future internet

MDPI

Article

Edge-to-Cloud Continuum Orchestrator Based on Heterogeneous Nodes for Urban Traffic Monitoring

Pietro Ruiu [1,*], Andrea Lagorio [1], Claudio Rubattu [1], Matteo Anedda [2,3], Michele Sanna [1] and Mauro Fadda [1]

[1] Department of Engineering, University of Sassari, 07100 Sassari, Italy; lagorio@uniss.it (A.L.); crubattu@uniss.it (C.R.); m.sanna51@studenti.uniss.it (M.S.); mauro.fadda@uniss.it (M.F.)
[2] Department of Electrical and Electronic Engineering, University of Cagliari, 09123 Cagliari, Italy
[3] CNIT (National Inter-University Consortium for Telecommunications), 09123 Cagliari, Italy
* Correspondence: pruiu@uniss.it

Abstract

This paper presents an edge-to-cloud orchestrator capable of supporting services running at the edge on heterogeneous nodes based on general-purpose processing units and Field Programmable Gate Array (FPGA) platform (i.e., AMD Kria K26 SoM) in an urban environment, integrated with a series of cloud-based services and capable of minimizing energy consumption. A use case of vehicle traffic monitoring is considered in a mobility scenario involving computing nodes equipped with video acquisition systems to evaluate the feasibility of the system. Since the use case concerns the monitoring of vehicular traffic by AI-based images and video processing, specific support for application orchestration in the form of containers was required. The development concerned the feasibility of managing containers with hardware acceleration derived from the Vitis AI design flow, leveraged to accelerate AI inference on the AMD Kria K26 SoM. A Kubernetes-based controller node was designed to facilitate the tracking and monitoring of specific vehicles. These vehicles may either be flagged by law enforcement authorities due to legal concerns or identified by the system itself through detection mechanisms deployed in computing nodes. Strategically distributed across the city, these nodes continuously analyze traffic, identifying vehicles that match the search criteria. Using containerized microservices and Kubernetes orchestration, the infrastructure ensures that tracking operations remain uninterrupted even in high-traffic scenarios.

Keywords: AI-based traffic monitoring; containerized microservices; edge computing; FPGA acceleration; Kubernetes orchestration

check for updates

Academic Editor: Qiang Duan

Received: 22 October 2025
Revised: 1 December 2025
Accepted: 10 December 2025
Published: 13 December 2025

Citation: Ruiu, P.; Lagorio, A.; Rubattu, C.; Anedda, M.; Sanna, M.; Fadda, M. Edge-to-Cloud Continuum Orchestrator Based on Heterogeneous Nodes for Urban Traffic Monitoring. *Future Internet* **2025**, *17*, 574. https://doi.org/10.3390/fi17120574

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The digital transformation is dramatically altering the way modern societies address mobility and urban infrastructure. With the proliferation of smart cities and the adoption of Cooperative, Connected, and Automated Mobility (CCAM), traffic systems are evolving from static, rule-based frameworks into dynamic, intelligent networks capable of real-time decision making and adaptation.

Determining how to allocate resources to maximize infrastructure efficiency is a well-established challenge. The edge-to-cloud layers can work together to provide effective and real-time data processing for a variety of applications and scenarios. This integrated system is known as the Computing Continuum, where computing resources are distributed across different layers. This approach leverages the cloud's scalable, high-performance

infrastructure and reliability alongside the edge's capability to perform low-latency, privacy-aware processing [1]. Emerging system's learning algorithms and new architectures will be able to forecast user mobility and service migration, predicting future service demands through continual and federated learning techniques. These approaches aim to ensure seamless service delivery in the face of ongoing change [2].

However, with the rise of Continuum Computing, new concepts and obstacles emerge regarding how to effectively manage this novel type of distributed network. Managing a continuum environment requires taking into account its inherent heterogeneity, which traditional cloud computing approaches often overlook [3]. For example, conventional task schedulers need to be adapted to incorporate additional parameters such as bandwidth consumption and usage costs, which are critical when selecting the most suitable node for a given task [4]. Although continuous computing is still a relatively new concept, it builds on a long history of research into efficient resource management across networks of computing nodes, an area that has been thoroughly explored in Cloud Computing and further advanced through recent developments in edge computing producing a wide range of algorithms, approaches, and frameworks for managing tasks and nodes [5–7].

Applications are typically composed of multiple containerized microservices, often implemented using Docker [8], which enables their deployment across the edge-to-cloud Continuum. This deployment strategy opens up new possibilities for delivering services with reduced latency and enhanced energy efficiency. Kubernetes has become a de facto standard for deploying services across multi-node systems. Its widespread adoption in cloud computing and microservices management is largely due to its powerful and reliable orchestration features. It is widely adopted for developing microservices, deploying applications, and managing containerized workflows. Kubernetes is an open-source container orchestration platform that automates the deployment, load balancing, and scaling of applications [9]. A container is a self-contained software unit that includes the necessary application settings (i.e., workload, state, and data) enabling the application to run independently and consistently within a pod. In a Kubernetes cluster, a pod constitutes the fundamental logical unit of replication, encapsulating one or more containers. A pod offers shared storage, networking, and a unified namespace, thereby defining the execution environment for its containers. The cluster constitutes the logical infrastructure in which pods are executed, managed, and orchestrated by users.

Modern applications exhibit increasingly complexity and frequently incorporate Artificial Intelligence (AI), leading to computationally intensive workloads that must also be handled at the edge nodes of a computing continuum infrastructure [10]. This growing complexity introduces dual requirements: on one hand, the need for functional flexibility to support diverse and evolving workloads; on the other, the necessity to meet stringent energy consumption constraints and real-time performance demands. Edge nodes based on heterogeneous hardware architectures, comprising both general-purpose processors and specialized platforms such as the AMD Kria K26 System-on-Module (SoM), offer a promising solution. These systems enable functional flexibility through software-based tools (e.g., Kubernetes) running on the general-purpose platform while also satisfying the demands of critical, often AI-based, tasks by leveraging dedicated hardware accelerators implemented via Field-Programmable Gate Array (FPGA) reconfiguration. This deployment strategy opens up new possibilities for delivering services with reduced latency and enhanced energy efficiency, as most inference and data processing tasks are executed locally on edge nodes equipped with heterogeneous hardware resources. By leveraging FPGA-based acceleration and reducing the amount of data transmitted to the cloud, the system minimizes both network overhead and power consumption while ensuring adaptive load distribution across the continuum.

This paper presents an edge-to-cloud continuum orchestrator designed for heterogeneous nodes integrating general-purpose processing units (GPPUs) and FPGA platform within an urban environment. The proposed system aims to minimize energy consumption by seamlessly integrating edge computing nodes equipped with video acquisition capabilities and cloud-based services. A vehicle traffic monitoring scenario serves as the case study to rigorously assess the feasibility of the proposed solution. The system is designed to operate in two distinct modes:

- A primary processing mode where each node analyzes a video stream to identify anomalies within a defined area. These anomalies include, but are not limited to, accidents, stationary vehicles, pedestrians, animals, and foreign objects;
- A multi-node re-identification that correlates vehicle images across video streams from multiple cameras.

Since the use case concerns the monitoring of vehicular traffic by AI-based processing of images and videos, specific support for application orchestration in the form of containers was carried out. The development concerned the feasibility of managing containers that leverage hardware acceleration through the Vitis AI design flow, specifically for accelerating AI inference on the AMD Kria K26 SoM. To facilitate the tracking and monitoring of specific vehicles, a Kubernetes-based controller node was designed.

Specific metrics have been identified in order to evaluate the performance of the system and the developed environment was transferred into a simulator to assess its feasibility. The evaluation methodologies employed relied on a fine-grained analysis, achieved through the decomposition of the system into elementary modules and the subsequent performance assessment of all the discrete components.

The main contributions of this paper are:

- Design of a container-based orchestrator for edge-to-cloud infrastructures with heterogeneous hardware (GPUs and FPGAs), supporting AI-driven urban traffic monitoring tasks while maximizing energy efficiency.
- Implementation of two AI-based urban traffic monitoring tasks deployable on edge devices, based on two real-world case study scenarios.
- Adoption of a hybrid evaluation strategy combining actual power measurements on ARM-based edge nodes and GPU-enabled servers with simulations to analyze the impact of scalability on overall system energy consumption.

The paper is structured as follows. Section 2 outlines the technical background and related works on edge-to-cloud continuum orchestrators. The architecture of the proposed system is shortly presented in Section 3, while scenario and use cases are described in Section 4. The orchestrator's design is detailed in Section 5, while the edge node is presented in Section 6. The performance evaluation methods used to assess the proposed solution, along with the corresponding results, are presented in Section 7. The proposed system was compared with existing frameworks in Section 8. Finally, the conclusions are drawn in Section 9.

2. Background

A comprehensive overview of the current state of the art in edge-to-cloud continuum orchestration systems is presented in this section. Additionally, it delves into the evolution and capabilities of container and microservice management platforms, emphasizing their adaptability to dynamic, resource-constrained, and latency-sensitive environments. In parallel, this work explores the integration of heterogeneous hardware accelerators, including GPUs, FPGAs, and AI-specific chips. These accelerators are increasingly essential for offloading and expediting computationally intensive tasks across the continuum. Fur-

thermore, we addressed the specific challenges and advancements in vehicular computing systems, where the mobility of nodes, the need for real-time processing, and the diversity of embedded resources pose unique orchestration and deployment requirements.

By surveying recent literature, frameworks, and technologies, this section aims to highlight the differences between existing solutions and the contribution provided by this paper.

2.1. Edge-to-Cloud Continuum Orchestrators

The emerging edge-to-cloud continuum architectural model integrates a wide range of heterogeneous components, from edge nodes to cloud infrastructures, significantly increasing management complexity. Managing data across distributed systems introduces new complexities including the organization of diverse data infrastructures, handling large and varied data volumes, ensuring seamless data access, streamlining internal data flow, and maintaining data integrity and availability. Due to the diverse nature of the devices and technologies involved, there is a critical need for hardware-independent and technology-neutral protocols [11]. The key characteristics of this approach include a hierarchical device structure with varying computational power, dynamic collaboration between different levels of the system, flexible and adaptive software, and pervasive intelligence at the edge. In addition, several critical questions have been raised to assess whether this technological vision is truly achievable [12]. Achieving optimal resource allocation in multi-node, heterogeneous environments frequently leads to highly non-convex optimization challenges. In this context, machine learning algorithms have proven to be a promising solution, offering near-optimal results for a wide range of complex optimization problems [13].

In [14], the authors presented a systematic review of the current techniques, recent systems, architectures, and frameworks that integrate Machine Learning and Data Analytics using leading learning paradigms (such as Online Learning, Transfer Learning, and Federated Learning [15]) to support collaborative training and decision making across the edge and cloud were analyzed. The main issue is the difficulty of gaining a comprehensive understanding of application workflow performance in such a distributed and heterogeneous environment, as well as optimizing that performance.

In [16], the authors introduced a Multi-Paradigm Deployment Model (MPDM) for Large-Scale Edge-Cloud Intelligence. This model formulates a multi-objective optimization problem design to generate a non-dominated Pareto front (PF) of deployment strategies that simultaneously balance system accuracy, service scalability, and deployment cost. An efficient constructive heuristic algorithm was developed that accesses both strong performance guarantees and efficient scalability when solving large-scale MDMP scenarios.

In [17], the authors outlined a TOSCA-based orchestrator designed to provide a flexible and scalable orchestration framework capable of managing the heterogeneity and dynamic conditions of edge environments, including improved performance (enabled by FaaS-driven parallelism), lower deployment costs, and better support for data-driven serverless workflows throughout the computing continuum.

In [18] the authors proposed a novel approach combining an optimal Mixed-Integer Linear Programming (MILP) model with a sub-optimal multi-agent rollout heuristic. This method is designed to orchestrate storage resources for hosting file fragments generated by erasure coding. These strategies aim to jointly optimize cost, latency, and average availability, while concurrently satisfying user-defined requirements.

Recent analyses of edge–cloud orchestration and continuum computing reinforce these findings. Gkonis et al. [11] provide a comprehensive survey highlighting open challenges in scalability, latency, and node heterogeneity. Furthermore, Sahu et al. [19] investigate

probabilistic scheduling policies aimed at reducing energy consumption while maintaining low end-to-end latency. Similarly, Sabbadin and Guérout [20] classify orchestration strategies that incorporate renewable energy integration, introducing sustainability as a key dimension of next-generation orchestration frameworks.

