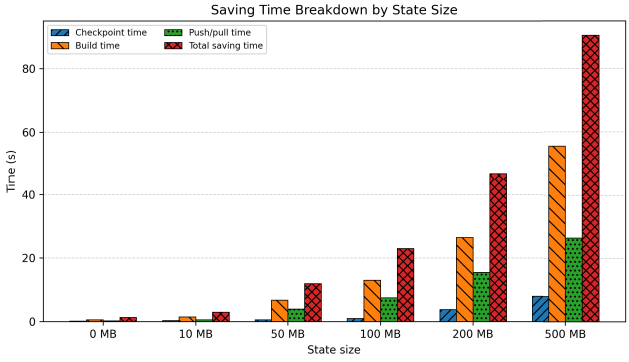


FIGURE 10. Saving time breakdown for a single container as a function of state size. Bars report the measured checkpoint creation, image build, push/pull, and total saving time components in seconds. Where repeated-run variability is available, error bars report the interquartile range (IQR). Image building dominates at large state sizes.

a: SAVING TIME

Fig. 10 decomposes saving time into checkpoint creation, image building, and push/pull. Checkpoint creation grows approximately linearly with state size, image building becomes the dominant component as state size increases, and push/pull to the local registry is secondary but non-negligible at the largest sizes. These trends, rather than any single value, indicate where the overhead accumulates.

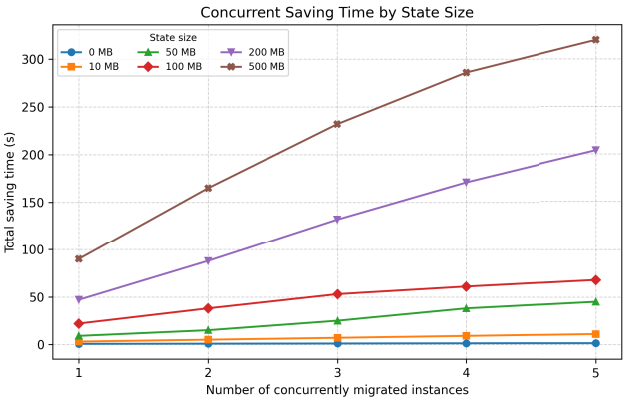


FIGURE 11. Total saving time for concurrent migration of 1–5 container instances across different state sizes. The x-axis reports the number of concurrently migrated instances, and the y-axis reports total saving time in seconds. Where repeated-run variability is available, error bars report the interquartile range (IQR). Sub-linear scaling at small state sizes reflects parallelism benefits; super-linear growth at large state sizes reflects CPU, memory-bandwidth, and I/O contention.

b: CONCURRENT MIGRATION

Fig. 11 shows total saving time when migrating multiple instances concurrently. At small state sizes the scaling is sub-linear, reflecting overlap of checkpoint operations across available CPU parallelism; at large state sizes it becomes super-linear as CPU, memory bandwidth, and storage I/O saturate. The orchestration implication for UC2 is that LiSO should stagger migrations in order of ascending state size when DeepStream is involved, rather than migrating all services on a node simultaneously.

c: RESTORATION TIME

Fig. 12 shows that restoration time scales more gently than saving time and is dominated by image transfer; once the image is local on the destination node, resuming the checkpointed state is rapid. For UC2 this suggests that pre-positioning checkpointed images at edge registry mirrors would reduce restoration time for UAV-to-edge migrations.

3) PROACTIVE TRIGGER VALIDATION

The proactive trigger is produced by the forecasting model of Section VI. Fig. 6 shows the two-phase training and fine-tuning architecture: the model is first pre-trained on the Timetrack memory-usage time series, sampled at 45-second intervals, and is then fine-tuned using real-time monitoring

