A Cognitive Serverless Framework for the Cloud-Edge Continuum

D5.6 Use Cases - Scientific Report - f

Version 2.0

1 March 2026

Abstract

COGNIT is an AI-Enabled Adaptive Serverless Framework for the Cognitive Cloud-Edge Continuum that enables the seamless, transparent, and trustworthy integration of data processing resources from providers and on-premises data centers in the cloud-edge continuum, and their automatic and intelligent adaptation to optimise where and how data is processed according to application requirements, changes in application demands and behaviour, and the operation of the infrastructure in terms of the main environmental sustainability metrics. This standalone document provides the overall vision of the scientific work carried out in WP5 during the project, being the final delivery of this WP. It describes the contribution of the Project's software requirements towards meeting the user requirements that guide the development of the COGNIT Framework, offers additional information about the domains targeted by the Use Cases and the Partners involved in them, and explains the Project's software integration process and infrastructure on its testbed environment. Additionally, it explains the software requirement verification tasks carried out under this WP.

This project is funded by the European Union's Horizon Europe research and innovation programme under Grant Agreement 101092711 – SovereignEdge.Cognit

Deliverable Metadata

Project Title:	A Cognitive Serverless Framework for the Cloud-Edge Continuum
Project Acronym:	SovereignEdge.Cognit
Call:	HORIZON-CL4-2022-DATA-01-02
Grant Agreement:	101092711
WP number and Title:	WP5. Adaptive Serverless Framework Integration and Validation
Nature:	R: Report
Dissemination Level:	PU: Public
Version:	2.0
Contractual Date of Delivery:	30/09/2025
Actual Date of Delivery:	04/03/2026
Lead Author:	Thomas Ohlson Timoudas & Joel Höglund (RISE)
Authors:	Monowar Bhuyan (UMU), Malik Bouhou (CETIC), Aritz Brosa (Ikerlan), Idoia de la Iglesia (Ikerlan), Agnieszka Frąc (Atende), Torsten Hallmann (SUSE), Joel Höglund (RISE), Carlos Lopez (ACISA), Mateusz Kobak (Phoenix), Tomasz Korniluk (Phoenix), Antonio Lalaguna (ACISA), Fátima Fernández (Ikerlan), Carlos Lopez (ACISA), Philippe Massonet (CETIC), Nikolaos Matskanis (CETIC), Xavier Lessage (CETIC), Jean Lazarou (CETIC), Daniel Olsson (RISE), Mikel Irazola (Ikerlan), Álvaro Puente (Ikerlan), Holger Pfister (SUSE), Francesco Renzi (Nature 4.0), Marco Mancini (OpenNebula), Juan José Ruiz (ACISA), Kaja Swat (Phoenix), Paul Townend (UMU), Iván Valdés (Ikerlan), Yashwant Singh Patel (UMU), Riccardo Valentini (Nature 4.0), Pavel Czerny (OpenNebula), Bruno Rodríguez (OpenNebula), Filippo Tagliacarne (Nature 4.0).
Status:	Submitted

Document History

Version	Issue Date	Status[1]	Content and changes
0.1	02/12/2025	Draft	Initial Draft
0.2	22/12/2025	Peer-reviewed	Reviewed Version
1.0	31/12/2025	Submitted	Final Version
1.1	01/02/2026	Draft	Initial Draft
1.2	28/02/2026	Peer-reviewed	Reviewed Version
2.0	04/03/2026	Submitted	Final Version

Peer Review History

Version	Peer Review Date	Reviewed By
0.2	29/12/2025	Antonio Álvarez (OpenNebula) & Marco Mancini (OpenNebula)
1.2	23/02/2026	Antonio Álvarez (OpenNebula) & Marco Mancini (OpenNebula)
1.2	02/02/2026	Torsten Hallmann (SUSE)
1.2	23/02/2026	Antonio Álvarez (OpenNebula)
1.2	28/02/2026	Marco Mancini (OpenNebula)

Summary of Changes from Previous Versions

First Version of Deliverable D5.6

[1] A deliverable can be in one of these stages: Draft, Peer-Reviewed, Submitted, and Approved.

Executive Summary

Deliverable D5.6, released at the end of the final Research & Innovation Cycle, is the final version of the Use Cases Scientific Report in WP5 ”Adaptive Serverless Framework Integration and Validation”.

This deliverable provides an overview of the work done in WP5 during the project for each of the four use cases. The document reports the implementation, tests and deployments done by the Use Case partners. Additionally, it offers updates on the software integration process, infrastructure, testbed environment, and verification of software requirements.

The work reported in this document is complemented by the detailed project software resources reported in Deliverable D5.9, and the use cases’ project demo description presented in Deliverable D5.12. In turn, the document is complementary to the global overview provided by Deliverable D2.6, and the final versions of the component-specific deliverables from WP3 (i.e. Deliverable D3.5) and WP4 (i.e. Deliverable D4.5).

Table of Contents

Abbreviations and Acronyms

3GPP	**3rd Generation Partnership Project**
5G	Fifth Generation Mobile Network
AD	Anomaly Detection
AI	Artificial Intelligence
API	Application Programming Interface
ATMS	Advanced Traffic Management System
AUROC	Area under the receiver operating characteristic
AWS	Amazon Web Services
BDD	Behavioural Driven Development
BESS	Battery Energy Storage System
CCAM	Cooperative, Connected and Automated Mobility
CC-CV	Current and Constant Voltage
CNN	Convolutional Neural Network
C-ITS	Cooperative Intelligent Transport System
C-V2X	Cellular Vehicle to Everything communication technology
DaaS	Data as a Service
DCC	Device Client in C
DDPG	Deep Deterministic Policy Gradient
DSRC	Dedicated Short Range Communications
DSO	Distribution System Operator
EC2	(Amazon) Elastic Compute Cloud
EEA	European Environment Agency
EM	Embedded Model
EN	European Norms
ENS	Esquema Nacional de Seguridad (Spanish National Security Schema)
ETSI	European Telecommunications Standards Institute
EV	Electric Vehicle
FaaS	Function as a Service
FAQ	Frequently Asked Questions
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
HEMS	Home Energy Management System

HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation and Air Conditioning
HMI	Human Machine Interface
ICE	Infrastructure and Cloud Research & Test Environment at RISE
ID	Identifier
IP	Internet Protocol
IPv6	Internet Protocol version 6
ITS	Intelligent Transport System
KIWI	Linux System Appliance Builder
KVM	Kernel-based Virtual Machine
kW	kiloWatt(s)
LPWAN	Low Power Wide Area Network
LTE	Long-Term Evolution
MaaS	Mobility as a Service
MAPEM	MAP (Topology) Extended Message
MDP	Markov Decision Process
ML	Machine Learning
M-Hub	Mobility Hub (advanced TLC)
MQTT	A lightweight, pub-sub, M2M network protocol for message queue service
NB-IoT	NarrowBand - Internet of Things
OBU	On Board Unit
OCPP	Open Charge Point Protocol
OS	Operating System
PCI	Peripheral Component Interconnect
PPO	Proximal Policy Optimisation
PTP	Public Transport Prioritisation
PV	Photovoltaic
QA	Quality Assurance
QoS	Quality of Service
RES	Renewable Energy Source
REST	Representational State Transfer
RL	Reinforcement Learning
RSU	Road Side Unit
RT	Response Time
RTOS	Real-Time Operating System

SAE	Society of Automotive Engineers
SEM	Smart Energy Meter
SPATEM	Signal Phase And Timing Extended Message
SR	Serverless Runtime
SREM	Signal Request Extended Message
SSEM	Signal Request Status Extended Message
SSH	Secure Shell
SSL	Secure Sockets Layer
SUMO	Simulation of Urban Mobility[2]
TCC	Traffic Control Center
TCP	Transmission Control Protocol
TDOA	Time Difference Of Arrival
TEE	Trusted Execution Environment
TLC	Traffic Light Controller
TOA	Time Of Arrival
TS	Technology Specifications
TSP	Traffic/Transit Signal Priority
TTC	TreeTalker Cyber
UART	Universal Asynchronous Receiver-Transmitter
V2X	Vehicle to Everything communication technology
VM	Virtual Machine
VPN	Virtual Private Network
Wh	Watthour(s)
WHO	World Health Organization

[2] An open source, highly portable, microscopic and continuous multi-modal traffic simulation package designed to handle large networks: https://eclipse.dev/sumo/

1. Introduction

This document represents the sixth and final version of the Use Cases Scientific Report, summarizing the outcomes of the project's efforts to advance the cloud-edge continuum through the validation of four representative use cases: Smart Cities, Wildfire Detection, Energy, and Cybersecurity. The deliverable reports on the work performed throughout the entire project lifecycle, providing use-case centred view of the achievements, integration activities, and lessons learned.