While current edge-to-cloud continuum orchestrators focus on resource management and deployment across the spectrum, the solution proposed in this paper distinguishes itself by explicitly integrating and optimizing for heterogeneous nodes, particularly those combining GPPUs with FPGA platforms like the AMD Kria K26 SoM. A core differentiating factor is our orchestrator's emphasis on minimizing energy consumption, a critical consideration often less prioritized in general-purpose continuum orchestrators.

2.2. Container and Microservice Management

Container and microservice management platforms have become essential tools in edge-to-cloud computing. However, their prevalent design is primarily tailored to homogeneous cloud infrastructures. This limitation curtails their effectiveness when applied to emerging computing paradigms. Specifically, default scheduling mechanisms struggle to manage workloads in heterogeneous systems, such as those prevalent in Continuum Computing, where nodes exhibit significantly in capabilities and constraints.

To overcome this challenge, the authors in [21] replaced the Kubernetes default scheduler with a custom implementation. This approach provides complete control over pod-to-node assignments, enabling the development of scheduling algorithms that account for all parameters relevant to deploying services across a continuum environment.

In [22], a cost-effective scheduling methodology for Kubernetes cluster was presented. This approach demonstrated benefits using a realistic workload, specifically vehicular cooperative perception showing a cost reduction compared to the default Kubernetes scheduler while maintaining the same quality of service.

In [23], the authors presented COgnitive Decentralized Edge-Cloud Orchestration (CODECO), an orchestration framework that leverages decentralized Artificial Intelligence methods to identify optimal deployment environment for supporting next-generation Internet applications across a mobile and heterogeneous edge-to-cloud Continuum. This work also developed a novel scheduler to manage pod-to-node matching decisions.

Orchestrating microservice applications across a federation of globally distributed Kubernetes clusters presents a complex and challenging optimization problem. This task is not only computationally demanding but also necessitates balancing multiple, often conflicting performance goals such as latency, deployment cost, and service disruption service. Traditional methods frequently over simplify this multi-objective problem by aggregating objectives typically through linear combinations. However, defining clear quantitative preferences among such diverse objectives *a priori*, especially via simple linear weighting proves difficult in practice.

The robust solution presented in [24] employs true Multi-Objective Optimization (MOO) techniques to derive a PF comprising a set of optimal trade-offs. This empowers the orchestrator to examine all viable "best" solutions and make a selection that precisely fits specific operational needs. To solve the MOO problem, the paper employs cutting-edge Multi-Objective Evolutionary Algorithms (MOEAs), demonstrating their effectiveness in this context. The presented results underscore the practical advantages of the MOO approach, yielding dozens of nondominated solutions and offering a well-distributed coverage across the objective space.

In [25], the authors describes IoTwins, a cost-effective platform for coding, deploying, and managing distributed Digital Twin (DT) applications. This platform operates on a TOSCA-compliant orchestration system, supporting to support the provisioning of

containerized applications across a Continuum Computing infrastructure that integrates both cloud and edge resources. It provides users with tools to configure generic Docker containers as modular components, link them into complex service chains. These chains can be deployed within heterogeneous environments that integrate both cloud and edge computing nodes.

The evolution of container-based deployments and microservice-driven architectures is also influenced by the increasing convergence between AI workloads and distributed computing infrastructures. Prangon et al. [26] discuss how edge-native containerized environments must dynamically adapt to heterogeneous resource availability while supporting AI inference continuity.

Existing container and microservice management platforms are predominantly designed for homogeneous cloud environments, facing significant challenges when applied to highly heterogeneous edge infrastructures. Our approach offers a distinct advantage by providing specialized support for orchestrating containerized applications with hardware acceleration, specifically leveraging the Vitis AI design flow for AMD Kria K26 SoM. This enables efficient and performant execution of AI-based workloads on diverse edge devices, an area where many current solutions lack comprehensive capabilities.

2.3. Cloud-Edge Continuum for Reconfigurable Heterogeneous Systems

In [27], the authors examined the feasibility of using containers to deploy accelerator virtualization frameworks and the possibility of migrating these containers across cloud-edge nodes. This work introduced the AVEC framework, an ongoing project focused on accelerator virtualization in cloud-edge environments. For the authors, leveraging Docker containers offers full isolation at both the package and operating system levels, thereby facilitating the fast and flexible deployment of virtualization frameworks across various network devices.

In [28], the ARTICo3 framework [29] was enhanced to support cloud environments featuring a host machine connected to an FPGA via PCIe. ARTICo3 has been incorporated into a Kubernetes-managed infrastructure, adopting a microservice approach. This integration provides users with the capability to effortlessly deploy and accelerate their applications on any node throughout the cloud-edge continuum.

The work in [30] introduced Robustness via Elasticity Control (RvE) Accelerators, a new model of reusable software components. RvE Accelerators are specifically designed with inherent cross-layer and cross-system capabilities, enabling them to sustain high service quality even when contending with conflicting optimization objectives across different layers.

In [31], the authors presented the design of the SERRANO architecture, emphasizing the benefits of hardware acceleration within a serverless computing paradigm across the cloud-edge continuum. The core of this work involved optimizing the complete execution workflow by extending the SERRANO SDK and its deployment tools, providing advanced acceleration capabilities.

The work in [32] introduced TF2AIF, a tool specifically designed to streamline the development and deployment of AI models throughout the heterogeneous cloud-edge continuum. TF2AIF achieves by automating the conversion of models between different AI frameworks, thereby ensuring compatibility with a wide range of platforms.

Regarding heterogeneous acceleration, Seamless MEC clusters have recently been explored as deployment platforms for hardware-accelerated AI inference across GPU- and FPGA-based nodes [33]. Although this approach demonstrates the feasibility of seamless accelerator integration, it relies on a fixed orchestration layer and does not explicitly address energy-aware placement policies.

Though other research addresses aspects of accelerator virtualization and the cloud-edge continuum for reconfigurable systems, our contribution stands out through the development of a dedicated edge-to-cloud orchestrator specifically tailored for urban traffic monitoring. This orchestrator seamlessly integrates GPPUs and FPGAs with a primary focus on minimizing energy consumption, offering a concrete and evaluated system for a demanding AI-based application. This provides a practical demonstration of how reconfigurable heterogeneous systems can be effectively managed and leveraged for real-world scenarios.

2.4. Vehicular Computing Networks

The work in [34] proposes an innovative distributed service placement framework designed for dynamic load management of smart mobility services within the edge-to-cloud continuum. This framework seeks to optimize Quality of Service (QoS), reduce service deployment costs, balance workloads, and contribute to sustainability goals. Comprehensive evaluations conducted with real-world infrastructure and traffic data from Munich, highlight the framework's ability to improve computing resilience and support the development of self-adaptive ICT systems.

In [35], the authors introduced hierarchical and analytical models to assess the availability of intelligent traffic management infrastructure. These infrastructures include a camera, an edge node, a fog node, and a cloud environment. The study key's contribution lies in its comprehensive availability evaluation from the sensor layer up to the cloud, encompassing all intermediate layers, including edge and fog.

In [36], the authors explored how the computing continuum can be leveraged to effectively handle urban mobility tasks within large-scale computing environments. Specifically, they utilized an edge-to-cloud architecture to process geotagged data generated at the network edge by taxis, vehicles, and smartphones. This gathered data was then processed in real time using machine learning techniques to address various everyday challenges.

The work in [37] introduced a modular orchestration framework built on top of Kubernetes to enhance application placement decisions. This framework integrates performance and network metrics, real-time resource usage, and expected round-trip times between service dependencies, optimizing workload distribution. Moreover, it supports dynamic workload re-scheduling to adapt to changes in application behavior or network latency, thereby maintaining stable performance.

In [38], the authors carried out a secondary incident detection, analysis, and prediction integrating real-time traffic data from a variety of heterogeneous sources, including in-vehicle sensors (e.g., tilt and accelerometers), roadside units, and surveillance cameras. Edge devices deployed throughout the transportation network process this data to evaluate key performance indicators such as traffic density, incident clearance time, collision types, and prediction accuracy. After extensive training and testing on large dataset, a comparative analysis was conducted using machine learning algorithms, specifically Linear Regression, Random Forest, K-Nearest Neighbors (KNN), and Support Vector Regression. The system was implemented and thoroughly evaluated using simulation tools such as OMNET++, Veins, and SUMO, based on parameters like vehicle speed, traffic flow, and round-trip time reduction.

Existing vehicular computing network frameworks often address concerns such as dynamic load management, QoS, and workload balancing. Our system differentiates itself by providing a comprehensive solution for AI-based vehicular traffic monitoring, incorporating both anomaly detection and multi-node vehicle re-identification. By leveraging heterogeneous edge nodes (GPPUs and FPGAs) and Kubernetes-based orchestration, our work delivers real-time performance and low power consumption, specifically optimized

for complex urban mobility scenarios, thus moving beyond general-purpose vehicular communication or computation paradigms.

2.5. Discussion and Research Gap

Compared to the approaches in literature, the proposed orchestrator extends the state of the art in three main directions. First, it explicitly targets heterogeneous edge nodes that integrate both general-purpose and FPGA-based computing units, and dynamically manages them through a Kubernetes-based orchestration layer connected with Azure IoT Hub. Second, while many existing frameworks address energy efficiency as a secondary metric, our architecture natively embeds an energy-aware orchestration logic that optimizes workload placement according to node characteristics and energy profiles. Third, the proposed design aligns with standardized edge–cloud frameworks, particularly the ETSI MEC and 3GPP Service-Based Architecture (SBA) principles, ensuring interoperability and portability across different domains. Therefore, the system not only builds upon the foundations laid by recent research but also provides a concrete implementation pathway toward scalable, secure, and energy-efficient orchestration in real-world smart city deployments. Table 1 summarizes the main characteristics of these frameworks. "Heterogeneity" indicates whether the framework natively supports heterogeneous and distributed execution across edge–cloud resources. "Hardware acceleration" specifies supported compute substrates (GPU, FPGA, or mixed). "Energy focus" distinguishes between implicit optimisation effects, partial or secondary consideration, and explicit primary optimisation goals. "Validation" refers to the evaluation methodology adopted (simulation, prototype, demonstrator, or hardware-based testing).The works summarized in Table 1 were selected as representative examples among the broader literature surveyed in this section. They all propose concrete orchestration or deployment frameworks for the edge–cloud continuum, rather than purely analytical models, domain-specific applications, or survey-only contributions. In particular, these frameworks address at least one of the following aspects: continuum-wide service orchestration, container-based deployment across heterogeneous nodes, or the integration of hardware accelerators. Other cited works focus on complementary topics such as availability modelling, traffic prediction, or architectural vision, and are therefore discussed in the text but not included in the comparative table to keep the analysis focused and concise. As highlighted, most existing frameworks either do not explicitly target energy-aware orchestration, or they do not combine heterogeneous GPU/FPGA edge nodes with a standard-compliant edge-to-cloud orchestration stack. In contrast, the proposed framework jointly addresses these three dimensions, providing a concrete implementation for urban traffic monitoring.

Table 1. Comparison of selected edge-to-cloud orchestration frameworks.

Framework	Heterogeneity	HW Acceleration	Energy Focus	Validation
MPDM [16]	Yes	None	Partial	Sim
TOSCA [17]	Yes	None	Explicit	Proto
MILP [18]	Partial	None	Partial	Sim + analitic
CODECO [23]	Yes	None	Implicit	Proto
IoTwins [25]	Yes	None	Partial	Testbed
AVEC [27]	Yes	GPU	No	Proto
ARTICo3 [28,29]	Yes	FPGA	Implicit	Proto
SERRANO [31]	Yes	Mixed	Explicit	Demo
Seamless MEC [33]	Yes	Mixed	No	Proto
Proposed framework	Yes	FPGA + GPU	Explicit	Sim + HW

3. System Architecture

The system proposed in this project aims to advance urban mobility by intelligently handling and analyzing data from the edge to the cloud. Its primary goal is to enhance traffic monitoring capabilities and the effective detection of specific information within urban and suburban environments.

The proposed infrastructure comprises a network of heterogeneous computing nodes equipped with video acquisition modules. The diagram shown in Figure 1 presents the logical architecture of the envisioned distributed orchestrator, highlighting both the system's core architectural components (right side) and the considered use case involving urban vehicular traffic (left side). Certain nodes in the network are integrated with AMD multiprocessor boards, which incorporate an ARM processor, a GPU, and a reconfigurable FPGA.

The targeted use case for this system involves video-based traffic monitoring within a smart city context. The orchestrator operates in order to manage two main tasks:

- Anomaly Detection (AD) entails the analysis of video streams to detect irregular events such as accidents, stationary vehicles, pedestrians, animals, or foreign objects on urban or rural roadways.
- Vehicle Re-Identification (VRI) supports the re-identification of vehicles by analyzing images extracted from video feeds across multiple cameras. A typical application includes tracking a specific vehicle along a monitored route within the camera network.