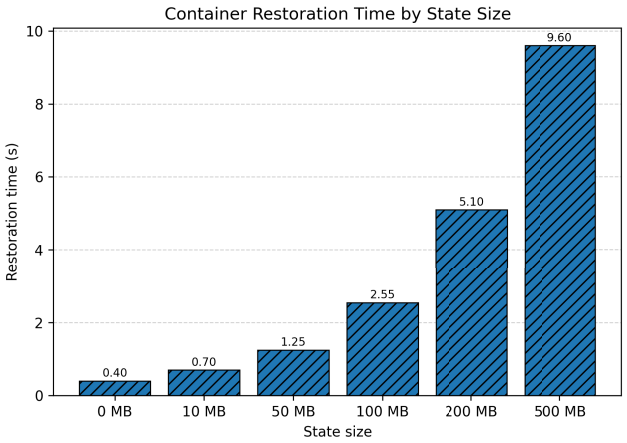


FIGURE 12. Container restoration time versus state size. The x-axis reports state size in MB, and the y-axis reports restoration time in seconds. Where repeated-run variability is available, error bars report the interquartile range (IQR). Image pull from the local registry dominates; actual state deserialization is fast once the image is local.

telemetry collected from each target node so that it adapts to infrastructure-specific resource consumption patterns.

Fig. 13 compares the fine-tuned model's predicted resource usage against the actual measured usage. The model follows the global resource-usage trajectory but underestimates sharp peak values, a known characteristic of time-series models optimized for average-pattern accuracy. We report this behavior as observed in Fig. 13 and do not claim any specific forecasting accuracy metric.

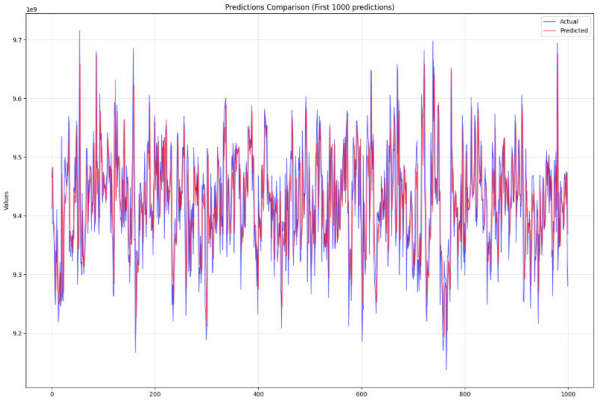


FIGURE 13. Predicted versus actual resource usage for the fine-tuned forecasting model. The figure compares the first 1000 prediction samples against the corresponding measured values. The model follows the overall resource-usage trajectory but underestimates some sharp peaks, motivating the use of the sensitivity parameter α in the migration-triggering policy.

Because the model struggles with peak values, the sensitivity parameter α is used to tune migration sensitivity by adjusting the effective migration threshold in (1). Fig. 7 shows how the rates of missed and unnecessary migrations vary with α: a lower α lowers the effective threshold and makes migration more aggressive, reducing missed

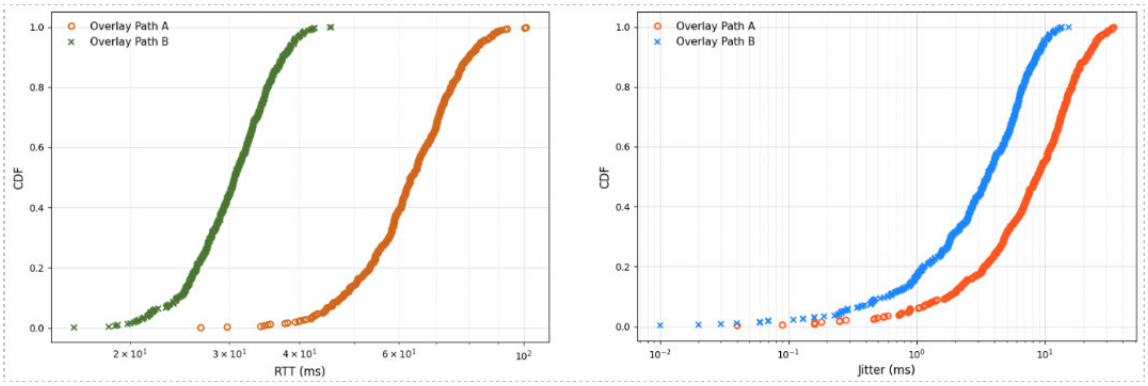


FIGURE 14. Cumulative distribution functions of RTT (left) and jitter (right) for Overlay Path A (higher latency, orange) versus Path B (lower latency, blue). Path B RTT spans 18–42 ms (median 30 ms) versus Path A's 28–100 ms (median 65 ms). Jitter: Path B median ≈5 ms; Path A median ≈8 ms.

migrations at the cost of more unnecessary ones, whereas a higher α reduces unnecessary migrations at the cost of more missed migrations. The curve therefore supports operator calibration of α according to deployment priorities rather than prescribing a single universal default value.