The report builds upon the initial objectives defined at the start of the project:

- To design and implement a flexible architecture enabling seamless interaction between cloud and edge resources.
- To validate this architecture through real-world scenarios that demonstrate its scalability, security, and performance benefits.
- To deliver a software stack and orchestration framework capable of supporting heterogeneous infrastructures and resource-constrained environments.

Over the course of the project, the COGNIT Framework has evolved significantly, driven by iterative feedback from the use cases and the requirements identified during successive research and innovation cycles. This evolution included the introduction of function offloading for hardware-constrained devices, enhanced orchestration capabilities through AI-enabled resource management, and automated deployment mechanisms via OpsForge. These advancements were validated through testing on both shared and local testbeds, leading up to the final demonstrations on integrated infrastructures.

The deliverable is organized into two main parts:

- Part I provides a detailed account of the validation use cases, including their reference scenarios, objectives, activities, infrastructure setup, and technology readiness outlook.
- Part II focuses on the software integration process, the evolution of the shared COGNIT testbed, and the verification of software and use case requirements.

The document concludes with a brief synthesis of the project's findings, highlighting the impact of the validated architecture and the improved technology readiness of the developed technologies.

This final report is intended to serve as a stand-alone document, capturing the relevant use case activities from initial design to final validation and offering insights into how the cloud-edge continuum can enable next-generation digital services across multiple domains.

PART I. Validation Use Cases

2. Overall Status

The table below shows the current status of each Software Requirement towards meeting its associated global and user requirements, following a simple colour code: for activities that have not finally been achieved and for completed activities:

	ID	DESCRIPTION	Device Client						COGNIT Frontend	Edge Cluster		Cloud-Edge Manager						AI-Enabled Orchestrator		Secure & Trusted Execution of Computing Environments		
			SR1.1	SR1.2	SR1.3	SR1.4	SR1.5	SR1.6	SR2.1	SR3.1	SR3.2	SR4.1	SR4.2	SR4.3	SR4.4	SR4.5	SR4.6	SR5.1	SR5.2	SR6.1	SR6.2	SR6.3
Sovereignty	SOR0.1	The COGNIT Framework shall be able to leverage public, private, and self-hosted cloud and edge infrastructures hosted in the European Union.																				
	SOR0.2	The implementation of the COGNIT Architecture shall maximise the use of European open source technologies and frameworks.																				
	SOR0.3	The COGNIT Framework shall provide an abstraction layer that ensures workload portability seamlessly across different infrastructure providers.																				
	SOR0.4	Data handling by the COGNIT Framework shall be compliant with the GDPR.																				
Sustainability	SUR0.1	Sustainability performance needs to be measurable (e.g. energy profiles should be queryable and updatable for every feature/component within the framework), including energy sources (e.g. renewable, non-renewable) and energy consumption profiles (e.g. estimated power consumption).																				
	SUR0.2	Sustainability needs to be maximised to reduce environmental footprint by leveraging edge characteristics (e.g. by increasing the share of renewables, minimising battery use/size, using energy otherwise wasted, or scaling down active Runtimes).																				
	SUR0.3	The whole energy lifecycle should be taken into account in order to implement a circular economy, including e.g. energy availability and cost and hardware degradation.																				
Interoperability	IR0.1	Deployment of the COGNIT Framework and of its components should be as portable as possible across heterogeneous infrastructures or cloud/edge service providers (e.g. by using broadly-adopted virtualisation and container technologies).																				
	IR0.2	Preference should be given to expanding existing frameworks, tools, and open standards.																				
	IR0.3	The interfaces of the COGNIT Framework shall be documented in order to facilitate discovery of its features by third-parties.																				
Security	SER0.1	Communications inside COGNIT, and between the COGNIT environment and the outside (e.g. IoT devices) should be encrypted and signed using security mechanisms such as SSLv3.																				
	SER0.2	The COGNIT Framework should be built following security-by-design and Zero Trust practices.																				
	SER0.3	The implementation of the COGNIT Framework should be aligned with the latest legislative frameworks, such as the NIS2 Directive, the GDPR, and the future Cyber Resilience Act (CRA).																				
	SER0.4	Runtimes should be protected against threats by the enforcement of security controls such as secure defaults, vulnerability scans, intrusion and anomaly detection and continuous security assessment (the specific controls to be implemented will be determined by a risk analysis).																				
	SER0.5	Resources should be protected by an Identity and Access Management (IAM) system, implementing role based access control (RBAC), security zones, and support for a multi-tenant security model.																				
	SER0.6	Integrity of the offloaded functions needs to be guaranteed, including the function inputs and outputs (also during the live migration of FaaS Runtimes).																				

	ID	DESCRIPTION	Device Client						COGNIT Frontend	Edge Cluster		Cloud-Edge Manager						AI-Enabled Orchestrator		Secure & Trusted Execution of Computing Environments		
			SR1.1	SR1.2	SR1.3	SR1.4	SR1.5	SR1.6	SR2.1	SR3.1	SR3.2	SR4.1	SR4.2	SR4.3	SR4.4	SR4.5	SR4.6	SR5.1	SR5.2	SR6.1	SR6.2	SR6.3
Common Requirements	UR0.2	Device applications should be able to upload data from the device ensuring data locality with respect to where the offloaded function is executed.	■	■					■	■												
	UR0.3	Device applications should be able to upload data from external backend storages ensuring data locality with respect to where the offloaded function is executed.	■	■					■	■												
	UR0.4	Execution of functions such as ML inference engines should be able to load machine learning models stored ensuring data locality with respect to where the function is executed.	■	■					■	■												
	UR0.5	Function execution can be executed in different tiers of the Cloud-Edge continuum according to network latency requirements.										■	■									
	UR0.6	Device application shall have the ability to define maximum execution time of the offloaded function upon offloading.							■										■			
	UR0.7	Device application shall have the ability to specify and enforce runtime maximum provisioning time and runtime shall be provisioned within the previously specified time.							■										■			
	UR0.8	Device applications must be able to request and obtain an authorization prior to establishing any further interaction with COGNIT.	■						■							■				■		
	UR0.9	IAM system integration for high granularity authentication and user management for device clients, provisioning engine and serverless runtime.	■						■													
	UR0.10	Push mechanism to inform about status or events from the COGNIT Framework back to the requestor device client.								■												
UC1	UR1.1	Function execution shall be supported in shared, multi-provider environments (with different access and authorization procedures), and the execution must be isolated from other processes on the host system.								■				■		■				■	■	■
	UR1.2	Device application shall have the ability to dynamically scale resources for offloading function execution to maximise exploitation of resources in shared environments, while avoid saturation or resources kidnapping.	■							■	■		■					■	■			
	UR1.3	Function execution should exploit data locality and prioritise edge nodes where the required data is already stored.		■						■									■			
	UR1.4	The whole life cycle of either function execution or code offloading should be auditable and non repudiable.	■							■					■							
	UR1.5	Device applications should be able to request execution over GPUs.	■																			
UC2	UR2.1	It shall be possible to obtain both a-priori estimates of expected, and actual measurements of, energy consumption of the execution of function.							■	■					■			■				
	UR2.2	COGNIT Framework should be able to adapt to rare events with sudden peaks of FaaS requests, in which the offloaded function requires much heavier computations and more frequent execution than usual.																■	■			
	UR2.3	Possibility for devices to request access to GPUs, when available, during high-alert mode.	■							■	■			■								
UC3	UR3.1	Device Client and user applications shall share a maximum of 500 kB of available RAM in total.	■			■																
	UR3.2	It shall be possible for the user application to dynamically scale up/down resources for function execution due to changes in the user preferences.	■																■			
	UR3.3	The SDK for the Device Client shall have support for the C programming language.	■		■																	
UC4	UR4.1	The Device Client should have the ability to dynamically set the permissible edge nodes for executing the function based on policy (e.g. geographic security zones, distance to edge node).	■						■			■					■		■	■		
	UR4.2	The COGNIT Framework should have the ability to live migrate of data/runtime to different edge locations based on policy and location of function execution (e.g. geographic security zones, distance to edge node).	■						■								■		■			
	UR4.3	The Device should be able to request the execution of a function as close as possible (in terms of latency) to the Device's location.	■						■								■		■			