The main hardware and software components of the system (Figure 1) are briefly presented and will be thoroughly described in the next dedicated sections. The proposed orchestration model follows design principles consistent with ETSI MEC and 3GPP SBA. Specifically, the system adopts (i) a microservice-based architecture with independently deployable functions exposed via lightweight APIs; (ii) a hierarchical edge–cloud split, where inference tasks execute locally at the edge while coordination and lifecycle management are handled centrally; (iii) container-based packaging ensuring application portability across heterogeneous hardware; (iv) an event-driven communication model through a message bus. These elements collectively reflect MEC's application portability and orchestration concepts as well as the SBA paradigm of service exposure and stateless, interoperable functions.

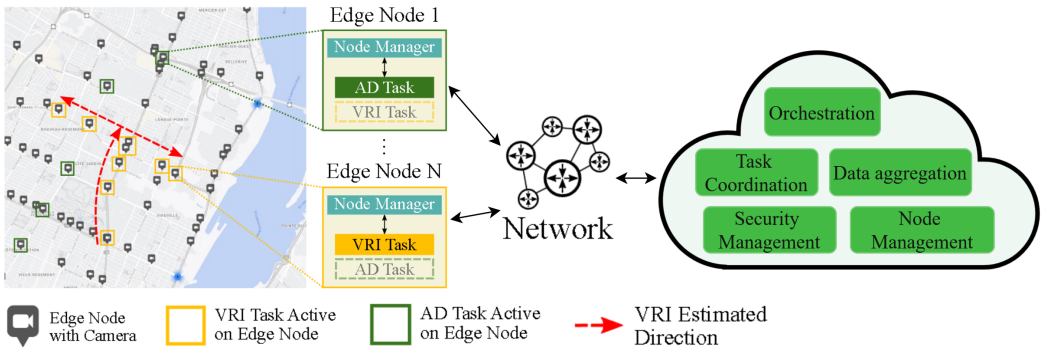

Figure 1. System architecture: use case involving urban vehicular traffic (left side) core components (right side).

3.1. Controller Node

The Controller Node represents a key component in the system architecture, responsible for making context-aware decisions across various application scenarios based on input received from edge nodes. Its primary role is to manage, coordinate, and scale the infrastructure used for monitoring vehicles flagged for anomalous behavior or law enforcement interest. It operates on Kubernetes [39] and Azure Cloud (Azure Kubernetes Service) [40], ensuring efficient interaction between camera-based detection units and

cloud-hosted processing services. By acting as a middleware layer, the Controller Node processes, routes, and monitors tracking requests while maintaining system resilience, scalability, and real-time performance.

As depicted in Figure 1, the Controller Node encompasses several key functional blocks:

- Orchestration: this block likely handles the deployment, scaling, and management of containerized applications and services across the edge and cloud infrastructure, ensuring efficient resource utilization and task execution. This aligns with the Controller Node's role in managing and scaling the infrastructure.
- Data Aggregation: this component is responsible for collecting and consolidating data from various edge nodes, providing a centralized point for processing information received from detection units. This supports the Controller Node's function of processing input from edge nodes.
- Task Coordination: this block ensures the synchronized execution and management of different tasks, such as AD and VRI, across the distributed network. This is consistent with the Controller Node's role in coordinating the monitoring infrastructure.
- Node Management: this function involves overseeing the status, health, and resources of both master and slave nodes within the Kubernetes cluster, ensuring their optimal performance and availability. The Controller Node is composed of a Master Node and multiple Slave Nodes, each implemented as a Kubernetes worker node, with application pods running Flask-based services for communication, processing, and orchestration logic.
- Security Management: this block is crucial for maintaining the security and integrity of the entire system, encompassing aspects like authentication, authorization, and data protection across the edge-to-cloud continuum.

A simple user interface was implemented to facilitate interaction with the system and support the Controller Node's operations, providing a practical means to test, debug, and validate the orchestration of edge and cloud resources during development, ensuring that context-aware decision making and real-time tracking functioned as intended (see Section 5.3.3 for further details).

3.2. Edge Nodes

The edge nodes are based on a heterogeneous computing platform that supports the orchestration of containerized workloads, including those leveraging FPGA-based hardware accelerators, through Kubernetes. These nodes are designed to perform tasks related to the vehicular traffic monitoring use case, satisfying real-time performance requirements and low power consumption. Such constraints are effectively addressed by the energy-efficient architecture of the heterogeneous platform, which combines general-purpose processing with dedicated AI acceleration. Within the Kubernetes-managed infrastructure, each node acts as a worker node capable of running containerized services corresponding to the AD and VRI tasks. Furthermore, the edge nodes interface directly with video sensors that provide the input data streams necessary for task execution. The adoption of the AMD Kria K26 SoM is instrumental in this context, thanks to its integrated support for a variety of standard interfaces tailored for computer vision applications, as well as its native compatibility with AI-accelerated processing pipelines.

4. Scenario and Use Cases

The case studies were selected based on their relevance and were chosen from those included in the AI City Challenge (https://www.aicitychallenge.org/, accessed on 9 December 2025). As briefly outlined in previous section, the system is designed to

operate in two modes: (i) AD serving as the primary process for analyzing video streams to pinpoint anomalies (e.g., accidents, stationary vehicles, pedestrians, animals, objects, etc.) on urban or extra-urban roads. (ii) VRI which enables tracking specific vehicles through camera-covered paths by re-identifying them from video stream images. The software and its runtime environment are encapsulated within a container. This approach, leveraging lightweight virtualization techniques, reduces overhead while maintaining a small application size [41], thereby facilitating the seamless movement of the application to the most suitable processing node.

4.1. Anomaly Detection

The AD module processes the video stream to estimate the background image. Subsequently, YOLO, a deep learning-based object detection algorithm [42], identifies specific objects within the scene, interpreting their presence as an anomaly. Background computation allows the system to isolate motion or anomalies within the scene, while YOLO provides robust detection of objects within each frame. Both the parameter settings and the overall configuration follow widely adopted guidelines from the scientific literature [43].

Figure 2 illustrates an example of AD on an extra-urban road, where a stationary vehicle—absent from the computed background—is identified and localized with a bounding box, highlighting its unexpected presence.

Figure 2. Example of background modeling and anomaly detection. The detected anomaly is highlighted within the red square.

Upon AD, the responsible node transmits a report to the Controller Node. The orchestrator may then initiate the following actions:

- Transmitting an alert signal to a control center, enabling a human operator to access relevant video streams and make informed manual decisions;
- Initiating enhanced video stream analysis at the edge by leveraging the embedded computational capabilities thus allowing the deployment of more complex detection models;
- Transferring the video stream from the edge to the cloud for advanced or centralized processing;
- Activating the VRI task to enable tracking of a specific vehicle.

All the active nodes perform the AD task as default operational mode, and the Controller Node can only interrupt it if the specific node is required for a VRI task.

4.2. Vehicle Re-Identification

The VRI task is a specialized operation designed to track a vehicle's movement within a defined geographic area. This task requires explicit initiation via an external request that specifies the vehicle to be tracked. For instance, in the case of a vehicle pursuit scenario the task is triggered by sending the target vehicle's image and last known location to the Controller Node.

Based on the reported location, the Controller Node activates the VRI task on nodes geographically positioned to cover the vehicle's likely. This area is centered around the location where the anomaly was initially detected. The vehicle's image is then transmitted to these nodes, which subsequently begin executing the VRI task by processing video streams

from their associated cameras to visually match the reference image. The detection module is based on a finetuned version of DINOv2 [44], trained on the Very Wild dataset [45] to ensure robust and domain-adapted feature extraction for vehicle re-identification.

If a node successfully identifies a matching vehicle, it transmits a message including the newly captured image, to the Controller Node. The Controller Node then recalculates the search area, centering it on the reporting node's location. Nodes within this updated area are activated for the VRI task, while those outsider revert to performing the default AD task.

This dynamic reallocation facilitates continuous vehicle tracking by updating and shifting the monitoring region whenever the vehicle is detected. If the vehicle is no longer identified by any camera within the active area, the VRI task terminates, and all nodes revert to their default AD operations.

Figure 3 illustrates the execution of VRI task, tracking a vehicle's movement through an urban environment, specifically a real city in Italy. The blue icon on the left represents the vehicle, moving from left to right along the depicted route. Each 300-m diameter circle on the map indicates a potential vehicle movement area at various tracking stages. These areas guide the system in activating specific camera nodes for the VRI task based on the vehicle's last known position. Red markers denote camera nodes that have positively identified the target vehicle, while green markers represent nodes actively participating in the VRI task but without current detection.

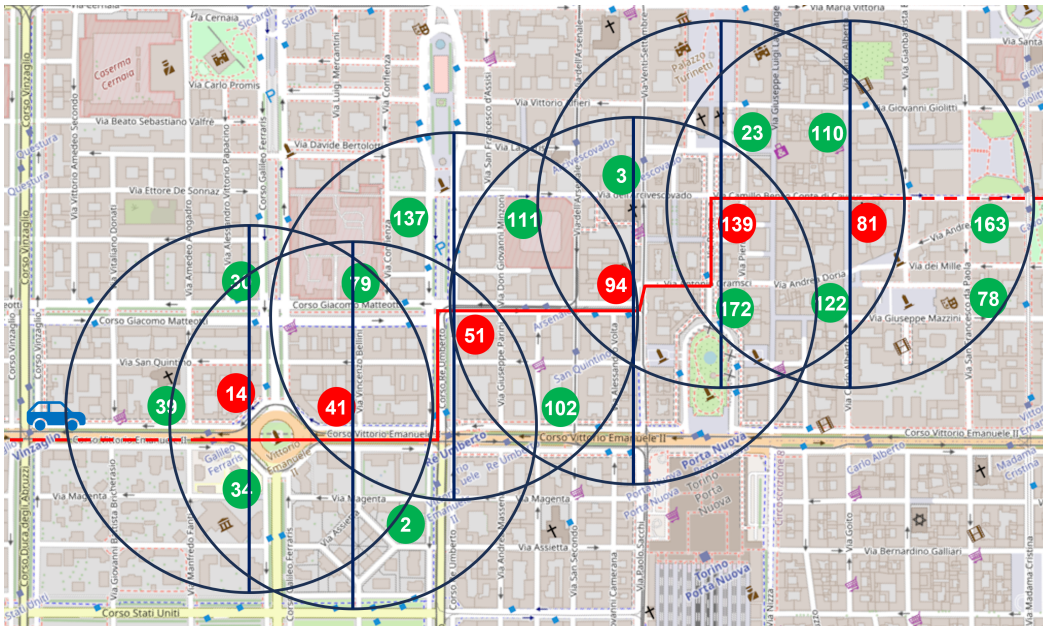

Figure 3. Illustration of the VRI task in an urban environment: red markers are camera nodes that have positively identified the target vehicle, green markers represent nodes actively participating in the VRI task, while the red line traces the vehicle's confirmed through successive detections.

This distribution shows how the system dynamically reallocates resources: cameras that detect the vehicle become focal points, prompting the Controller Node to redefine the monitoring area centered on the latest positive detection. The red line traces the vehicle's confirmed through successive detections. This approach enables adaptive, real-time tracking as the system progressively shifts its focus based on ongoing detections, thereby optimizing computational resource utilization and coverage. During the preliminary phase, a synthetic dataset was employed for the evaluation of the VRI model, due to the lack of publicly available datasets specifically tailored to this use case [46].

5. Orchestrator's Design

This section details the implementation of the orchestrator and its operational mechanisms facilitating the execution of the two distinct tasks: AD and VRI.

5.1. Technology Stack and Platform Overview of the Controller Node

The Controller Node utilizes a modern, containerized architecture built around the Python programming language such as Flask, Redis and MongoDB. Flask (https://flask.palletsprojects.com/en/stable/, accessed on 9 December 2025), a lightweight web framework suitable for microservices, handles the core application logic, including API interactions and inter-node coordination. Background task execution is managed via a Celery-based asynchronous queuing mechanism, enabling the distributed and fault-tolerant processing of high-frequency events such as vehicle detections and tracking updates. Redis (https://redis.io/, accessed on 9 December 2025) serves a dual purpose within the system, functioning as both the message broker for Celery workers to facilitate reliable task dispatch and execution, and as a temporary in-memory data store for caching intermediate tracking states, enabling rapid data access during real-time processing. Concurrently, MongoDB (https://www.mongodb.com/, accessed on 9 December 2025) acts as the central document-oriented database, storing structured data pertinent to tracking sessions, alerts, and system logs.