E. UC2 SCENARIO 4: SD-WAN MULTI-SITE CONNECTIVITY

1) PATH CHARACTERIZATION

Path B exhibits round-trip time (RTT) of 18–42 ms (median 30 ms) with jitter median of ≈5 ms. Path A exhibits RTT of 28–100 ms (median 65 ms) with jitter median of ≈8 ms. The RTT difference at the median is ∼35 ms—a factor of 2.2×—which, when combined with jitter, has a disproportionate impact on time-sensitive video streaming and real-time analytics.

Fig. 15 shows the impact of SD-WAN traffic redirection on application-level video-analytics FPS over a 120-second window.

2) ANALYSIS

Without SD-WAN-enabled path selection, the application is pinned to the static path, which in the worst case includes Path A characteristics—high RTT and high jitter causing video frame buffering, pipeline stalls, and reduced FPS. Mean FPS without redirect: **53.7** (range 46–60 FPS). Mean FPS with SD-WAN redirect: **114.9**—a gain of **61.2 FPS**, corresponding to a 2.14× improvement. In the implemented architecture, path changes are requested by LiSO through the northbound API exposed by the SD-WAN controller; the controller then updates the SD-WAN edge components that enforce the selected overlay path. This result has three important implications:

1) The network path is as important as compute resources for real-time video analytics; ignoring it leads to severe underutilization of GPU capacity.
2) The CECCM's SD-WAN integration elevates the network from a passive transport medium to an actively managed resource, co-optimized with compute placement.
3) In real WAN environments, where path quality may vary more dynamically than in the emulated laboratory setup, the benefit of SD-WAN control will depend on the controller's ability to detect degradation and redirect traffic quickly.

F. UC2 SCENARIO 5: MONITORING-DRIVEN AUTONOMOUS ADAPTATION

The monitoring pipeline was validated end-to-end: CPU, GPU, memory, FPS, and service health metrics were collected at each site, forwarded to the central Kafka/InfluxDB stack, and visualized in Grafana dashboards accessible to operators. Alert rules were configured for FPS degradation and sustained CPU overload, and were observed to trigger correctly during induced load events. LiSO's autoscaling mechanism successfully adjusted resource allocations in response to monitoring signals, demonstrating the closed observe-decide-act loop required for zero-touch operations.

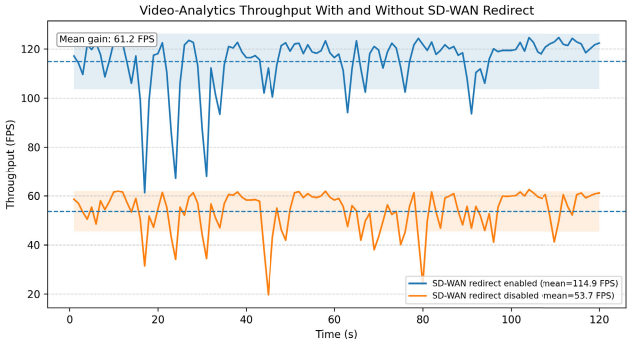


FIGURE 15. Video-analytics FPS over 120 s with and without SD-WAN traffic redirection. The curves show per-second FPS samples, while the horizontal markers report mean FPS for each configuration. SD-WAN-enabled dynamic path selection improves mean throughput from 53.7 to 114.9 FPS, corresponding to a 61.2 FPS gain and a 2.14× improvement.

G. SUMMARY OF EXPERIMENTAL FINDINGS

Table 4 consolidates the key quantitative results.

TABLE 4. Summary of key quantitative results.