Table 2.1. Final status of each Software Requirement towards meeting its associated global/user requirements.

3. Use Case #1: Smart Cities

Connected vehicles and autonomous driving are expected to revolutionise transportation systems, improving road safety and traffic efficiency while reducing accidents. This Use Case will demonstrate the capabilities of edge computing for future smart city platforms and smart mobility services—the main challenge will be managing dense networks of edge infrastructure resources, as well as a variety of services with different QoS requirements. The transportation systems need to be interoperable, intelligent, secure and support the development of multi-tier edge applications that can be deployed across the cloud-edge continuum, leveraging data locality to reduce overhead, and be managed securely using cloud-native practices.

Densified cities face serious traffic and pollution problems every day, and it is necessary for people to move in a more optimal and safe way to be able to maintain acceptable levels of time and quality in transportation while minimising pollution.

With reference to the organisation of transport in a city, there are still certain differences in the approach according to the cities, but there are structural lines in which all the cities converge, thanks to European directives. Reducing atmospheric pollution, optimising travel time in collective public transport, promoting the use of bicycles or walking through safe and universally accessible infrastructures, these are the common values that will lead citizens to have a better quality of life.

The key aspect for better mobility in a city is public transport, and all the measures oriented to improve it will have a great impact on the city and on the lives of the citizens.

According to Eurostat[3], the average occupancy of vehicles in daily urban journeys does not reach 1.5 people in most of the study countries, and this makes us reason of the inefficiency of private vehicles in terms of the space occupied on the streets and roads in relation to the number of passengers:

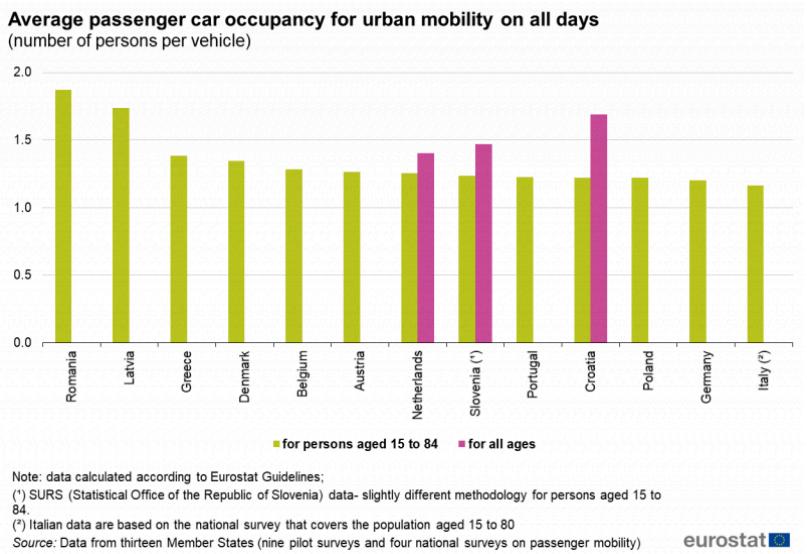


Figure 3.1. Car occupancy

[3] https://ec.europa.eu/eurostat/statistics-explained/index.php?title=File:Average_passenger_car_occupancy_for_urban_mobility_on_all_days_v2.png

On the other hand, road transport is one of the main causes of air pollution and greenhouse gas emissions in urban areas, and the costs of congestion to society amount to approximately €270 billion per year[4].

Poor air quality, especially in urban areas, continues to affect the health of the European population. According to the latest EEA estimates[5], at least 238,000 people died prematurely in the EU in 2020 due to exposure to PM2.5[6] pollution above the WHO guidance level of 5 μg/m³. Nitrogen dioxide pollution caused 49,000 premature deaths in the EU and 24,000 from ozone exposure.

Vehicle-to-everything (V2X) technology is seen as key to improving road safety and enabling full autonomy. Building a functional, interoperable ecosystem requires coordination among all stakeholders. This article reviews Europe's V2X deployment progress and near-term outlook. Europe serves as an ideal testbed for V2X and Cooperative Intelligent Transport Systems (C-ITS), thanks to its advanced road networks, numerous transport operators, and strong automotive industry.

Pursuing Vision Zero[7], Europe has promoted V2X deployment since the early 2010s. To accelerate C-ITS rollout, Member States and infrastructure operators created the C-Roads Platform[8], a cross-border initiative for an integrated, interoperable network.

In the private sector, many automotive OEMs are integrating V2X onboard units (OBUs) into new vehicle models. Volkswagen, an early adopter, has equipped its entire ID. electric lineup with V2X OBUs. BMW recently announced plans to use V2X for vehicle-to-grid (V2G) bidirectional charging, while Mercedes-Benz is focusing on its cloud platform to deliver real-time vehicle-to-vehicle (V2V) alerts.

Yet, despite these initiatives, large-scale V2X deployments remain limited across Europe. The main obstacle to V2X deployment has been the lack of consensus on a common communication protocol. The rivalry between WLAN-based DSRC and cellular-based C-V2X (LTE/5G) has slowed progress, with manufacturers divided by their technological preferences. By 2025, North America and China had largely standardized on C-V2X, phasing out DSRC. In Europe, however, Volkswagen continues to use DSRC, while BMW and Daimler support C-V2X.

Another obstacle that has been slowing down V2X deployment is the lack of incentives. Given the fragmented landscape, governments and regulators play a key role in driving early V2X adoption. Europe's New Car Assessment Programme (Euro NCAP)[9] has mandated that, starting in 2024, vehicles must include V2X connectivity to qualify for a five-star safety rating. This policy is expected to strongly motivate OEMs to accelerate deployment.

[4] https://op.europa.eu/webpub/eca/special-reports/urban-mobility-6-2020/en/
[5] https://www.eea.europa.eu/publications/air-quality-in-europe-2022/air-quality-in-europe-2022
[6] fine particulate matter in the air that is less than 2.5 micrometers in diameter
[7] https://visionzeronetwork.org/about/what-is-vision-zero/
[8] https://www.c-roads.eu/platform.html
[9] https://cdn.euroncap.com/media/30700/euroncap-roadmap-2025-v4.pdf

ACISA's vision is that deploying V2X technology to address real and immediate urban needs, currently managed through legacy, non-standardized solutions, is the most effective approach to overcome those obstacles and accelerate adoption by the cities and fully realize the technology's potential. The current Use Case pretends to demonstrate that vision.