All system components are containerized using Docker and deployed within a Kubernetes cluster, ensuring scalability, resilience, and automated orchestration. Helm charts are utilized for deployment and service configuration. Identity and infrastructure access within the cloud environment are managed through pod-level managed identities and Role-Based Access Control (RBAC). Logging, health monitoring, and inter-service communication are fully integrated using standard Kubernetes primitives, rendering the Controller Node a modular, scalable, and cloud-native core of the overall system. Each Camera Node is associated with a Slave Node, while the Master Node coordinates deployment, task distribution, and centralized control.

The proposed deployment adopts a hybrid orchestration architecture to balance feature completeness and edge-level efficiency. The orchestration control plane and management components run on an x86-based workstation configured with Kubernetes Client (v1.34.1) and Server (v1.32.5) versions, providing full compatibility with standard Kubernetes APIs, Helm-based deployment, cluster management, and monitoring extensions. Conversely, the AMD Kria K26 SoM edge nodes operate using K3s, a lightweight CNCF-certified Kubernetes distribution designed for resource-constrained devices. K3s offers reduced memory footprint, simplified service execution, and low-overhead startup characteristics, making it suitable for embedded inference workloads. This design enables full interoperability across cloud and edge tiers while ensuring realistic deployment conditions for heterogeneous nodes.

The Master Node, running a master pod, functions as the centralized orchestrator of the tracking infrastructure. It is responsible for deploying and decommissioning Slave Nodes and Camera Nodes dynamically. Each camera node at the edge level has a specific geographical location. Based on this, the master node decides to assign a specific slave node, which acts as a gateway. Each slave node can manage up to 30 camera nodes within a 1 km radius and is initialized either when dealing with the first camera node of the system in a specific geographical location (i.e., no other camera nodes are present within 1 km) or when the other slave nodes in that specific area are saturated. Upon receiving a deployment request, the Master Node scales the Kubernetes node pool, launches a new Slave Node pod, and registers its metadata in the central MongoDB database [47]. Beyond node management, the Master Node serves as the gateway for tracking requests originating from external

authorities. It intelligently determines and assigns tasks to the most suitable nearby camera nodes. Furthermore, it aggregates tracking results from various Slave Nodes, maintaining a geo-spatial map of all node positions, and provides essential coordination logic to ensure seamless vehicle tracking across different node boundaries. To prevent blocking calls and enable parallel task handling without compromising system responsiveness, the Master Node leverages Celery with Redis for asynchronously queuing internal operations [48].

Slave Nodes process anomaly alerts from their Camera Nodes, forwarding them to the Master Node for storage and evaluation. Slave Nodes depend on the Master Node for coordination, task distribution, and access to the registry of neighboring nodes via geospatial queries. This creates a tight feedback loop: the Master Node provides updated network topology and control instructions, while Slave Nodes supply ground-level anomaly data and status updates. Upon receiving a tracking assignment, the Master Node first directly involves all camera nodes within a 300-m radius, switching from AD to VRI task, and verifies vehicle detection. If detected, it logs the result and propagates the request to neighboring Camera Nodes to continue the tracking chain. If no match occurs within a predefined window, the task is managed based on system policy, either expired or revived.

The relationship between the Controller Node elements and Camera Nodes operates on an inherently hierarchical and reactive paradigm:

1. Camera Nodes, functioning as localized detection agents, transmit alerts and data raw to their assigned Slave Nodes;
2. The Slave Nodes analyze this incoming data, coordinate with geographically proximate nodes, and escalate pertinent nodes and findings to the Master Node as required;
3. While edge devices perform preliminary data analysis, the controller-side pods (Master and Slave Nodes) are primarily responsible for orchestrating the system-wide tracking logic and ensuring seamless continuity of surveillance across the entire distributed network.

Each Camera Node is assigned a Slave Node to ensure strict resource isolation, simplified coordination, and high scalability. The Master Node maintains its own task handling, processing incoming tasks in parallel through replicated service instances and coordinating results within the node. This design enhances fault tolerance, as failure or overload in one node impacts only its associated Camera Node, leaving the remainder of the system unaffected. From a scalability perspective, the infrastructure can dynamically adjust the number of active instances within each node based on workload demand, enabling fine-grained performance tuning and efficient utilization of cloud resources. Future extensions of the orchestration layer could adopt Kubernetes Custom Resource Definitions (CRDs) and operator patterns to encapsulate application-specific control logic. This evolution will improve maintainability, enable automatic reconciliation of edge resources, and enhance compatibility with standard cloud-native ecosystems.

5.2. Scalability and Deployment

Application layer scalability is achieved through a dynamic deployment model. The Master Node is manually deployed due to its centralized control responsibilities and limited replication requirements, ensuring full administrative oversight during setup. Conversely, Slave Nodes and Camera Nodes are deployed automatically if necessary.

To achieve cloud-to-edge communication that is between the Camera Node (edge) and the Master and Slaves Nodes (cloud), an IoT Hub on Azure was created, as shown in Figure 4. The IoT Hub on Azure serves as the central cloud-based service that enables secure communication between edge devices and the cloud. In this architecture, each Camera Node is registered as an edge device within the IoT Hub. This registration process creates a secure identity for the device and generates the connection string required to

establish a trusted link between the edge and the cloud. The Azure IoT Edge Agent plays a crucial role in managing the interaction between the Camera Nodes and the cloud. It is responsible for securely connecting the devices to the Azure Kubernetes cluster while also handling the deployment and monitoring of workloads on the edge devices. In this way, the agent ensures that each Camera Node remains properly synchronized with the IoT Hub and can reliably execute the tasks defined from the cloud. Finally, the Azure IoT Edge Runtime is the software component installed directly on each Camera Node. It provides the necessary environment for running containerized workloads locally on the device, reducing the reliance on constant cloud connectivity. By enabling local data processing and communication, the runtime ensures lower latency and greater resilience, while still maintaining a secure and managed connection to the Azure IoT Hub through the IoT Edge Agent. At the same time, a dedicated worker node is deployed and the Camera Node is associated to a Slave Node. Each Slave Node hosts the necessary components to communicate and manage data received from connected Camera Nodes. The Master Node uses dedicated application pods to allow switching from AD to VRI task for tracking operations performed by the involved Camera Nodes, thereby ensuring a strict one-to-one mapping between each Camera Node and its cloud-side processing environment.

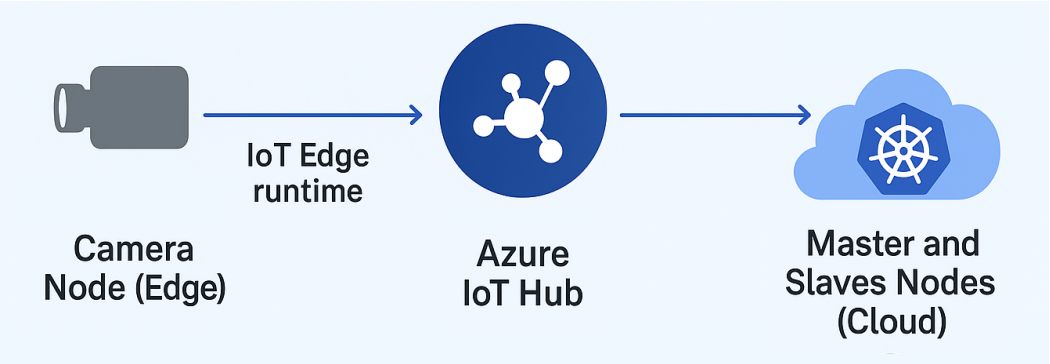

Figure 4. Cloud-to-Edge communication.

5.3. Traffic Monitoring Application

This section provides additional details on how the orchestrator manages the two tasks in the case study. It also includes further information on the implemented dashboard.

5.3.1. AD Algorithm

This subsection presents the algorithm carried out to allow the AD execution, directly performed at the Camera Node level, based on predefined behavioral rules configured within each device. When a condition meets the criteria for an anomaly, the Camera Node immediately sends an alert to its associated Slave Node. This alert includes the anomaly type, detection time, and basic metadata. Upon receiving the alert, the Slave Node forwards it without delay to the Master Node for central processing. Each anomaly is treated independently, and the detection logic is optimized for rapid decision making to allow immediate propagation and potential triggering of higher-level actions, such as initiating a tracking task.

5.3.2. VRI Algorithm

The VRI task within the system is initiated exclusively by the Master Node, which serves as the central coordinator for launching and distributing tracking tasks. A tracking request may originate from external authorities based on operational needs. The Master Node determines whether a new tracking task should be activated, depending on predefined preferences configured by the authorities. Once a decision is made to start tracking,

the Master Node selects the Camera Node (engaged node) geographically closest to the initial detection location, or to a location explicitly specified by the authorities to obtain a list of all Camera Nodes located within a 300-m radius. This radius defines a circular tracking zone, with the engaged node at its center. The vehicle itself is tracked using an image, or generic information as model and color.

The tracking request is considered active only within a specific time window, defined as a cycle duration. The tracking request is considered active only within a specific time window, defined as a cycle duration. During this period, if any Camera Node detects a vehicle matching the tracking profile immediately informs the Master Node. This new node then becomes the center of the next 300-m tracking zone, and the cycle repeats. The task reaches a valid stopping condition when the vehicle not being detected in any node during a full cycle, or being detected twice consecutively by the same node.

The length of each cycle is configurable and designed to reflect both the geographical context and urban mobility characteristics of the deployment area. It is influenced by factors such as node density, average inter-node distances, and local traffic behavior. This duration will be defined per city as part of the system deployment plan and can be adjusted later as a configurable preference by the authorities. The cycle-based propagation mechanism ensures that tracking is reactive, focused, and resource-efficient, while maintaining the flexibility to adapt to various urban environments and law enforcement strategies.

5.3.3. Monitoring and Dashboard Interface

The proposed architecture has been conceived to ensure compatibility with emerging edge-to-cloud interoperability standards. At the edge layer, the system aligns with the ETSI Multi-access Edge Computing (MEC) framework by supporting localized service execution, low-latency analytics, and standardized service exposure through APIs. This alignment facilitates the deployment of AI-driven monitoring services near data sources while preserving the modularity and scalability of cloud-native architectures. At the orchestration and cloud levels, the proposed system embraces the 3GPP Service-Based Architecture (SBA) principles, adopting modular microservices and standardized interfaces for inter-component communication. This approach promotes interoperability with telecommunication infrastructures and external network functions, enabling the orchestrator to be seamlessly integrated within broader 5G and IoT ecosystems. Through this dual alignment, the orchestrator can be adapted to a wide range of domains beyond urban mobility, such as industrial IoT, environmental sensing, or public safety, while ensuring adherence to recognized standards that support interoperability, scalability, and future evolution.

5.4. Data Privacy and Security Considerations

The proposed orchestrator has been designed with strong attention to data protection aspects, which are of paramount importance in urban surveillance systems. Two complementary dimensions are considered, that is, security and privacy. From a security perspective, the system minimizes exposure by design. Edge nodes perform local video processing and transmit only lightweight informational messages, such as anomaly alerts or vehicle identifiers derived from AI inference. This approach drastically reduces the amount of raw data circulating across the network. In cases where partial offloading to the cloud is required, e.g., when performing advanced analysis or cross-node coordination, standard protection mechanisms are applied, including encrypted communication channels (TLS), secure virtual private network (VPN) tunnels, and mutual authentication through the Azure IoT Hub. Role-Based Access Control (RBAC) within the Kubernetes cluster ensures that each microservice and node operates under the principle of least privilege, preventing unauthorized access to sensitive resources. Regarding privacy, the architecture

follows a "process-at-the-edge" principle. Video frames are analyzed in real time without being permanently stored, and no personally identifiable information is retained. Since the anomaly detection and re-identification algorithms rely on abstract features extracted from the images, elements such as faces or license plates are not required for processing. When temporary visual data buffering is unavoidable, e.g., during debugging or system optimization, automatic obfuscation or anonymization techniques (e.g., blurring of identifiable regions) could be applied before any transfer beyond the local node. These measures could ensure compliance with privacy regulations such as the EU General Data Protection Regulation (GDPR) while maintaining system efficiency and responsiveness. Collectively, the combination of decentralized processing, encrypted communications, and optional selective anonymization forms a multi-layered defense model. This model not only mitigates the risk of data leakage but also reinforces user trust and public acceptance, critical factors for the adoption of AI-based monitoring systems in smart city environments.