Metric	Baseline	With CECCM
Network traffic reduction (end/far-edge inference)	0%	$\geq 82\%$
Video FPS (single static path)	53.7	—
Video FPS (SD-WAN redirect)	—	114.9 (+61.2)
SD-WAN RTT (median, Path A→B)	65 ms	30 ms (−54%)
Jetson board input power	5.307 W idle	6.137 W active detection (+15.6%)
Migration downtime (M1–M7)	No reactive baseline evaluated	Limited downtime across all seven scenarios (Fig. 9); total migration time grows with scope and DeepStream involvement

IX. DISCUSSION

A. PRACTICAL IMPLICATIONS

The 82% traffic reduction achieved by end/far-edge inference has immediate practical impact for deployment economics. For a multi-UAV deployment with 10 drones each streaming ∼10 Mbps raw video, moving inference to the far edge reduces backhaul requirements from ∼100 Mbps to ∼18 Mbps—a difference that can determine whether a 5G network slice is cost-effective and whether real-time alerting latency is acceptable.

The Jetson power measurements complement the network-traffic results by quantifying the compute-side energy cost of onboard inference. Over a 10-minute idle window and a 20-minute active detection window, DeepStream primary detection increased Jetson board input power from 5.307 W to 6.137 W on average, corresponding to a 0.830 W or 15.6% increase. This does not directly translate into full UAV flight-time extension, because propulsion typically dominates the total energy budget. However, it confirms that service relocation reduces the compute-side power draw on the UAV-class node, improving the energy margin when the UAV approaches a battery threshold.

The 114.9 FPS achieved with SD-WAN redirect should be read as the measured video-analytics processing throughput sustained over the 120-second evaluation window, rather than as the physical camera input frame rate. This throughput headroom indicates that the system is not GPU-limited under the tested conditions and can absorb additional camera streams, positioning the architecture as scalable beyond the single-UAV UC2 scenario to multi-UAV deployments sharing the same edge infrastructure.

The limited downtime observed across the seven migration scenarios in Fig. 9 supports the feasibility of maintaining service continuity during migration, although long-term availability must still be evaluated over extended operational periods. No separate reactive Kubernetes baseline was evaluated, so these results characterize LiSO's post-migration behavior rather than a controlled comparison against Kubernetes-native recovery. The post-migration strategy is the key mechanism; its behavior is validated across multiple UC2 microservice migration scenarios, while the state-size experiments separately characterize the overhead of the checkpointing mechanism.

B. SCALABILITY CONSIDERATIONS

The current testbed validates single-UAV operation. Scaling to N concurrent UAVs introduces several challenges.

1) CONCURRENT MIGRATIONS

The parallel migration results (Fig. 11) show super-linear time growth at large state sizes. With multiple UAVs simultaneously reaching operational boundaries, LiSO must implement priority-aware, staggered migration scheduling. A migration queue ordered by urgency (battery level, SLA violation risk) and service state size would reduce contention.

2) REGISTRY BANDWIDTH

In multi-UAV scenarios, pushing and pulling multiple large checkpointed images simultaneously will stress the registry network interface. Distributed registries co-located at edge sites—rather than a single centralized registry—would mitigate this bottleneck. Recent work on layer-reuse-aware microservice migration in UAV edge systems further shows that container-layer reuse and registry selection are important optimization dimensions for reducing provisioning overhead [14].

3) FORECASTING MODEL DIVERSITY

The proactive migration model is fine-tuned per infrastructure node. For a fleet of 10–100 UAVs, maintaining per-node models requires a federated learning or model distillation approach to avoid centralized training bottlenecks.

C. LIMITATIONS

The following limitations are acknowledged:

1) **Single-UAV evaluation.** All far-edge experiments involve one UAV. Multi-UAV coordination, potential 5G channel contention, and interference effects are not evaluated.
2) **Battery-autonomy measurement.** The energy evaluation measures Jetson board-level DC input power during idle and sustained DeepStream primary detection, not full UAV flight autonomy. A complete autonomy study would require instrumenting the physical UAV battery and measuring propulsion, avionics, camera, wireless transmission, and Jetson compute power during controlled flight. In addition, the stable 20-minute workload uses a primary-detection

DeepStream pipeline with `fakesink`; it does not include the unstable tracker, secondary classifiers, Python frame-copying, WebSocket/database processing, or on-screen display. Therefore, the experiment estimates the compute-side energy benefit of migrating inference away from the UAV-class node, but does not claim a measured end-to-end flight-time extension.