Fortunately, several companies share ACISA's vision, and postulated as key players in the competitive V2X automotive market, including Continental AG, Autotalks, ETrans, Qualcomm, Delphi (Aptiv), Denso, General Motors, HARMAN, Arada, Cohda Wireless, Savari, Kapsch and associations such as Car2Car[10] or 5GGAA[11]. All of them are investing not only in the development of new devices and technologies but also evangelizing towards the adoption of these technologies. .

The European Commission has taken another decisive step supporting the technology by creating the V2X Cluster[12], a collaborative initiative uniting 8 Horizon Europe projects including SCALE[13], Drive2X[14], EV4EU[15], ePowerMove[16], FLOW[17], Neverflat[18], AHEAD[19], and XL-Connect[20], to advance Europe's leadership in interoperable, sustainable, connected, electric and digitalized mobility. The cluster emphasizes that decarbonizing transport requires more than vehicle innovation alone. Central to the V2X vision is enabling vehicles to communicate with infrastructure, energy networks, and each other, unlocking services that enhance efficiency, safety, and resilience. Beyond technical outcomes, the initiative seeks to engage with policymakers, industry stakeholders and civil society to accelerate adoption and remove barriers to large-scale roll-out.

Additionally positive news from outside Europe that emerged in recent years gives reason to hope for widespread acceptance of the technology:

"Hyundai Mobis Co., the world's sixth-largest auto component maker, is slated to partner with Israeli chipset maker Autotalks Ltd. to develop 5G network-based vehicle-to-everything (V2X) integrated control technology to bolster the safety of autonomous vehicles".[21]

"At the end of 2022 ITS America together with 5GAA and other transportation organizations reaffirmed their continued support for the "rapid, widespread deployment of Vehicle to Everything (V2X) technologies in order to further improve safety on American roads". [22]

ACISA has been designing and manufacturing ITS Software and Hardware solutions for cities, roads and tunnels for more than 50 years, and in the last decade all city solutions have been aimed at optimising traffic systems, providing significant improvements in the quality of life of citizens and focusing on reducing pollution and protecting VRUs (Vulnerable Road Users). Nowadays ACISA is developing a new generation of Intelligent

[10] https://www.car-2-car.org
[11] https://5gaa.org
[12] https://scale-horizon.eu/get-to-know-the-v2x-cluster/
[13] https://scale-horizon.eu
[14] https://drive2x.eu
[15] https://ev4eu.eu
[16] https://www.epowermove.eu
[17] https://theflowproject.eu
[18] https://neverflat.eu
[19] https://horizon-ahead.eu
[20] https://xlconnect.eu
[21] https://auto-talks.com/hyundai-mobis-israels-autotalks-to-develop-v2x-for-self-driving-cars/
[22] https://www.itsinternational.com/its4/its6/its8/news/2023-pivotal-year-us-v2x

Transport System (ITS) systems for urban and interurban environments, aspiring to leverage open technologies in the cloud and edge computing fields to achieve a faster pace of innovation, more resilient systems and fewer siloed proprietary solutions. This will benefit both customers, by avoiding vendor lock-in scenarios, and other stakeholders by attracting the vibrant open source community to the ITS domain. For the last two years, ACISA's R&D team has been working on a new concept of Traffic Light Controller (TLC) that aims to leverage the privileged location held by each TLC in the layout of a city, to provision an enhanced distributed computing capacity, which enables real low-latency, high-capacity infrastructure to deploy advanced services for traffic management, video analytics, and mobility.

As a member of Aldesa Group, one of the largest construction groups in Spain, ACISA is a technological company that focuses on innovative engineering solutions offering comprehensive services within four industries:

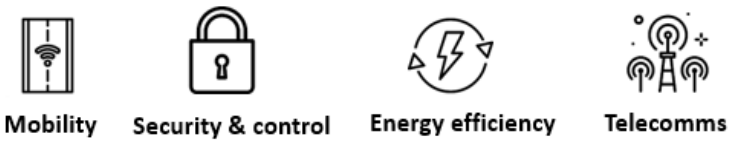


Figure 3.2. Industrial sectors where ACISA is active

With 30 years of experience and a remarkably diversified portfolio of high-profile projects worldwide, ACISA is constantly adapting our technological solutions to the needs of the global market on all levels: social, technological, financial, and environmental. Characterised by personalisation, scalability and adaptability, our software intelligence serves a variety of applications within the business areas of urban and interurban transit, public transport, fleet management and connected vehicle automation.

Believing in positive change, our products are designed to contribute towards a greener, safer and smarter future based on IoT, ITS & C-ITS, ATMS and MaaS – the areas where our R&D efforts are currently focused on. By applying our next generation solutions to improve the quality of life and the efficiency of operations, we are very proudly collaborating with local, national and international stakeholders both from the public and private sector.

ACISA's long experience in the fields of urban and interurban infrastructure, energy efficiency, information technologies and telecommunication services gives us the competitive advantage of quality that our clients trust and rely on.

- 1000 collaborators around the world.
- Management of urban traffic in cities of 200,000+ inhabitants, including: Terrassa, Córdoba, Granada, Alicante, Barcelona and Madrid.
- Maintenance of 100,000+ public light points.
- 600+ km of ITS installed on roads to reduce accident rates.

Our Mobility-Hub (henceforth M-Hub) is designed as an advanced traffic light controller at the edge, aiming to reduce response times and improve the efficiency of public transportation and emergency services. Considering that this infrastructure is often owned by the city councils themselves, providing open and standards-based technology will allow cities to deploy robust smart-city IoT services on top of its already existing traffic infrastructure. Other stakeholders include traffic/mobility departments inside city councils, and other local authorities, such as police, ambulance operators, and firefighters. ACISA acts as the technology integrator that installs, configures, and maintains the ITS systems throughout the term of the contract with the City Council of Granada (Delegation of Mobility, Citizen Protection, Urban Agenda, Sustainability and Next Generation Funds – Mobility Department) for the *"Maintenance and Improvement of the Traffic Light Regulation and Access Control Systems of the City of Granada."*

3.1. Reference Scenario

This Use Case will explore the improvement of the Transient Signal Priority (TSP) service based on V2X technology, to improve public transportation in Smart Cities. This is especially important in densely populated areas, where traffic congestion impacts response times for emergency services, or causes delays to public transportation. The solutions of the future can help save lives and reduce property damage, as well as improve public transportation by reducing travel times and improving schedule adherence. This can encourage more people to use public transportation, which can reduce traffic congestion and improve air quality in urban areas.