6. Computing Platform at the Edge

As previously mentioned, the edge nodes rely on a heterogeneous computing platform. Among off-the-shelf devices, one of the most interesting is the AMD Kria K26 SoM [49], which can be integrated into the specialized prototyping board for image and video processing AMD Kria KV260 Vision AI Starter Kit (see Figure 5). The heterogeneous nature of such a SoM facilitates the execution of various types of tasks on the most appropriate computing resources. As depicted in Figure 6, the execution of the AI-based tasks (described in Section 4) considers the hardware acceleration by using the FPGA substrate present in this SoM. On the other hand, the external control of the edge node through container orchestration is handled by the software environment running on Kria's general purpose processor.

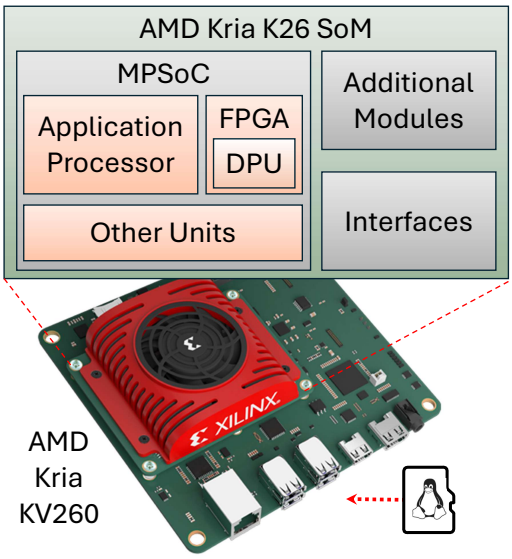

Figure 5. Edge device overview.

6.1. Edge Device Specifications

The AMD Kria K26 SoM embeds a Zynq™ UltraScale+™ MPSoC, an on-SoM memory (i.e., 4 GB 64-bit DDR4 non-ECC and 16 GB eMMC), interfacing components, and additional modules such as a power manager and a security module. The MPSoC includes an application processor (i.e., Quad-core Arm® Cortex®-A53 MPCore™ up to 1.5 GHz), a real-time processor (i.e., Dual-core Arm Cortex-R5F MPCore up to 600 MHz), a Graphics Processing Unit (GPU) (i.e., Mali™-400 MP2 up to 667 MHz), an On-Chip Memory (OCM), a DDR controller, a programmable logic (i.e., FPGA), various standard interfaces (e.g., Eth-

ernet and USB) and internal interconnections. The SoM described above is integrated into the AMD Kria KV260 prototyping board, which makes its interfaces accessible and features a form factor suitable for edge computing, especially for computer vision applications that require configuring the FPGA with the Deep-learning Processing Unit (DPU) provided by the vendor.

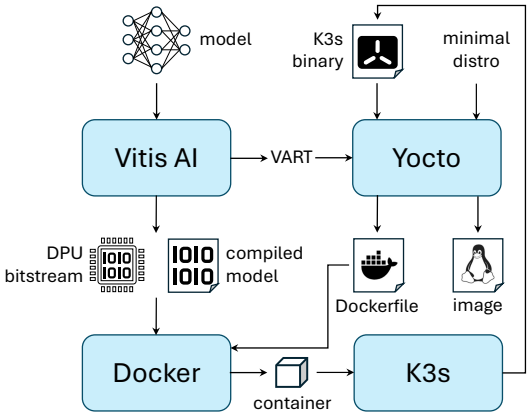

Figure 6. Workflow at the edge.

6.2. Hardware Acceleration Flow

AMD Vitis AI [50] is the tool used in this work to accelerate the execution of the deep learning models underlying the tasks described in Section 4, specifically, YOLOv3 [51] for the AD task and ResNet-50 [52] for the VRI task, which can perform most of the computations in the DINO architecture. Vitis AI provides a comprehensive development environment for deploying AI models on AMD hardware platforms, including the AMD Kria K26 SoM. The toolchain supports model optimization through quantization, compilation into instructions for the DPU, and deployment via a dedicated runtime. In addition to enabling acceleration of custom-trained models, Vitis AI also provides access to a collection of pre-optimized and ready-to-deploy models through the Vitis AI Model Zoo (https://github.com/Xilinx/Vitis-AI/tree/master/model_zoo, accessed on 9 December 2025). The models used in this work are derived from those available in Model Zoo, enabling faster development and integration on the Kria platform.

6.3. Software Environment at the Edge

The software environment used in this work is based on a customized Linux distribution built with the Yocto Project (www.yoctoproject.org/, accessed on 9 December 2025), tailored to meet the specific requirements of edge deployment. In particular, this approach enables (i) a reduced memory footprint of the operating system, which is critical for optimizing the overall resource usage and allows for the creation of lightweight containers for both the AD and VRI tasks; (ii) seamless integration into an edge-to-cloud infrastructure through orchestration with Kubernetes, facilitating scalability, monitoring, and remote management of containerized services; and (iii) full support for hardware acceleration using the Vitis AI design flow. In particular, the inclusion of the Vitis AI Runtime (VART) in the customized distribution ensures efficient execution and management of the DPU tasks, allowing the accelerated models to be deployed and executed directly within the containerized environment on the Kria platform.

7. Evaluation Methodologies and Results

Given the complexity and cost associated with conducting large-scale experimental deployments, simulation offers a practical means to evaluate system behavior under re-

alistic conditions. In the context of this project, the budget did not include the hardware required to perform real-world deployments for validating the proposed methodologies. Consequently, a hybrid strategy was adopted to obtain results that approximate real operating scenarios as closely as possible. Actual power consumption was measured for tasks executed on different hardware platforms—ARM-based edge nodes and GPU-enabled servers—and these measurements were then used to simulate how scalability would affect the system's overall energy usage. The selected hardware configurations are representative of those typically employed in real deployments. This section describes the adopted approach in detail and outlines how it supports bridging the gap between prototype testing and operational environments.

7.1. Identification and Collection of Reference Metrics

The simulation code was developed to model the system's behavior under realistic conditions by capturing the most relevant parameters. Key elements considered include: (i) the network topology, (ii) the computational and energy performance of the nodes for each task, and (iii) the available computational resources on the platform, along with a set of simulated events to evaluate system performance under stress conditions. Accurately measuring real-time energy consumption in computing systems presents several challenges. Tools such as Intel RAPL, available on Intel processors since 2011 and compatible with utilities like 'powerstat' and 'perf', focus primarily on CPU power usage and do not provide a complete assessment of total system energy consumption. To address this limitation, energy evaluation methods generally rely on either direct measurements using physical sensors (electrical or thermal) or predictive models that estimate energy use based on hardware performance counters, memory activity, fan speeds, networks and architectural characteristics [53–55]. For the infrastructure considered in this study, the most effective approach in terms of accuracy and practicality was direct characterization through measurements on representative devices, specifically a Kria board for edge nodes and a dedicated server for the central node. This method enables reliable assessment of energy performance across heterogeneous components under realistic operational conditions. Accordingly, an automated consumption-monitoring system was developed, activated upon the initiation of a task on a device and relying on periodic polling of the smart meter to acquire energy-usage data.

The smart meter selected for this application is the Shelly PM (Mini Gen3 https://www.shelly.com/blogs/documentation/shelly-pm-mini-gen3, accessed on 9 December 2025). This device includes an extensive API interface that enables reliable sensor querying.

We conducted measurements both on the Kria KV260 board and on a workstation equipped with an NVIDIA RTX5060Ti 8GB GPU and an AMD Ryzen5 9600X CPU. A Python script queried the sensor via a dedicated LAN network every 250 ms to receive instantaneous electric power values, as shown in Figure 7. The polling script was fully automated to start and stop in sync with the execution of our test containers for the AD and VRI tasks. Once a test container was launched, the script generated a CSV file adding a new record with the timestamp and the corresponding power value. The polling software was designed to introduce negligible overhead on the server and zero impact on the Kria board. In fact, the system enables remote test execution on the board through a lightweight SSH or serial communication, allowing the test to be started and cleanly terminated at the end of execution without interfering with the device's performance.

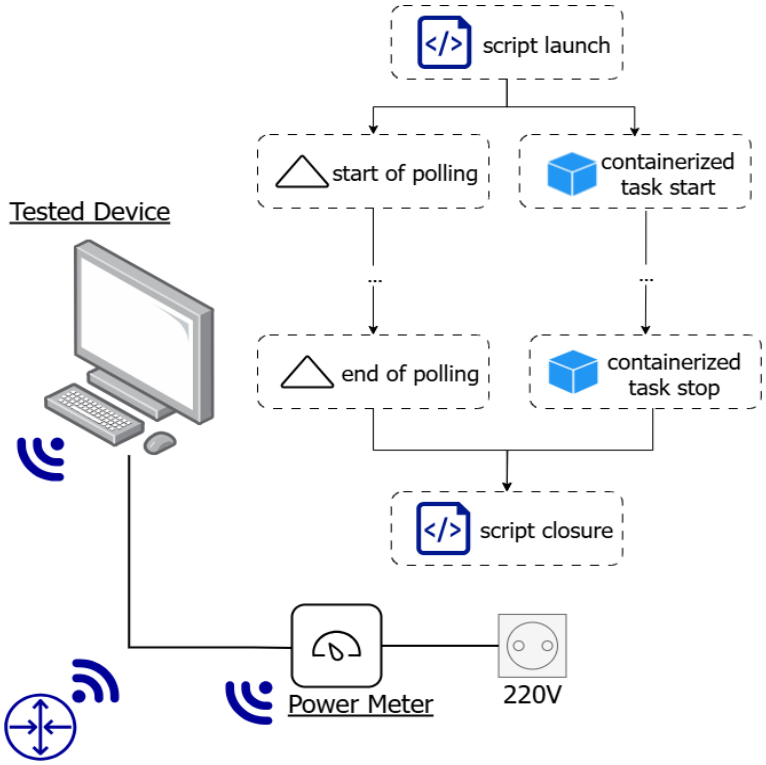

Figure 7. Polling procedure scheme.

The two tests, corresponding to the AD and VRI tasks, were executed 30 and 300 times, respectively. The reliability of the resulting dataset is supported by the statistical indicators computed from our measurements. Specifically, the Coefficient of Variation (CV) and the Relative Standard Error (RSE) for the execution time were both below 2% and 1‰, respectively. These values indicate that the statistical noise in the measurements is minimal, and therefore advanced techniques such as nested sampling or variance component analysis are not required.

To accurately reproduce the timing behavior of the real system, it is essential to model not only the processing phase but also the idle interval that naturally occurs between consecutive acquisition bursts. This is particularly relevant for the AD task, where a fixed interval of 3600 frames separates successive processing windows of 360 frames, and this idle period must be faithfully replicated. Therefore, after the 12-s burst and the subsequent processing, the container enforces a virtual wait time—modeled using `sleep`—calculated as:

$$\text{Wait time} = 3600 \times \left(\frac{12\text{ s}}{360}\right) - T_{\text{processing}}$$

where $T_{\text{processing}}$ is the actual time required for background computation and YOLO inference. This modeling reflects the fact that in a real-world streaming scenario, the data flow from the camera would not pause during processing, meaning the delay between two processing steps accounts for both computation and inter-burst idle time. Therefore, the actual interval between two processing phases is two minutes.

The simulation aims to model various characteristics of the target infrastructure, including network topology, computational performance, and task-related software dependencies. Based on these requirements, the simulation was also designed to replicate network latencies and behavior, particularly under vertical load scaling scenarios—where computation is offloaded from edge nodes to a cloud server. Before implementing the simulation engine itself, a configuration schema was formalized using YAML files to organize the simulation input data.

7.2. Simulation Structure

The simulation was designed modularly to ensure adaptability to system updates and different target infrastructures. Its workflow is schematically illustrated in Figure 8. The simulator models key characteristics of the target environment, including network topology, computational performance, and task-specific software dependencies. It is also designed to replicate network behavior and latencies, particularly under vertical load scaling scenarios, where computation can be offloaded from edge nodes to a cloud server. A configuration schema was formalized using YAML files to organize simulation input data before implementing the simulation engine.