3) **Emulated WAN characteristics.** SD-WAN experiments use `netem` to emulate WAN conditions. Real-world WAN paths may exhibit different and more dynamic characteristics, including correlated burst losses that `netem` does not fully model.
4) **State size proxy.** In-memory SQLite databases are used as proxies for real microservice state. The actual checkpoint size of a running DeepStream pipeline depends on model architecture, buffer configuration, and runtime state, and may differ from the SQLite model.
5) **Forecasting model accuracy.** The fine-tuned forecasting model struggles with peak values, as illustrated in Fig. 13. Future work should explore ensemble methods or anomaly-detection-based triggers to complement the forecasting model for peak events.
6) **Security evaluation.** While SD-WAN overlays provide encrypted inter-site tunnels, end-to-end security—including mutual Transport Layer Security (mTLS) between microservices, zero-trust access control, and hardware-based attestation for edge nodes—remains outside the scope of this paper. Security and trust management for the broader AC3 cloud–edge continuum architecture is addressed separately in prior AC3 work [11].
7) **No reactive baseline.** The UC2 validation did not include a separate reactive Kubernetes eviction/delete–recreate baseline; therefore, the migration results characterize LiSO's post-migration behavior and the relationship between downtime and total migration time, rather than a controlled comparison against Kubernetes-native recovery.
8) **Homogeneous access-network validation.** The mobility mechanism is designed at the orchestration layer and is intended to remain independent of the underlying access technology. However, the current implementation has been validated only under homogeneous 5G connectivity. The paper therefore evaluates service relocation across compute nodes, not radio-layer handoff or heterogeneous access handover such as Wi-Fi-to-5G mobility.

D. COMPARISON WITH THE STATE OF THE ART

Relative to related work (Table 1), the AC3 CECCM combines UAV end/far-edge execution, stateful migration, descriptor-driven orchestration with forecasting-assisted migration, SD-WAN integration, and multi-site experimental validation. The 82% traffic reduction confirms the expected benefit of moving video analytics closer to the data source. The downtime observed across the migration scenarios (Fig. 9) reflects LiSO's post-migration strategy, in which the source instance remains active until the destination instance is ready and traffic can be redirected. The present results extend this approach to a multi-cluster cloud, edge, and end/far-edge orchestration context and separately characterize checkpoint overhead for state sizes up to 500 MB. Because no reactive Kubernetes baseline was evaluated, these results are presented as a characterization of LiSO's migration behavior rather than as a controlled comparison with existing recovery mechanisms.

X. LESSONS LEARNED

L1: Declarative descriptors are a prerequisite for automation. The OSR-AppD pipeline proves essential for enabling LiSO to make placement, scaling, and migration decisions without developer intervention. Hardcoded deployment manifests—even carefully maintained ones—cannot support the dynamic reconfigurations that CECC environments require.

L2: End/far-edge inference is the single highest-impact design decision. The 82% traffic reduction is achieved by a single architectural choice: running DeepStream on the Jetson rather than in the cloud. Compression, caching, and SD-WAN optimization remain useful, but they cannot compensate for a poor compute-placement decision when raw video backhaul dominates the traffic budget.

L3: The network is a first-class compute resource. The 2.1× FPS improvement from SD-WAN dynamic path selection demonstrates that network management must be co-optimized with compute placement. Systems that treat the network as a fixed background service leave substantial performance on the table.

L4: Post-migration strategy decouples service interruption from migration complexity. LiSO's post-migration approach maintains the source instance until the target instance is ready, bounding user-visible interruption mainly to the traffic-switch interval even when multiple microservices are migrated. Service interruption is primarily determined by the traffic-switch mechanism, while state size and migration duration mainly affect the length of the dual-instance migration phase.