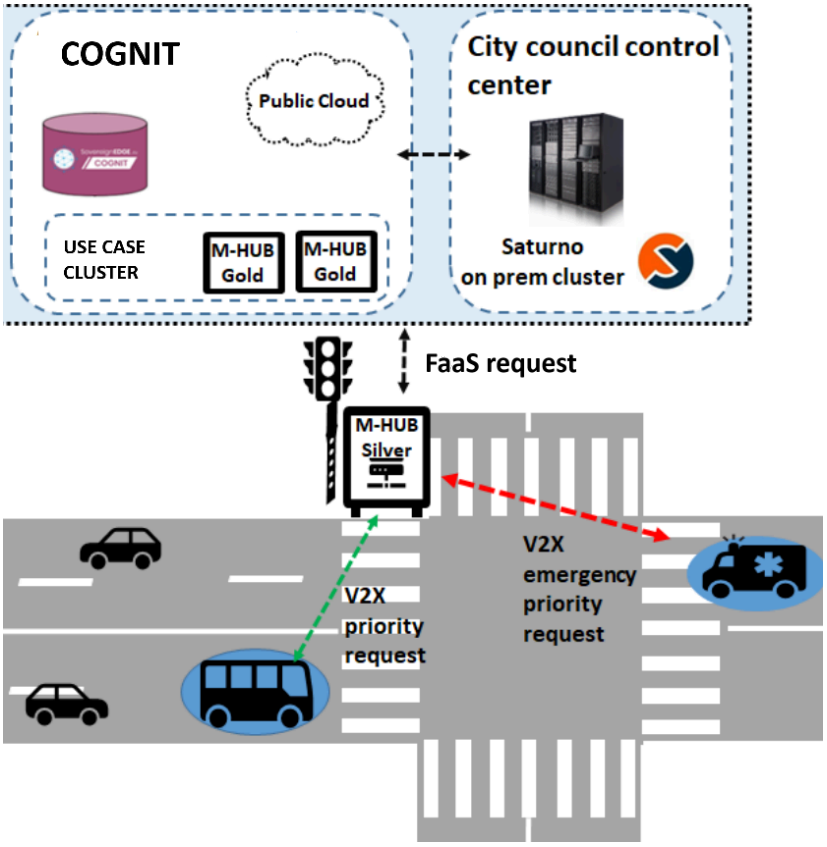


Figure 3.3. Reference Scenario for the Smart City Use Case

In this scenario, a vehicle initiates a priority request by sending a standardised V2X priority request message. Such messages are intercepted by a Road-Side Unit (RSU)—these will be installed along the smart roads of the future—that forwards V2X messages to a nearby M-Hub. A simplified illustration (with the RSU hidden) is given in Figure 3.3. This M-Hub can then decide whether to adjust the traffic signals to grant priority to the approaching vehicle. For example, when an active emergency vehicle approaches an intersection, the corresponding M-Hub will change the traffic signal to green almost immediately in their direction (always ensuring a safe transition), possibly breaking the synchronism of the traffic plan, while other traffic will be held at a red light, as illustrated in Figure 3.4. If the approaching vehicle is instead a bus from the public transport system of the city, the M-Hub will make a decision based on various parameters, e.g. prior delays, the time in the cycle at which the demand is received, and the expected time of arrival at the traffic light. The controller chooses the most efficient method, without altering the synchronism of the overall traffic plan: In case the signal is green, it will try to extend the time of green to allow passage within the same signal phase, as illustrated in Figure 3.5a; in case the signal is red, it will instead try to minimise the time to green, as illustrated in Figure 3.5b.

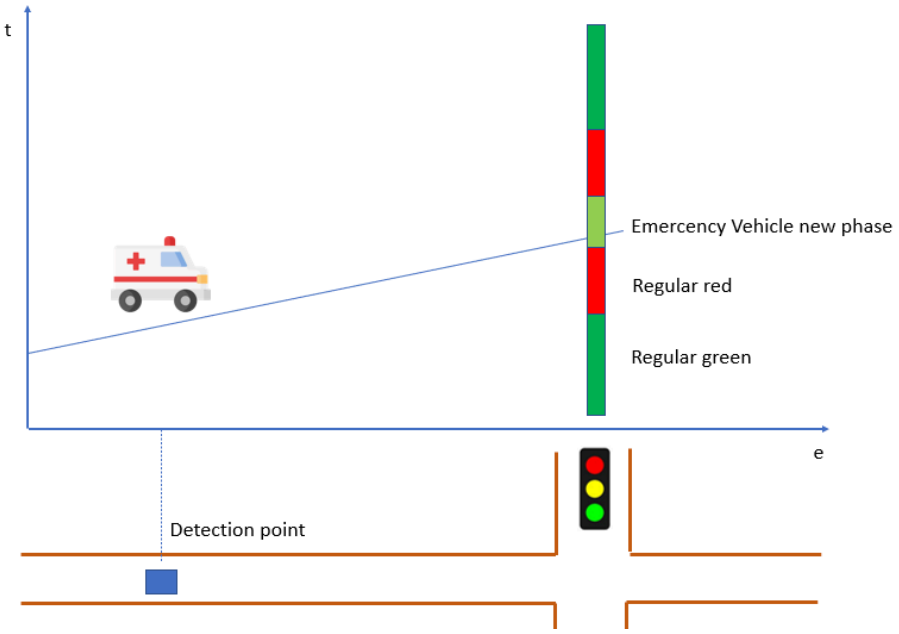
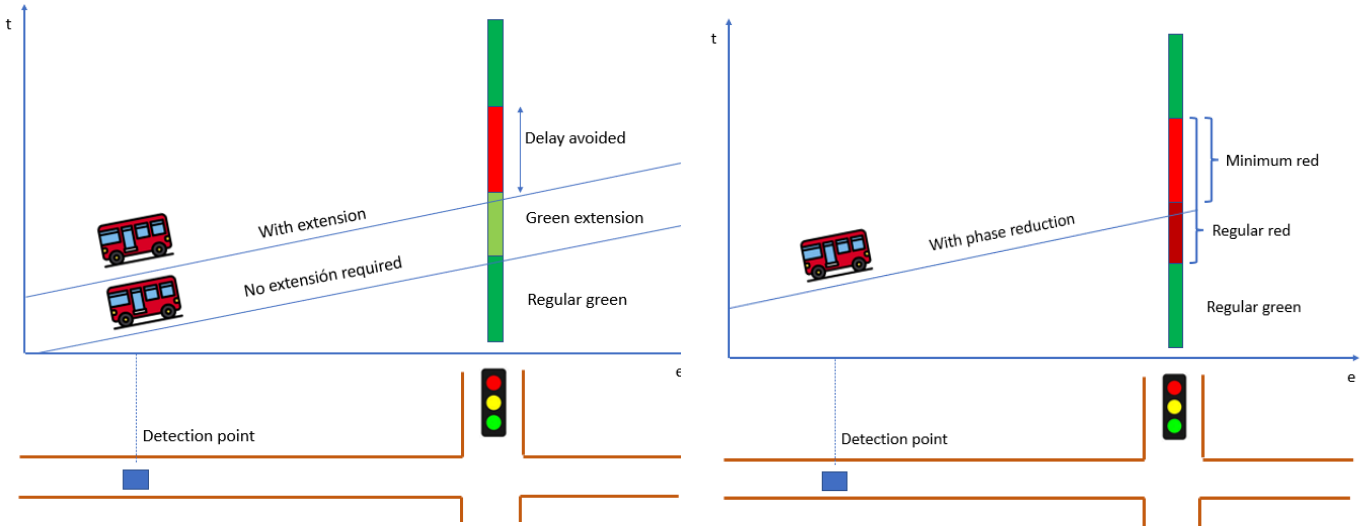


Figure 3.4. Traffic phase diagram for emergency vehicle priority request

Today, the necessary information to process all the traffic priority requests is held centrally in ACISA's traffic management platform Saturno (in the Control Center), but the aim is to distribute it among some of the more powerful M-Hub nodes, to bring this decision closer to the requestor. The less powerful of the M-Hub nodes will instead act as clients that offload computations to the cloud-edge continuum, which combines these powerful M-Hub nodes (the far edge) with the on-prem data center in the Control Center (near edge) and the Cloud, where more computation capacity is available, based on edge applications' requirements.



(a) Green extension diagram **(b)** Red reduction diagram

Figure 3.5. Traffic phase diagrams for public transport vehicle priority request

For reference, current solutions for implementing bus priority, without the use of standardised V2X technology, typically use one of several technologies, such as direct radio communication between the bus and the traffic light, or on board legacy technologies (see Figure 3.6) that asks for priority when the bus or the emergency vehicle is located in a certain area or detection point near to the intersection. The basic architecture of the current solution used by the Use Case is based on an on-premises server located in the Control Center that is running a monolithic application that manages all the priority requests.

Figure 3.6. Legacy sensor for bus detection

To demonstrate the capabilities of edge computing for improving the efficiency of public transportation and emergency services, UC1 focuses on public bus and emergency vehicles prioritisation. A key element is the traffic signal priority mechanism, which allows special vehicles to get priority over other vehicles, when approaching an intersection. This is achieved by modifying TLC phase timings when necessary, either by extending the green phase, reducing the red one, or even forcing a new phase (in the case of emergency vehicles).