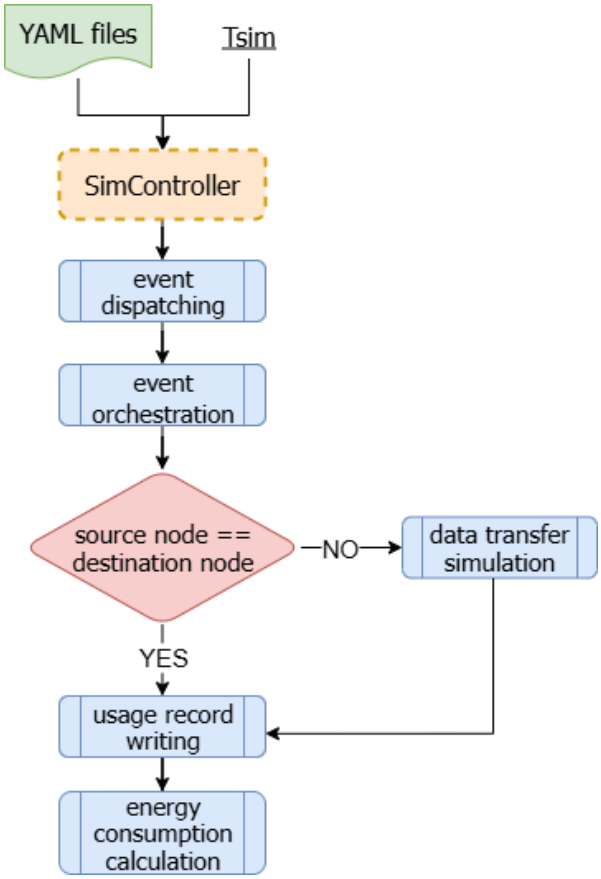

Figure 8. Simulation flow diagram.

The simulator models a distributed computing infrastructure to evaluate task execution and energy consumption. The infrastructure is represented as a network of nodes, each with specific hardware resources and preloaded container images. Tasks are dispatched to appropriate nodes, which may be local or remote. For remote execution, data transfers between nodes are simulated along the shortest path, with transmission time represented in discrete steps. Node activity during both computation and idle periods is tracked to compute total energy consumption, accounting for task-specific hardware usage and execution modes. Tasks can be preemptively interrupted, and such events are logged. The simulation proceeds until a predefined time limit is reached, after which overall performance metrics and energy consumption are calculated.

This lightweight simulator is not intended to emulate the target machines in full detail. Rather, it provides an automated, configurable tool for evaluating infrastructure performance in predefined scenarios.

7.3. Results

The simulations were conducted on hypothetical scenarios involving a variable number of nodes distributed across a grid. The area was divided into sub-areas, each consisting of 6 edge nodes, with one hypothetical workstation node assigned to support the computation within each sub-area. For the purpose of these experiments, the workstation node was assumed to provide additional computational capacity, particularly in the context of VRI tasks. The experiments aimed to assess the scalability and performance of both AD and VRI tasks in large-scale distributed edge-computing scenarios.

The AD tests were simulated with a fixed number of edge nodes executing the computations, both on CPU and on DPU. Specifically, the tests considered scenarios with 120, 1200, and 12,000 nodes, comparing the performance of AD tasks executed solely on the edge nodes, both using CPU processing and leveraging DPU acceleration. The energy consumption for this simulation is illustrated in Figure 9.

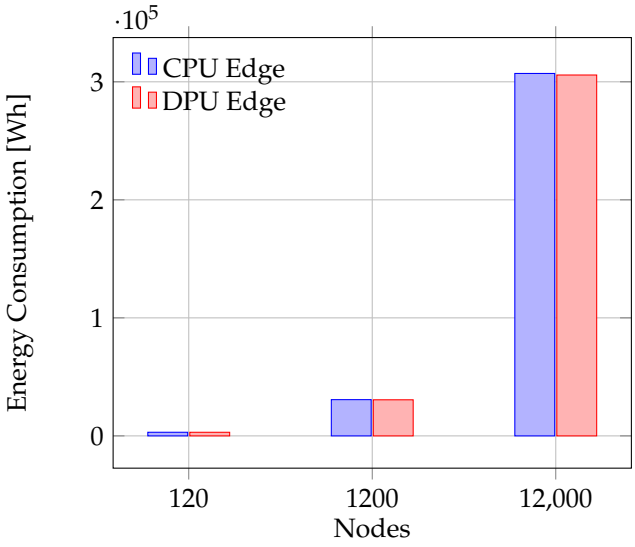

Figure 9. AD simulation results for edge nodes

Simulation results in Figure 9 reveal excellent scalability characteristics. As the system scales from 120 to 12,000 nodes, energy consumption increases linearly rather than exponentially, suggesting that the orchestration approach effectively distributes computational load. At maximum scale, DPU-accelerated execution consumed approximately $3.0 \cdot 10^5$ Wh, compared to $3.1 \cdot 10^5$ Wh for CPU-based processing—a modest 4.8% reduction that becomes increasingly significant at scale. These results suggest that, while individual node-level energy savings are limited, the cumulative effect across large urban deployments can yield meaningful reductions in total energy consumption.

For VRI, a different computational model was considered due to the real-time processing requirements. In this case, the VRI task was assumed to be executed on GPUs, with each sub-area acting as a computational block that uses its local GPU resource to process the video streams. The experiment involved 1200 fixed edge nodes, where a fixed percentage of the nodes (10%, 20%, 30%, 40%, and 50%) were tasked with VRI processing. A further test was conducted in which the proportion of nodes executing VRI dynamically increased over the course of the simulation, starting at 10% at t_0, then moving to 20%, 30%, 40%, and 50% during the simulation, respectively at 1/5, 2/5, 3/5, and 4/5 of the total simulation time. In all tests, where a node was not operating in VRI mode, it was assigned the baseline AD task. The energy hourly consumption for VRI is shown in Figure 10.

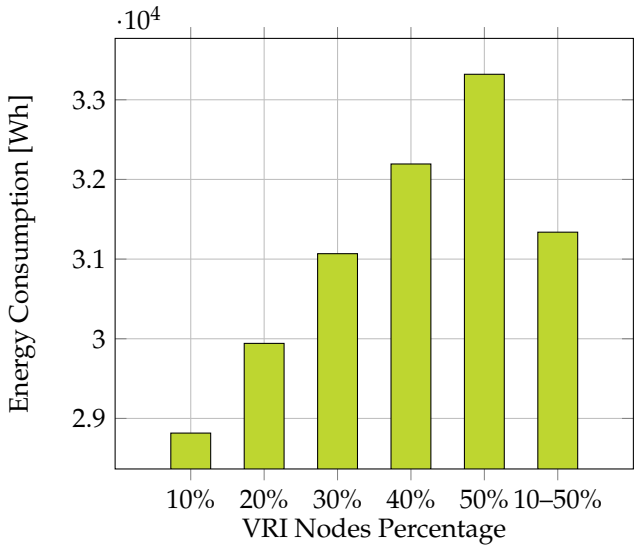

Figure 10. AD+VRI simulation results over 1200 nodes.

The VRI task results in Figure 10 further validate this finding, showing that even when dynamically increasing the percentage of nodes performing VRI tasks from 10% to 50% (represented by the 10–50% column), the system maintains better energy efficiency (approximately $3.15 \cdot 10^4$ Wh) compared to a static 50% deployment (approximately $3.33 \cdot 10^4$ Wh).

The mixed AD+VRI deployment scenarios revealed approximately 10% incremental energy consumption for each 10% increase in nodes performing VRI tasks, rising from $2.88 \cdot 10^4$ Wh at 10% VRI coverage to $3.33 \cdot 10^4$ Wh at 50% coverage. The dynamic scenario, where VRI allocation increased progressively throughout the simulation period, exhibited similar aggregate consumption, validating the orchestrator's ability to maintain energy efficiency during task transitions. These findings support the adoption of edge-heavy architectures for continuous AD operations, with selective cloud offloading reserved for computationally intensive VRI tasks when local resources are insufficient. The measured 8–12× energy advantage of edge processing, combined with the sub-linear scaling properties of the proposed orchestration framework, positions this approach as both technically feasible and operationally sustainable for large-scale smart city deployments.

This energy consumption pattern strongly supports the viability of the proposed heterogeneous edge-to-cloud orchestration approach, particularly in large-scale urban deployments where both operational costs and environmental impact are critical considerations. The data suggests that dynamic task allocation based on real-time demands can provide approximately 6% energy savings compared to static resource allocation strategies in VRI-intensive scenarios.

This can be considered a starting point for smart planning aimed at deploying an adequate number of nodes to cover a specific area of interest, as well as the basis for the correct sizing of the scheduling algorithms in order to balance the number of nodes and the overall energy consumption throughout the monitoring system's lifecycle.

Table 2 reports the energy characterization results for the evaluated nodes. The data correspond to test executions in which model acceleration was performed on the CPU and GPU for the server node, and on the CPU and DPU within the FPGA for the edge node based on the Kria KV260 platform.

The energy characterization data demonstrates a clear efficiency advantage when using hardware acceleration for edge nodes, with DPU-accelerated AD tasks consuming 10.48 Wh compared to 10.59 Wh for CPU-only execution. This represents a modest but meaningful improvement of approximately 1% in energy efficiency at the edge. This near-parity suggests that, for AD workloads, the primary advantage of hardware acceleration

lies in improved throughput and reduced latency rather than substantial energy savings. Conversely, the server node exhibited higher absolute energy consumption (62.0 Wh for CPU, 65.7 Wh for GPU), representing an 6% increase compared to edge execution. This disparity underscores the energy efficiency benefits of edge-based processing for continuous monitoring tasks. For this task, moving from the edge platform to the server leads to an increase in power consumption of approximately a factor of 6, which is attributable to the hardware complexity rather than to the execution of the task itself.

Table 2. Energy characterizations [Wh] for different computing architectures. "–" indicates that the component is not present on the node, while N.A. denotes that the value has not been calculated.

Task	Node	CPU	GPU	DPU
AD	Edge	10.59	–	10.48
	Server	62.0	65.7	–
VRI	Edge	11.75	–	N.A.
	Server	97.7	146.3	–

The VRI task revealed more pronounced differences in the computational landscape. The edge node consumed 11.75 Wh during VRI execution, representing a 10.9% increase over the AD baseline. The server node's GPU accelerated VRI consumed 146.3 Wh, a 49.8% increase over its CPU-based counterpart (97.7 Wh), highlighting the computationally intensive nature of embedding-based re-identification algorithms. The energy differential between edge and cloud GPU execution for VRI demonstrates the substantial energy overhead associated with offloading such tasks to centralized infrastructure. In this case, the comparison between edge and server power consumption shows an even larger increase than in the AD scenario, reaching approximately 9 times the consumption of the edge platform.

The experimental characterization, summarized in Table 3, provides a clear comparison between CPU, GPU, and DPU-based execution times for both AD and VRI tasks.

Table 3. Task's execution time [s] for different computing architectures. "–" indicates that the component is not present on the node, while N.A. denotes that the value has not been calculated.

Task	Node	CPU	GPU	DPU
AD	Edge	134.67	–	133.17
	Server	132.25	132.26	–
VRI	Edge	22.42	–	N.A.
	Server	0.131	0.027	–

For the AD task, at the edge level, the AMD Kria K26 SoM demonstrates a nearly identical execution time between CPU (134.67 s) and DPU (133.17 s) while reducing energy consumption. Although the absolute energy gain (1%) appears modest, it confirms that FPGA acceleration preserves performance while maintaining lower power budgets, which is an essential feature for continuous, real-time edge operations. At the server level, as expected, the GPU exhibits higher power consumption, yet it does not provide any corresponding benefit in terms of execution time.

For the VRI task, execution times vary significantly. The gap between edge and server is substantial: for example, when considering the same architecture (CPU), execution time decreases by roughly two orders of magnitude, from 22.42 s on the edge platform to 0.131 s on the server. Focusing solely on the server, switching to the GPU provides an additional

improvement of about one order of magnitude, reducing execution time from 0.131 s on the CPU to 0.027 s on the GPU. This performance gain, however, comes at the cost of increased power consumption, as shown in Table 2.

Figure 11 shows that, for AD task, hardware-accelerated edge nodes achieve nearly equivalent computational performance to server-class hardware, with an notable reduction in energy consumption. Conversely, for VRI, the complexity of the task demands substantial computational power to achieve acceptable execution times. Figure 11 shows that the three-order-of-magnitude improvement obtained by using the GPU comes at an energy cost of approximately one order of magnitude.