L5: Proactive migration requires calibration. The α parameter in (1) must be tuned per deployment. An initial calibration procedure is recommended in which α is swept over a validation window and set to the value minimizing a weighted sum of missed and unnecessary migrations, with weights reflecting the operator's priorities.

L6: Architecture patterns generalize. The UC2 architecture—microservice decomposition, descriptor-based lifecycle management, SD-WAN interconnection, and stateful migration—is not parking-specific. It applies directly to industrial inspection (UAVs plus edge inference for defect detection), smart mobility hubs (multi-camera vehicle

classification), and environmental monitoring (UAV-based air quality mapping with real-time edge analytics).

XI. CONCLUSION

This paper has presented the design, implementation, and experimental validation of UC2, a UAV-augmented smart parking surveillance system deployed across a cloud, edge, and end/far-edge computing continuum and managed by the AC3 CECCM. Rather than a new standalone algorithm, the contribution is an integrated lifecycle-management pipeline whose stages were validated end-to-end on geographically distributed infrastructure: the OSR captures application intent and constraints; LiSO performs descriptor-driven deployment across heterogeneous Kubernetes and K3s clusters; the SD-WAN controller manages runtime connectivity; a checkpointer operator characterizes stateful-migration overhead; and a forecasting-assisted policy maintains service continuity during UAV hand-offs. The experimental evidence comprises at least an 82% reduction in network traffic through end/far-edge inference on UAV-mounted NVIDIA Jetson hardware, exceeding the 50% DoA target; an increase in video-analytics throughput from 53.7 to 114.9 FPS through SD-WAN-based dynamic path selection; a Jetson-based power-consumption experiment showing that sustained DeepStream primary detection increases board input power from 5.307 W to 6.137 W on average, corresponding to a 0.830 W or 15.6% compute-side power increase that can be reduced when the workload is migrated away from the UAV-class node; a systematic characterization of stateful-migration saving and restoration overhead across state sizes of 10–500 MB and up to five concurrent instances; and the downtime and total migration-time trends across seven heterogeneous migration scenarios shown in Fig. 9, where LiSO's post-migration strategy keeps the source instance active until the destination is ready.

Several directions extend this work. A controlled comparison against reactive Kubernetes recovery—such as eviction and delete–recreate—would further quantify the benefit of LiSO's post-migration strategy. Larger multi-UAV deployments would expose coordination effects, 5G channel contention, full UAV battery-autonomy behavior, and the need for concurrent, priority-aware migration scheduling. Distributed edge registries would accelerate checkpointed-image distribution as the fleet grows. The forecasting stage can be strengthened with ensemble or anomaly-detection methods that better capture peak resource demand. Finally, end-to-end zero-trust security—including mutual TLS between microservices and hardware-based attestation for edge nodes—would harden the pipeline for production use. These directions build on the validated pipeline presented here rather than addressing gaps within it.

ACKNOWLEDGMENT

The authors used ChatGPT (OpenAI) as an editorial assistant during manuscript preparation, specifically to improve clarity, refine sentence structure, harmonize terminology, and assist with formatting the manuscript according to the IEEE Access template. All technical content, experimental design, results, analyses, conclusions, and source citations were conceived, produced, and verified by them. All AI-assisted text was reviewed, edited, and validated by them, who take full responsibility for the integrity and accuracy of the manuscript.

REFERENCES

[1] X. Chen, Z. Guo, X. Wang, C. Feng, H. H. Yang, S. Han, X. Wang, and T. Q. S. Quek, "Toward 6G native-AI network: Foundation model-based cloud-edge-end collaboration framework," *IEEE Commun. Mag.*, vol. 63, no. 8, pp. 23–30, Aug. 2025, doi: 10.1109/MCOM.001.2400582.

[2] *OpenFog Reference Architecture for Fog Computing*, OpenFog Consortium, Feb. 2017.

[3] N. H. Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based Internet of Things services: Comprehensive survey and future perspectives," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 899–922, Dec. 2016, doi: 10.1109/JIOT.2016.2612119.