Nowadays, when a priority is requested by any of those special vehicles, the central traffic management system first needs to decide if it should be approved or not. This is typically based on variables like traffic congestion, bus schedules, etc., and also external systems that might get consulted. Once priority is granted, the objective is that the vehicle encounters a green light upon reaching the TLC. Therefore, the vehicle's arrival time must be calculated in real time, taking into account factors such as its position, speed, intersection layout, traffic status, etc.

Vehicle priority is currently resolved with custom proprietary solutions that lack standardisation and interoperability, and leave no room for improvement with new features or variables to be considered. All the information needed to process traffic priority requests is held centrally in a Traffic Control Center (TCC), and managed by specialised traffic software suites like the one commercialised by ACISA, called Saturno.

In the context of this use case, an innovative approach is adopted by using standardised V2X communications between vehicles and infrastructure (V2I) to support the priority service.

Cooperative ITS (C-ITS)[23] embraces a wide variety of communications-related applications and standards, including V2X communication technology based on Dedicated Short Range Communications (DSRC), and documented under several European Norms (EN) and Technology Specifications (TS) published by ETSI[24]. 3GPP also included V2X functionalities starting from LTE release 14, based on cellular radio technology (LTE, 5G, 6G). In January 2020, ETSI with the collaboration of 5GAA published EN 303 613[25] defining the use of C-V2X as an access layer technology for ITS in the 5 GHz frequency band. This ensures that all vendors compliant with the standards will be interoperable, avoiding a vendor lock-in effect. In addition, cooperative V2X technology was designed to meet the needs of many safety use cases that can be added over time, with no additional investment in infrastructure.

In addition to these communication standards, application layer V2X services are also being harmonised to ensure vendor interoperability. The current use case is covered in *UNE-CEN ISO/TS 19091 "Intelligent transport systems - Cooperative ITS - Using V2I and I2V communications for applications related to signalized intersections*"[26].

To seamlessly integrate these advanced technologies into urban traffic management, additional requirements must be considered. Specifically, augmenting computation power at the edge is crucial to meet these data processing demands and low-latency applications effectively.

M-Hub is a new generation of TLC designed by ACISA, which incorporates edge computing capabilities, aiming to accelerate the adoption of Cooperative, Connected and Automated Mobility (CCAM) services in urban areas, while providing a common edge infrastructure to other mobility-related systems.

[23] https://www.etsi.org/technologies/automotive-intelligent-transport
[24] https://www.etsi.org/committee/1402-its
[25] https://www.etsi.org/deliver/etsi_en/303600_303699/303613/01.01.01_60/en_303613v010101p.pdf
[26] https://www.iso.org/standard/73781.html

Unique Challenges
Implementation of the scenario described above presents a number of unique challenges and novelties, including:

- Edge services in general and V2X in particular represent a new business case for both ACISA and city councils.
- V2X services are not yet generally adopted by city operators.
- Novel use of a network of traffic light controllers in a city to deploy advanced mobility services.
- Deploy research development in a controlled real pilot.
- Use FaaS technology in the cloud continuum, integrated with V2X and TLC technologies to provide advanced mobility services.

3.1.1. System and Architecture Description

The Use Case involves the following system components, as illustrated in Figure 3.7:

- **Saturno**: ACISA's Smart Mobility platform. It exposes a REST API, and can be deployed flexibly according to the city's needs. It may be hosted on-premises in the city's Control Center, in a public cloud environment or in ACISA's private data center, as implemented in Granada.
- **Mobility-Hub (M-Hub)**—Acting as COGNIT Client: M-Hubs have the capability to receive and process the priority request from the vehicle detection system, besides their conventional functionality as traffic light controllers. A TLC like M-Hub is usually deployed at every intersection in a medium or big city, and is capable of directly controlling the traffic lights at intersections, and, in the particular case of M-Hub, capable of hosting edge computing applications . These are the devices initiating FaaS requests.
- **COGNIT Edge node:** These operate as nearedge nodes in the COGNIT cloud-edge continuum, capable of executing Serverless Runtimes. The information required for processing a traffic priority request (today held centralised in Saturno) will be distributed among the edge nodes. These nodes should therefore in general have all the data needed to execute traffic simulations with specialised engines such as SUMO[27] t. Should it need extra data from external systems, it will be able to request the missing info either from ACISA's traffic management platform Saturno, or any other related information system.

[27] https://sumo.dlr.de/docs/index.html

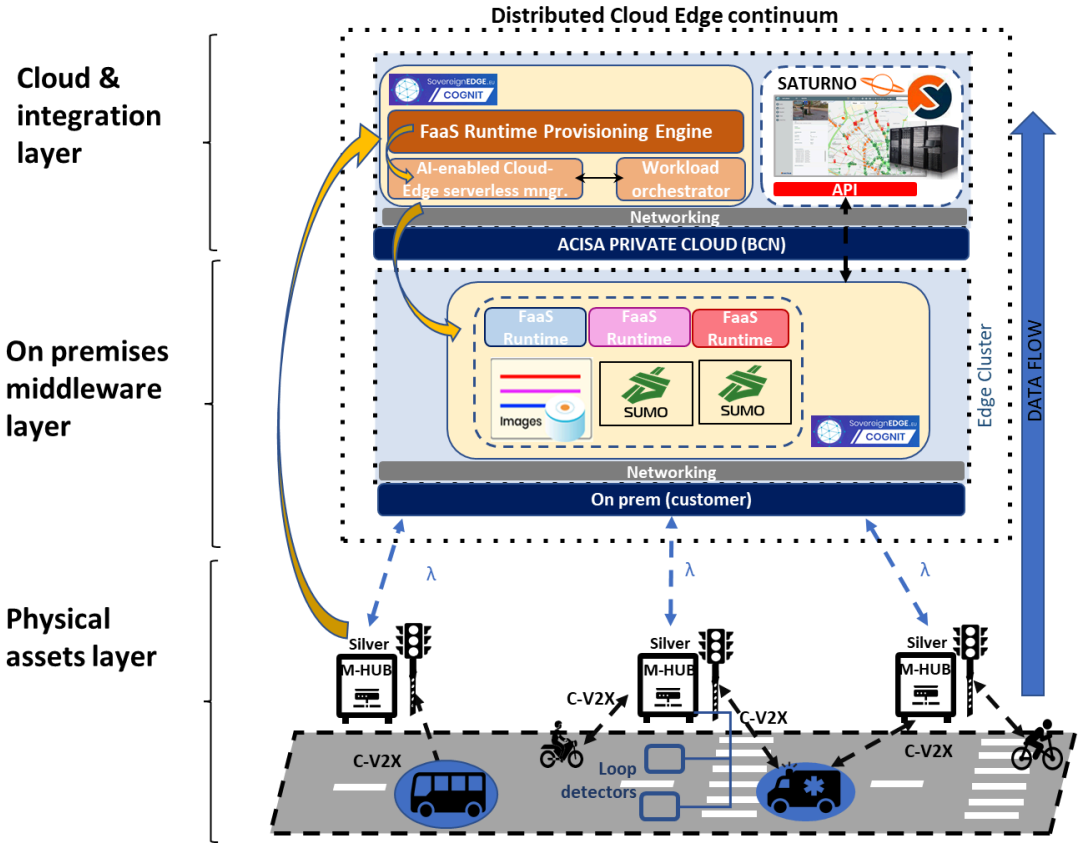


Figure 3.7. Architecture for the Smart City Use Case.