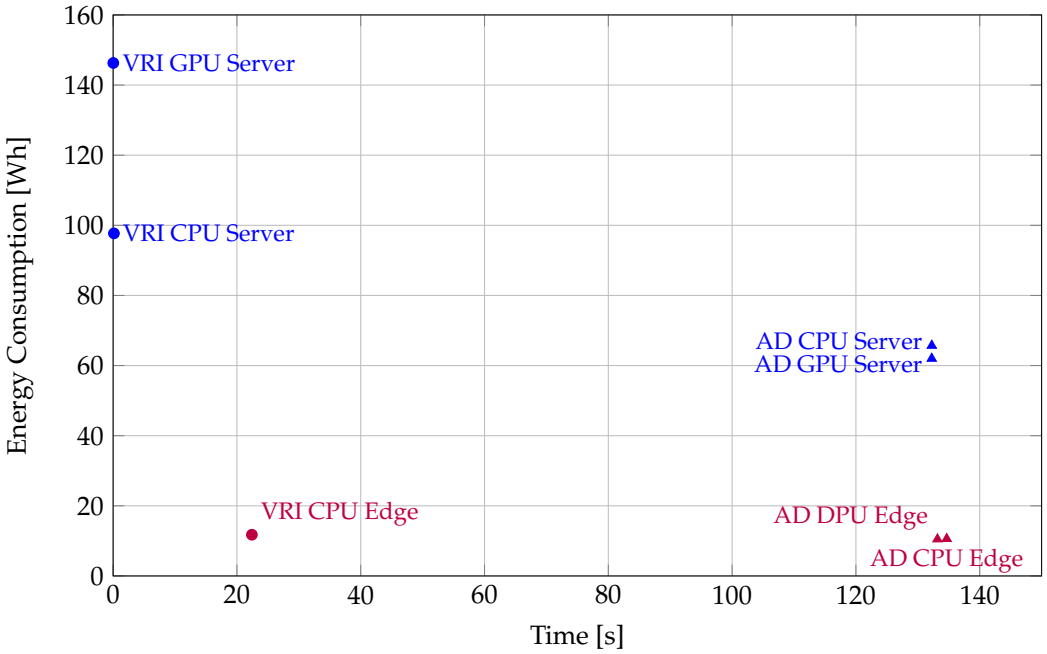

Figure 11. Time-Energy Consumption graph for AD and VRI tasks.

To estimate the total energy consumption under concurrent task execution, a simplified additive model was adopted. In this model, the overall power usage for a set of workloads is approximated as the sum of their individual contributions, measured with respect to the idle baseline of the device. This assumption provides a conservative upper bound for total power, ensuring that combined workloads do not underestimate energy demand.

The rationale behind this approximation lies in physical constraints of embedded hardware. When multiple tasks are executed in parallel, shared resources such as memory buses, voltage regulators, and thermal envelopes, typically cause diminishing marginal power increases. Consequently, the real total consumption is expected to be lower than the sum of the isolated measurements, validating the additive approach as a safe and realistic upper limit.

Adopting this model strikes a balance between experimental feasibility and accuracy. It avoids the exponential complexity of exhaustively measuring every possible workload combination while maintaining sufficiently accurate energy estimates for large-scale simulations. This practical simplification supports the broader goal of the study: to assess scalability and energy sustainability of heterogeneous edge-to-cloud orchestration in realistic urban monitoring deployments.

The consistency between the simulated results and the physical measurements obtained on the AMD Kria K26 SoM reinforces the validity of the adopted energy model. Even with its conservative nature, the model closely approximates real energy trends observed during isolated task execution, confirming that the additive assumption does not

significantly distort the overall energy profile. This consistency provides confidence that the orchestrator's large-scale simulation results (Figures 9 and 10) can be generalized to real-world deployments. From an operational perspective, the energy estimation approach also supports predictive resource planning. By providing upper-bound estimates of energy demand, it enables city administrators or infrastructure providers to pre-allocate energy budgets per monitoring zone and forecast the impact of dynamic task allocation (e.g., when the ratio of AD to VRI nodes varies over time). This is particularly valuable for green smart-city initiatives, where energy-aware orchestration directly translates into reduced CO_2 emissions and improved sustainability metrics.

Finally, although individual per-node energy savings appear modest, their aggregate impact across a dense urban deployment is substantial. A 5–10% improvement in energy efficiency per edge node, scaled to thousands of units continuously operating, translates into measurable reductions in urban data-center load and long-term operational costs. Therefore, even a simple yet conservative estimation framework, as proposed here, contributes significantly to the design of energy-resilient edge-to-cloud ecosystems.

8. Comparative Analysis with Existing Systems and Frameworks

Conventional traffic monitoring solutions rely on centralized video management systems where edge cameras stream video to central servers for processing. Compared to the proposed system, these architectures exhibit higher bandwidth consumption, that is, continuous full-resolution streaming compared to event-triggered transmission, as well as increased latency (i.e., typically 500 ms–2 s versus sub-100 ms for local AD). However, they benefit from mature deployment ecosystems, standardized protocols (ONVIF, RTSP), and lower per-node hardware costs. The proposed system's edge intelligence reduces network load by 80–90%.

Commercial AI traffic platforms typically deploy GPU-based edge servers covering 8–16 cameras per unit, processing video streams for traffic flow analysis, incident detection, and vehicle classification. NoTraffic's system (https://www.notraffic.com/, accessed on 9 December 2025) achieves more than 95% detection accuracy with 200 ms latency using NVIDIA Jetson edge devices. Jetson Xavier NX consumes 10–20 W under load, comparable to the Kria's 10.5 W, but without FPGA flexibility. Commercial systems use proprietary cloud platforms rather than open-source Kubernetes, limiting interoperability but ensuring vendor support.

The proposed system's advantage lies in its open architecture and container orchestration, enabling custom AI models and integration with smart city platforms. However, it lacks the deployment maturity, accuracy validation, and turnkey installation of commercial solutions.

The proposed system's custom scheduler enables domain-specific optimizations (e.g., 300-m radius task assignment based on geographic proximity) that standard Kubernetes schedulers cannot natively handle. However, this customization introduces maintenance burden and deviation from community standards. Alternative approaches using Kubernetes custom resources (CRDs) and operators could achieve similar functionality while remaining within the standard ecosystem.

K3s (https://k3s.io/, accessed on 9 December 2025) is explicitly designed for edge computing, with 40 MB binary size vs. standard Kubernetes' 1 GB+ distribution. Compared to the paper's implementation K3s control plane uses 512 MB RAM vs. estimated 2–4 GB for the proposed Master Node Edge suitability, supports ARM64 natively and includes SQLite backend option, compared to MongoDB dependency in the paper. K3s lacks built-in geographic awareness and custom scheduling logic for vehicle tracking.

The paper could have achieved 50–60% resource reduction by adopting K3s as the base orchestration layer and implementing geographic scheduling as a K3s-compatible scheduler extender. The choice of full Kubernetes suggests the authors prioritized Azure Kubernetes Service compatibility over edge-optimized design.

KubeEdge (https://kubeedge.io/, accessed on 9 December 2025) extends Kubernetes specifically for edge scenarios with cloud-edge coordination, offline autonomy, and resource-constrained device support. KubeEdge nodes continue operating during cloud disconnection, while the proposed system depends on Azure IoT Hub. Moreover, KubeEdge includes device twin abstractions for IoT integration. The proposed system implements custom camera node management Maturity.

OpenFaaS (https://www.openfaas.com/, accessed on 9 December 2025) enables serverless function deployment on Kubernetes, optimizing for stateless, event-driven workloads. OpenFaaS functions are stateless and short-lived (seconds to minutes); AD/VRI tasks require continuous video processing with maintained state (background models, vehicle embeddings). Function-based execution could reduce idle resource consumption but would increase latency for time-sensitive tasks. OpenFaaS is fundamentally unsuited for continuous video stream processing. The proposed system's container-based approach is appropriate, though a hybrid model, persistent containers for AD, serverless functions for report processing and Master Node orchestration logic could optimize resource utilization. Table 4 summarizes this comparative analysis. Future enhancements will consider integrating Digital Twin mechanisms, such as the DT4S conceptual strategy [56], to represent virtual replicas of physical nodes and urban mobility contexts. This approach could improve lifecycle management, semantic interoperability, and predictive maintenance within the orchestrated environment.

Table 4. Comparison with Existing Systems.

Feature	Proposed System	Standard K8s	K3s	KubeEdge	OpenFaaS
Edge Autonomy	Low	Medium	Medium	Excellent	Medium
Energy Efficiency	High (measured)	Low	High	High	Medium
AI Support	Excellent	Good	Good	Good	Limited
Deployment Maturity	Low (prototype)	Excellent	Good	Medium	Good
Geographic Awareness	Excellent (custom)	Poor	Poor	Medium	Poor

9. Conclusions

This study presented an advanced edge-to-cloud orchestration framework specifically developed for real-time monitoring of urban traffic using heterogeneous computing architectures. By combining general-purpose processors with FPGA-accelerated computing on AMD Kria K26 SoMs, the system achieves high computational performance while maintaining low energy consumption, essential requirement for smart city applications.

The orchestration platform is based on Kubernetes supporting efficient deployment and scaling of AI-based microservices, handling tasks such as AD and vehicle re-identification. It intelligently distributes workloads between edge devices and the cloud to ensure continuous operation and rapid responsiveness. The AD component uses deep learning techniques to detect unusual traffic events, while the vehicle re-identification module employs a customized Vision Transformer model (DINOv2) to accurately track vehicles across multiple camera sources.

Comprehensive tests, including simulated scenarios for energy usage and task execution, demonstrate that the system satisfies real-time demands while optimizing resource allocation. Edge execution offers up to 8–12× lower energy consumption compared to

equivalent cloud-based processing. Hardware acceleration through the DPU enables stable, low-power inference without performance degradation. Dynamic orchestration between AD and VRI tasks ensures balanced resource utilization and adaptive load management.

The analysis of obtained results clearly quantifies these benefits, showing that the marginal energy gains per node translate into substantial savings at city scale, thus supporting environmentally sustainable smart-city infrastructures.

Future work may focus on enhancing the orchestration logic with predictive models for service migration and exploring federated learning approaches to further improve adaptability in evolving urban scenarios. The additive power estimation model adopted in this work, while theoretically sound under submodularity assumptions, may overestimate actual energy consumption during concurrent task execution. Real-world scenarios where AD and VRI tasks run simultaneously could benefit from shared resource utilization (e.g., common memory access patterns, shared preprocessing pipelines) that would violate the strict additivity assumption. Future work should validate these estimates through direct measurement of multi-task workloads to quantify the magnitude of this conservative bias.

Author Contributions: Conceptualization, P.R., A.L., C.R. and M.F.; Methodology, P.R., A.L., C.R. and M.F.; Software, P.R., A.L., C.R., M.A. and M.S.; Validation, P.R.; Formal analysis, P.R., A.L., C.R., M.A., M.S. and M.F.; Investigation, P.R., A.L., C.R., M.S. and M.F.; Writing—original draft, P.R., A.L., C.R., M.A., M.S. and M.F.; Writing—review & editing, P.R., A.L., M.A. and M.F.; Supervision, M.F.; Project administration, M.F.; Funding acquisition, P.R., C.R. and M.F. All authors have read and agreed to the published version of the manuscript.

Funding: The research activities described in this paper have been conducted within the R&D project "ONE-Orchestrator for distributed edge-to-cloud systems based on Heterogeneous Nodes in urban environment" CUP C89J4000260004 funded by "RESearch and innovation on future Telecommunications systems and networks, to make Italy more smART" program, funded by EU–NextGenerationEU-Cascading call for state universities and EPRs supervised by the MUR to draw on the funds CUP E13C22001870001 Spoke 4.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

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

References

1. Al-Dulaimy, A.; Jansen, M.; Johansson, B.; Trivedi, A.; Iosup, A.; Ashjaei, M.; Galletta, A.; Kimovski, D.; Prodan, R.; Tserpes, K.; et al. The computing continuum: From IoT to the cloud. *Internet Things* **2024**, *27*, 101272. [CrossRef]
2. Farhoudi, M.; Shokrnezhad, M.; Taleb, T.; Li, R.; Song, J. Discovery of 6G Services and Resources in Edge-Cloud-Continuum. *IEEE Netw.* **2025**, *39*, 223–232. [CrossRef]
3. Ruiu, P.; Scionti, A.; Nider, J.; Rapoport, M. Workload management for power efficiency in heterogeneous data centers. In Proceedings of the 2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), Fukuoka, Japan, 6–8 July 2016; IEEE: New York, NY, USA, 2016; pp. 23–30.
4. Fu, K.; Zhang, W.; Chen, Q.; Zeng, D.; Guo, M. Adaptive Resource Efficient Microservice Deployment in Cloud-Edge Continuum. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1825–1840. [CrossRef]
5. Belcastro, L.; Marozzo, F.; Orsino, A.; Talia, D.; Trunfio, P. Navigating the Edge-Cloud Continuum: A State-of-Practice Survey. *arXiv* **2025**, arXiv:2506.02003.
6. Chiang, Y.; Zhang, Y.; Luo, H.; Chen, T.Y.; Chen, G.H.; Chen, H.T.; Wang, Y.J.; Wei, H.Y.; Chou, C.T. Management and Orchestration of Edge Computing for IoT: A Comprehensive Survey. *IEEE Internet Things J.* **2023**, *10*, 14307–14331. [CrossRef]
7. Scionti, A.; Ruiu, P.; Terzo, O.; Nider, J.; Petrie, C.; Baldoni, N. Opera: A low power approach to the next generation cloud infrastructures. In Proceedings of the 2016 Euromicro Conference on Digital System Design (DSD), Limassol, Cyprus, 31 August–2 September 2016; IEEE: New York, NY, USA, 2016; pp. 326–333.
8. Boettiger, C. An introduction to Docker for reproducible research. *ACM SIGOPS Oper. Syst. Rev.* **2015**, *49*, 71–79. [CrossRef]
9. Niazi, M.; Abbas, S.; Soliman, A.H.; Alyas, T.; Asif, S.; Faiz, T. Vertical Pod Autoscaling in Kubernetes for Elastic Container Collaborative Framework. *Comput. Mater. Contin.* **2022**, *74*, 591–606.