[4] A. Kadouma, I. Afolabi, A. Shukla, M. Bagaa, A. Ksentini, and M. Elmusrati, "A unified application profile model for microservices-based applications in the cloud-edge computing continuum," in *Proc. IEEE Conf. Netw. Function Virtualization Softw.-Defined Netw. (NFV-SDN)*, Athens, Greece, Nov. 2025, pp. 1–7, doi: 10.1109/NFV-SDN66355.2025.11349369.

[5] Q. Wu, Y. Zeng, and R. Zhang, "Joint trajectory and communication design for multi-UAV enabled wireless networks," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 2109–2121, Mar. 2018, doi: 10.1109/TWC.2017.2789293.

[6] A. Brogi, S. Forti, C. Guerrero, and I. Lera, "How to best deploy your fog applications, seriously," in *Proc. IEEE Int. Symp. Comput. Commun. (ISCC)*, Natal, Brazil, Jun. 2018, pp. 1–7.

[7] E. Casalicchio and V. Perciballi, "Auto-scaling of containers: The impact of relative and absolute metrics," in *Proc. IEEE 2nd Int. Workshops Found. Appl. Self Syst. (FAS*W)*, Hong Kong, Sep. 2017, pp. 207–214.

[8] J. Dogani, R. Namvar, and F. Khunjush, "Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey," *Comput. Commun.*, vol. 209, pp. 120–150, Sep. 2023, doi: 10.1016/j.comcom.2023.06.010.

[9] B. Jeong and Y.-S. Jeong, "Autoscaling techniques in cloud-native computing: A comprehensive survey," *Comput. Sci. Rev.*, vol. 58, Nov. 2025, Art. no. 100791, doi: 10.1016/j.cosrev.2025.100791.

[10] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog computing for the Internet of Things: A survey," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–41, Feb. 2019.

[11] S. Messaoudi, A.-E. Meliani, A. Mokhtari, and A. Ksentini, "Security and trust management in cloud edge continuum: AC3 project approach," in *Proc. IEEE 29th Int. Workshop Comput. Aided Model. Design Commun. Links Netw. (CAMAD)*, Athens, Greece, Oct. 2024, pp. 1–6, doi: 10.1109/CAMAD62243.2024.10942825.

[12] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140–147, Feb. 2018, doi: 10.1109/MWC.2017.1700011.

[13] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 1, pp. 108–119, Jan. 2017, doi: 10.1109/TETC.2015.2508382.

[14] A. E. Meliani, M. Bagaa, and A. Ksentini, "Layer-reuse aware optimization for efficient microservice migration in UAV edge systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Glasgow, U.K., May 2026, pp. 1–6.

[15] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using AI planning," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4226–4240, Dec. 2022.

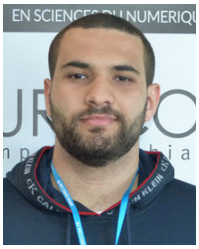
[16] A.-E. Meliani, A. Ksentini, M. Mekki, A. Kadouma, D. Amaxilatis, A. Ba, E. O. Coronado, J. Beredimas, V. Moulos, S. Sengupta, D. Klonidis, and C. V. Verikoukis, "AI-native CECC management architecture: Enabling scalable and intelligent cloud-edge computing," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Athens, Greece, Nov. 2025, pp. 1–6, doi: 10.1109/NFV-SDN66355.2025.11349565.

[17] AC3 Project Consortium. (May 2024). *D2.4 Business Analysis of the CECC and Use Case Requirements*. Accessed: Jul. 31, 2026. [Online]. Available: https://ac3-project.eu/wp-content/uploads/2024/07/Final-draft_AC3_Business-analysis-of-CECC-and-use-case-requirements_22.05.2024_FINAL.pdf

[18] M. A. Elghani, A. Sagar, A. Ksentini, and R. Knopp, "TimeTrack: A dataset for exploring temporal patterns and predictive insights into OpenAirInterface (OAI) CI/CD cluster," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Montreal, QC, Canada, Jun. 2025, pp. 1978–1983, doi: 10.1109/ICC52391.2025.11161298.