- **On-premise Data Center**—Near edge infrastructure managed under the COGNIT Framework: where COGNIT edge nodes are deployed. This may represent the City's traffic Control Center or a private cloud data center, and will have connections to other information systems that are related to the traffic management of the city.
- **Road-Side Unit (RSU)**: A device capable of intercepting, reading and transmitting standardised V2X messages. RSUs will be installed strategically in close proximity to smart road infrastructure to ensure proper coverage in the area of interest.
- **On Board Unit (OBU)**: Vehicle communication is supported by a V2X On Board Unit, equipped with a Global Navigation Satellite System (GNSS) receiver and its antennas, that can communicate with the RSU through different V2X radio standards (C-ITS or C-V2X). Future versions will be able to connect straight away with a Telco Multi-Access Edge Computing (MEC) node via Long-Term Evolution (LTE) or Fifth generation mobile network (5G).

The reference scenario described in Figure 3.7 involves the following steps:

1) A vehicle initiates a priority (green light) request by sending a standardised V2X Signal Request Extended Message (SREM) message (containing, e.g. bus ID, Lane ID, delay information, and other related data if available), first intercepted by an RSU that forwards the request to the relevant M-Hub, which filters incoming V2X messages for priority requests.

2) Upon receiving a priority request, a FaaS request will be initiated by the M-Hub, to offload the processing to decide whether priority should be given.
3) The function will make use of either traditional algorithms or AI models to decide whether to grant or deny priority, assessing the traffic situation at the intersection under consideration based on the following data from the offloading M-Hub, included as arguments to the Serverless Runtime:
    - Data stored in the particular M-Hub node that initiated the FaaS request, e.g. traffic light controller status, loop detector data (inductive loop under the road surface to detect vehicles), and V2X Cooperative Awareness Messages (CAMs) from other vehicles;
    - Data from the vehicle contained in the V2X SREM message received by the M-Hub node, e.g. bus/vehicle ID, lane ID, other related data and delay information if available.
4) In case the function needs extra data from external systems, it will be able to request the missing info either from ACISA's traffic management platform Saturno, Automatic Vehicle Location (AVL) system from the public transport operator, the control center (CC), or any other related information system.
5) Once the priority request has been processed and analysed, a message will be sent back to the requester through another standardised ETSI message for V2X, Signal request Status Extended Message (SSEM).

For several ITS services, communication in vehicular networks is eminently public, but confidentiality, integrity and non-repudiation must be guaranteed. As an example, V2X messages may not be encrypted for the sake of efficiency, latency, or limited computational power within the vehicles. However, to ensure the integrity of the messages, all V2X messages will carry a digital signature. This measure guarantees that the message was not tampered with during transmission and provides an added layer of security to the communication system.

The Smart City use case will explore the use of V2X technology for public bus and emergency vehicles prioritisation at urban road intersections. In this scenario, a vehicle equipped with a V2X OBU sends a V2X priority request message, which is intercepted by an RSU, which then forwards it to its nearby M-Hub, installed at the intersection.

The objective of the pilot is to move this decision closer to the requesting vehicle in the far edge premises of a customer. M-Hub, as clients of the COGNIT platform, make remote FaaS requests to the COGNIT Edge node , to trigger the computational processes needed to grant or deny priority pass to the vehicle.

The figure 3.8 illustrates how the reference use case architecture can be realised using the COGNIT Framework. In this architecture, the priority simulation is executed by serverless functions provisioned through the COGNIT Framework and deployed on edge cluster nodes, which are installed at the customer's far-edge premises.

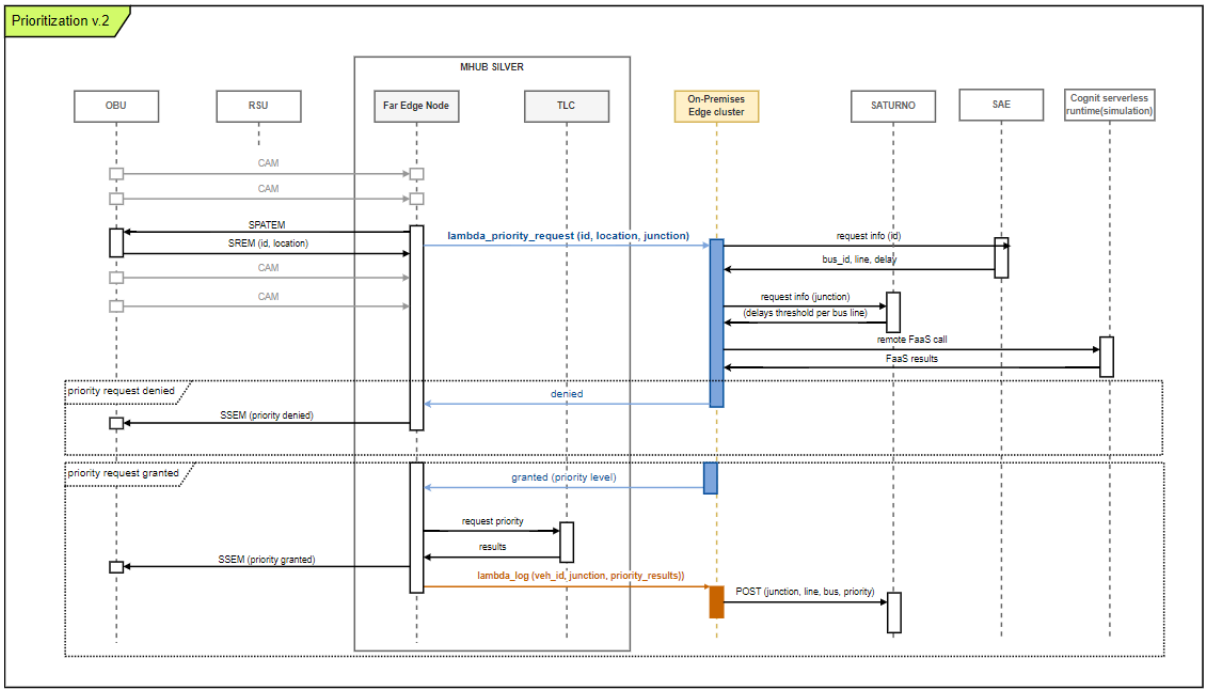


Figure 3.8. Sequence diagram showing different dependent systems

A bus requests priority by sending a standardised V2X SREM message after receiving intersection data through SPATEM and MAPEM V2X messages, as defined in SAE J2735[28] Message Set Dictionary. The SREM message includes essential details such as the bus ID, ingress/egress lanes and, when available, bus delay information. The M-Hub offloads additional verifications to the serverless COGNIT Framework, which possesses all the necessary data to perform traffic simulations and approve or deny the priority request.

Upon receiving a priority request from an authorised vehicle, the M-Hub may initiate a function call to the COGNIT FaaS service, including additional contextual information to execute on demand traffic simulation by the SUMO[29]. This is compute demanding functionality which would be unsuitable for client deployment, instead it will be leveraged using the remote FaaS. The function offloading requires setting up simulation software images in the COGNIT Framework, including simulation models of the junctions and some scripts to process the outcomes, ensuring these elements to be available when the remote call is made. The context information necessary to evaluate the priority requested, is collected by Saturno. There is a recurrent process which offloads that information to the shared data service, so that it is always updated and available for the simulation executions.

Once the priority analysis is done and communicated back to the M-Hub, it generates a response to be transmitted to the requesting vehicle. This response takes the form of another standardised V2X message known as the Signal Request Status Extended Message (SSEM), informing whether the priority has been conceded or rejected.

[28] https://www.sae.org/standards/content/j2735_202007/
[29] Simulation of Urban Mobility, an open source, highly portable, microscopic and continuous multi-modal traffic simulation package designed to handle large networks (https://eclipse.dev/sumo/).

Our use case is focused on leveraging the capabilities of remote execution of complex simulation on the COGNIT Framework. This will provide our edge systems with valuable information to make decisions about how to deal with specific traffic situations.