10. Liu, X.; Yang, J.; Zou, C.; Chen, Q.; Yan, X.; Chen, Y.; Cai, C. Collaborative Edge Computing With FPGA-Based CNN Accelerators for Energy-Efficient and Time-Aware Face Tracking System. *IEEE Trans. Comput. Soc. Syst.* **2022**, *9*, 252–266. [CrossRef]
11. Gkonis, P.; Giannopoulos, A.; Trakadas, P.; Masip-Bruin, X.; D'Andria, F. A survey on IoT-edge-cloud continuum systems: Status, challenges, use cases, and open issues. *Future Internet* **2023**, *15*, 383. [CrossRef]
12. Khalyeyev, D.; Bureš, T.; Hnětynka, P. Towards characterization of edge-cloud continuum. In Proceedings of the European Conference on Software Architecture, Prague, Czech Republic, 19–23 September 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 215–230.
13. Samie, F.; Bauer, L.; Henkel, J. From Cloud Down to Things: An Overview of Machine Learning in Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4921–4934. [CrossRef]
14. Rosendo, D.; Costan, A.; Valduriez, P.; Antoniu, G. Distributed intelligence on the edge-to-cloud continuum: A systematic literature review. *J. Parallel Distrib. Comput.* **2022**, *166*, 71–94. [CrossRef]
15. Guo, W.; Zhuang, F.; Zhang, X.; Tong, Y.; Dong, J. A comprehensive survey of federated transfer learning: Challenges, methods and applications. *Front. Comput. Sci.* **2024**, *18*, 186356. [CrossRef]
16. Wang, L.; Ren, X.; Zhao, C.; Zhao, F.; Yang, S. MPDM: A Multi-Paradigm Deployment Model for Large-Scale Edge-Cloud Intelligence. *IEEE Internet Things J.* **2023**, *10*, 8773–8785. [CrossRef]
17. Caballer, M.; Moltó, G.; Calatrava, A.; Blanquer, I. Infrastructure manager: A tosca-based orchestrator for the computing continuum. *J. Grid Comput.* **2023**, *21*, 51. [CrossRef]
18. Kontodimas, K.; Soumplis, P.; Kretsis, A.; Kokkinos, P.; Fehér, M.; Lucani, D.E.; Varvarigos, E. Secure Distributed Storage Orchestration on Heterogeneous Cloud-Edge Infrastructures. *IEEE Trans. Cloud Comput.* **2023**, *11*, 3407–3425. [CrossRef]
19. Sahu, D.; Mandal, S.; Dey, S. Optimizing Energy and Latency in Edge Computing through Probabilistic Scheduling. *Sci. Rep.* **2025**, *15*, 30452. [CrossRef]
20. Sabbadin, A.; Guérout, T. Towards a Classification of Edge Service Orchestration Strategies Integrating Renewable Energy. In Proceedings of the International Conference on Energy-Aware Computing; Tunis, Tunisia, 3–6 December 2024, Springer: Berlin/Heidelberg, Germany, 2025; pp. 305–317. [CrossRef]
21. Robles-Enciso, A.; Skarmeta, A.F. Adapting Containerized Workloads for the Continuum Computing. *IEEE Access* **2024**, *12*, 104102–104114. [CrossRef]
22. Rac, S.; Brorsson, M. Cost-aware service placement and scheduling in the edge-cloud continuum. *ACM Trans. Archit. Code Optim.* **2024**, *21*, 1–24. [CrossRef]
23. Sofia, R.C.; Salomon, J.; Ferlin-Reiter, S.; Garcés-Erice, L.; Urbanetz, P.; Mueller, H.; Touma, R.; Espinosa, A.; Contreras, L.M.; Theodorou, V.; et al. A Framework for Cognitive, Decentralized Container Orchestration. *IEEE Access* **2024**, *12*, 79978–80008. [CrossRef]
24. Di Cicco, N.; Poltronieri, F.; Santos, J.; Zaccarini, M.; Tortonesi, M.; Stefanelli, C.; De Turck, F. Multi-Objective Scheduling and Resource Allocation of Kubernetes Replicas Across the Compute Continuum. In Proceedings of the 2024 20th International Conference on Network and Service Management (CNSM), Prague, Czech Republic, 28–31 October 2024; pp. 1–9. [CrossRef]
25. Bellavista, P.; Di Modica, G. IoTwins: Implementing distributed and hybrid digital twins in industrial manufacturing and facility management settings. *Future Internet* **2024**, *16*, 65. [CrossRef]
26. Prangon, N.F.; Wu, J. AI and Computing Horizons: Cloud and Edge in the New Era. *J. Sens. Actuator Netw.* **2024**, *13*, 44. [CrossRef]
27. Kennedy, J.; Sharma, V.; Varghese, B.; Reaño, C. Multi-Tier GPU Virtualization for Deep Learning in Cloud-Edge Systems. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 2107–2123. [CrossRef]
28. De Ulzurrun, I.D.; Encinas, J.; Rodríguez, A.; Otero, A. Cloud-Edge Continuum Infrastructure for Reconfigurable Multi-Accelerator Systems. In Proceedings of the 2024 39th Conference on Design of Circuits and Integrated Systems (DCIS), Catania, Italy, 13–15 November 2024; pp. 1–6. [CrossRef]
29. Rodríguez, A.; Valverde, J.; Portilla, J.; Otero, A.; Riesgo, T.; De la Torre, E. Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework. *Sensors* **2018**, *18*, 1877. [CrossRef] [PubMed]
30. Truong, H.L.; Magoutis, K. Robustness via Elasticity Accelerators for the IoT-Edge-Cloud Continuum. In Proceedings of the 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), Vancouver, WA, USA, 6–9 December 2022; pp. 291–296. [CrossRef]
31. Nanos, A.; Kretsis, A.; Mainas, C.; Ntouskos, G.; Ferikoglou, A.; Danopoulos, D.; Kokkinis, A.; Masouros, D.; Siozios, K.; Soumplis, P.; et al. Hardware-Accelerated FaaS for the Edge-Cloud Continuum. In Proceedings of the 2023 IEEE 31st International Conference on Network Protocols (ICNP), Reykjavik, Iceland, 10–13 October 2023; pp. 1–6. [CrossRef]
32. Leftheriotis, A.; Tzenetopoulos, A.; Lentaris, G.; Soudris, D.; Theodoridis, G. TF2AIF: Facilitating development and deployment of accelerated AI models on the cloud-edge continuum. In Proceedings of the 2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Antwerp, Belgium, 3–6 June 2024; pp. 931–936. [CrossRef]

33. Zhang, Z.; Lin, Y.; Wang, C. Seamless HW-Accelerated AI Serving in Heterogeneous MEC Clusters. In Proceedings of the 2024 ACM Workshop on Edge Systems, Analytics and Networking, Athens, Greece, 22 April 2024; ACM: New York, NY, USA, 2024. [CrossRef]
34. Nezami, Z.; Chaniotakis, E.; Pournaras, E. When Computing Follows Vehicles: Decentralized Mobility-Aware Resource Allocation for Edge-to-Cloud Continuum. *IEEE Internet Things J.* **2025**, *12*, 23324–23340. [CrossRef]
35. Pereira, P.; Melo, C.; Araujo, J.; Dantas, J.; Santos, V.; Maciel, P. Availability model for edge-fog-cloud continuum: An evaluation of an end-to-end infrastructure of intelligent traffic management service. *J. Supercomput.* **2022**, *78*, 4421–4448. [CrossRef]
36. Belcastro, L.; Marozzo, F.; Orsino, A.; Talia, D.; Trunfio, P. Edge-Cloud Continuum Solutions for Urban Mobility Prediction and Planning. *IEEE Access* **2023**, *11*, 38864–38874. [CrossRef]
37. Rosmaninho, R.; Raposo, D.; Rito, P.; Sargento, S. Edge-Cloud Continuum Orchestration of Critical Services: A Smart-City Approach. *IEEE Trans. Serv. Comput.* **2025**, *18*, 1381–1396. [CrossRef]
38. Chavhan, S.; Deepika, I.S.; Gupta, D.; Rodrigues, J.J.P.C. Energy-Efficient-Enabled Edge-AI-IoT Integrated Traffic Incident Analysis and Avoidance of Secondary Incidents. *IEEE Internet Things J.* **2025**, *12*, 34720–34730. [CrossRef]
39. Poulton, N. *The Kubernetes Book*; NIGEL POULTON LTD.: Cheshire, UK, 2023.
40. Wilder, B. *Cloud Architecture Patterns: Using Microsoft Azure*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2012.
41. Tosatto, A.; Ruiu, P.; Attanasio, A. Container-based orchestration in cloud: State of the art and challenges. In Proceedings of the 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, Santa Catarina, Brazil, 8–10 July 2015; IEEE: New York, NY, USA, 2015; pp. 70–75.
42. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
43. Aboah, A. A vision-based system for traffic anomaly detection using deep learning and decision trees. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 4207–4212.
44. Oquab, M.; Darcet, T.; Moutakanni, T.; Vo, H.V.; Szafraniec, M.; Khalidov, V.; Fernandez, P.; Haziza, D.; Massa, F.; El-Nouby, A.; et al. DINOv2: Learning Robust Visual Features without Supervision. *arXiv* **2023**, arXiv:2304.07193.
45. Miao, J.; Wei, Y.; Wu, Y.; Liang, C.; Li, G.; Yang, Y. Vspw: A large-scale dataset for video scene parsing in the wild. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 4133–4143.
46. Delussu, R.; Putzu, L.; Fumera, G. Synthetic Data for Video Surveillance Applications of Computer Vision: A Review. *Int. J. Comput. Vis.* **2024**, *132*, 4473–4509. [CrossRef]
47. Chauhan, A. A review on various aspects of MongoDB databases. *Int. J. Eng. Res. Technol. (IJERT)* **2019**, *8*, 90–92.
48. Harshjeet.; Amudha, S.; Gogoi, C.; Snehalatha, N. Enhancing Visual Creativity with Neural Style Transfer using Celery Backend Architecture. In Proceedings of the 2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 5–7 June 2024; IEEE: New York, NY, USA, 2024; pp. 966–974. [CrossRef]
49. AMD Xilinx. Kria K26 SOM: The Ideal Platform for Vision AI at the Edge. 2021. Available online: https://docs.amd.com/v/u/en-US/wp529-som-benchmarks (accessed on 9 December 2025).
50. Xilinx, A. AMD Vitis AI Software-Enabling Real-Time Inference Acceleration on AMD Adaptive SoCs, FPGAs and Acceleration Cards. 2025. Available online: https://www.amd.com/en/products/software/vitis-ai.html (accessed on 9 December 2025).
51. Farhadi, A.; Redmon, J. Yolov3: An incremental improvement. In Proceedings of the Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; Springer: Berlin/Heidelberg, Germany, 2018; Volume 1804, pp. 1–6.
52. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778. [CrossRef]
53. Li, T.; John, L. Run-time Modeling and Estimation of Operating System Power Consumption. *ACM SIGMETRICS Perform. Eval. Rev.* **2003**, *31*, 160–171. [CrossRef]
54. Ruiu, P.; Bianco, A.; Fiandrino, C.; Giaccone, P.; Kliazovich, D. Power comparison of cloud data center architectures. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 23–27 May 2016; IEEE: New York, NY, USA, 2016; pp. 1–6.
55. Ruiu, P.; Fiandrino, C.; Giaccone, P.; Bianco, A.; Kliazovich, D.; Bouvry, P. On the energy-proportionality of data center networks. *IEEE Trans. Sustain. Comput.* **2017**, *2*, 197–210. [CrossRef]
56. Rabah, S.; Zacharewicz, G.; Chapurlat, V. Digital twin for services (DT4S): Conceptual strategy. *IFAC-PapersOnLine* **2022**, *55*, 3256–3261. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.