[19] A. E. Meliani, M. Mekki, and A. Ksentini, "Resiliency focused proactive lifecycle management for stateful microservices in multi-cluster containerized environments," *Comput. Commun.*, vol. 236, Apr. 2025, Art. no. 108111, doi: 10.1016/j.comcom.2025.108111.

ABDELHAK KADOUMA received the Engineering degree in computer science. He is currently pursuing the Ph.D. degree. He is a Software Engineer in Finland. He is conducting advanced research on application lifecycle management, ontology-driven systems, and semantic reasoning, with the University of Vaasa, Vaasa, Finland, in collaboration with Fingletek Oy, Espoo, Finland. His work focuses on AI-based application profiling, service orchestration, and policy-aware systems design within large-scale distributed environments. He has contributed to multiple research and innovation projects, including ontology and semantic-aware orchestration frameworks and backend systems development for real-world deployments. His research interests include cloud-native architectures, microservices, knowledge representation, and AI-driven automation.



ABDELGHANI MELIANI is currently a Doctoral Researcher with the Communication Systems Department, EURECOM, Biot, France. His research focuses on cloud–edge computing, microservice orchestration, and intelligent resource management in next-generation networks. His work primarily addresses challenges in the cloud–edge continuum, including service orchestration, lifecycle management of stateful microservices, and the integration of AI-driven approaches for scalable and resilient systems. His recent publications include work on lightweight resource exposure frameworks, proactive lifecycle management of microservices, and optimization techniques for microservice migration in UAV-assisted edge systems. He has contributed to several international conferences and journals within the IEEE ecosystem, including the IEEE International Conference on Communications.

QING LI is currently a Full-Stack Developer and a Software Engineer with Fingletek Oy, Espoo, Finland. He has broad experience across both front-end and back-end development, working on the design, development, and testing of applications that combine technical functionality with practical usability. His background encompasses collaborative software development, scalable systems design, API development, and continuous integration practices. He is motivated by creating efficient and maintainable systems that address real-world needs, and brings both analytical thinking and attention to detail to technical challenges in cloud-native and the IoT application development.

AMIT K. SHUKLA received the M.Sc. (Hons.) and Ph.D. degrees in computer science from South Asian University, New Delhi, India, in 2013. He is currently an Associate Professor with the School of Technology and Innovations, Computer Science, University of Vaasa, Vaasa, Finland. He has a *h*-index of 21 (Google Scholar) and more than 40 publications in SCI journals and CORE A/B-ranked conferences. He also holds two years of industrial experience as a Software Engineer on projects for Nokia and Sony. His research interests include computational intelligence, fuzzy sets and systems, industry 4.0, explainable artificial intelligence, anomaly detection, transfer learning, machine learning, and informatics. He was a recipient of the prestigious INSPIRE Fellowship from the Department of Science and Technology, Government of India, during his Ph.D. studies. He chaired the ACM Chapter of South Asian University from 2011 to 2012. He has chaired sessions with WCCI in 2020 and 2022, respectively. He serves as an Associate Editor for *Heliyon* (Elsevier) and the Section Board Member for *Axioms* (MDPI). He is an IEEE Young Professional.

MOHAMMED ELMUSRATI (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees (Hons.) in telecommunication engineering from the Electrical and Electronic Engineering Department, Benghazi University (formerly Garyounis University), Benghazi, Libya, in 1991 and 1995, respectively, and the Licentiate of Science (Hons.) and the Doctor of Science degrees in control engineering from Aalto University (formerly Helsinki University of Technology), Espoo, Finland, in 2002 and 2004, respectively. From 1991 to 1999, he was an Assistant Lecturer and a Lecturer with Garyounis University. From 1999 to 2004, he was a Researcher with the Control Engineering Laboratory, Helsinki University of Technology. He joined as a Lecturer with the University of Vaasa, Finland, in 2004. He has been a Professor there since August 2007, where he leads the Cyber-Physical Systems research team and heads the international program on Sustainable Autonomous Systems. He has authored more than 185 peer-reviewed articles, books, and technical reports. He is a member of the Society of Industrial and Applied Mathematics (SIAM) and the Automation Society in Finland.