To run complex simulations, this use case needs some specific functionality so that the models can be executed quickly. Installing the required simulation tool, in our case SUMO, and loading the corresponding models, is a time-consuming operation. Running these steps on demand would be an inefficient way of preparing the remote call.

For this reason, we opted for including the necessary tools and models in the image associated with the service, which gets triggered when a remote call is requested. That way the model is ready to be executed as quickly as possible on demand. When the simulation is run, one of the input parameters is the traffic congestion status, which is always kept updated by the Saturno Traffic Management System and stored in the DaaS.

3.1.2 Technical Challenges

Deploying the priority on-demand system for public and emergency vehicles poses several challenges:

- It must provide a low response time, otherwise the decision about priority might arrive too late to be applied to the traffic regulation.
- Depending on the capabilities and policies of each customer (usually city councils) we can find many different scenarios like:
    - Big cities, with the capability to own and manage complex systems that per policy might want to keep everything on premises.
    - Big cities, with the capability to own and manage complex systems that per policy might want to keep the COGNIT edge cluster and compute nodes on premises, but that would rather prefer to outsource the management of the virtualization solution in a kind of multi-tenant architecture
    - Small cities with no capabilities or will to manage this system, and would rather prefer a cloud wise solution where their traffic controllers rely on making requests to a cloud based COGNIT infrastructure.
- An on-premises serverless solution offers several strategic advantages over hyperscaler platforms such as AWS Lambda[30], Azure Functions[31], or Google Cloud Run Functions[32]. One of the primary benefits is the elimination of cold starts that are common in cloud-based serverless architectures, which can significantly affect performance and response times for latency-sensitive applications. By deploying serverless workloads locally, organizations gain more predictable and consistent performance. Additionally, the cost model of hyperscaler services often becomes unpredictable at scale, with expenses tied to execution time, data transfer, and storage rapidly accumulating. In contrast, an on-premises serverless approach allows organizations to leverage existing infrastructure, leading to lower and more controllable operational costs. Moreover, cloud solutions often require the use of specific operating systems, runtimes, or device configurations dictated by the

[30] https://aws.amazon.com/lambda/
[31] https://azure.microsoft.com/es-es/products/functions/
[32] https://cloud.google.com/functions

provider's environment, adding licensing and compatibility costs. An on-premises solution avoids these constraints, offering greater flexibility, cost efficiency, and alignment with enterprise security and compliance requirements.

- The traffic behaviour shows a strong variability and for that reason we can expect that the demand to the COGNIT systems can have a great variability and the system must be able to scale up or down according to traffic pattern changes. A good balance between commitment of resources, response times, and costs must be found.
- Such variability in the demand might require scaling up the services in different clusters or compute nodes and there must be a mechanism so that the required tools, like simulator and models, are seamlessly available for execution on demand of the FaaS.
- Our use case requires a form of persistence for caching traffic conditions and their corresponding simulation results, allowing us to avoid rerunning simulations with similar input parameters that would yield identical outcomes. This approach enhances overall computational efficiency and reduces redundant processing.

3.1.3 Relevant Key Features of COGNIT

The following features of the COGNIT Framework are of particular relevance to this use case:

- Enable seamless deployment of FaaS and data persistence services on premises and specific nodes.
- Proximity-based deployment, that will reduce the overall latency of the solution.
- Easy management of applications deployed over distributed premises from different customers.
- Safeguarding confidentiality and locality of customer data, even when a multi-tenancy approach is not allowed by them.
- Policies to determine whether a function needs to be executed in specific servers/"zones"/"areas" to comply with stakeholders' compliance requirements.
- Service availability across the cloud-edge continuum.
- Support in the serverless environment, the deployment of complex models and simulation tools to be executed by simple FaaS calls.
- Support the required data persistence.
- Potential access to energy-efficient resources powered by cognitive technologies (i.e. systems that mimic human functions as learning, reasoning, etc.).

3.2. Objectives and validation criteria

The objectives of this Use Case are to:

- Implement a solution for priority management in public transport and emergency vehicles based on V2X technology, through direct interaction between the on-board unit (OBU) and the roadside unit (RSU) and the integration with the M-Hub and with the Saturno cloud platform, both developed by ACISA.

- Develop and deploy our applications on the COGNIT platform.
- Automate resource provisioning for handling V2X messages using COGNIT, to dynamically allocate Serverless Runtimes between the hierarchy of M-Hubs and the Control Center. As the traffic load increases, increasingly dynamic resource orchestration is needed.
- Integrate infrastructure encompassing far edge nodes, on-premise data center (near edge), and public cloud, all managed as a single cloud-edge continuum under one single framework.

3.3. Summary of activities done during the project

This section contains a summary of the activities done during this project. It is mostly focused on the final snapshot, and the history of changes and evolution has been omitted.

3.3.1 Summary of developments

- Creation of an initial Smart Cities Device client using the initial version of the framework and architecture.
    - Making a request to the simulator, passing the simulation parameters and getting the resulting performance metrics.
- Evolution of the Smart Cities Device Client software to adapt it to the new architecture V2.0.Tests of FaaS execution using this new architecture and device client framework software were made successfully. The FaaS has been executed in the compute nodes at the ICE Edge Cluster.
- Tests of FaaS execution in the Edge Cluster at Barcelona. Some manual adjustments in the device client framework were needed to redirect the requests to the Edge Cluster in Barcelona.
- Usage pattern identification: the facilities of Granada were used to gather real traffic information, and real performance of TSP service. These experiences have helped refine the algorithm that will be implemented in the FaaS.
- M-Hub Edge: Added the following functionality:
    - When a V2X bus priority request is received, M-Hub will launch a FaaS to determine whether it grants or denies priority to the bus.
    - The M-Hub will process the FaaS answer and will request priority to the inner TLC.
- Saturno periodically uploads traffic state information to the DaaS as a dynamic input to the simulation model.
- FaaS improvements:
    - For the current scenario, check if equivalent simulations (same timeframe/traffic conditions) are available in the DaaS to avoid simulation time.
    - Otherwise, request the execution of simulations.
    - Calculate a decision with the results of the simulations and return to the M-Hub Edge.
    - The FaaS may cache calculated simulations in the DaaS so that they can be reused to serve further similar requests.

- Tests and validation of the deployment of Transit Signal Priority (TSP) V2X service in Granada: Once the system was launched, statistics on bus detections were being collected, and various tests were carried out in order to find the optimal way to manage priorities. This acquired knowledge will be applied to the final development of the FaaS.
- Digital twin modelling: The digital model of the intersection requires the creation of models within the Saturno platform, and simulation models to be launched within COGNIT's nodes.
- Completed testing (excluding FaaS functionality) and data gathering for the V2X TSP service in Granada, consisting of 38 Mobility Hubs, 38 RSUs managing 114 intersections, along with 19 Bus which were equipped with V2X OBUs.
- Migration of the Saturno platform to new facilities, ensuring its integrity.

3.3.2 Implementation and Deployment Tasks Executed

- Setting up a COGNIT Edge Cluster on-premise as a testbed for UC1.
- Installing and setting up all the M-Hubs to be operative and ready to start testing with the COGNIT Edge Cluster.
- New iteration of the SUMO model for all of the Junctions under test in Granada, to add a new function performance metric closely related to our actual needs about Priority simulation results. Traffic light functionality and traffic demand have been added to be used in the test scenario.
- Creating mock requests using performance monitoring tools to simulate FaaS requests made by M-Hub.

3.3.3 Modeling in Saturno

We researched how to align the messages between the devices and the edge with standards, its content and the equipment configuration as described in Figure 3.7.

Request patterns

According to the samples obtained from our devices on the field, this is a typical pattern on a business day at a specific junction:

Figure 3.9. A snapshot of the priority request pattern obtained in the test bed on a typical day.

Traffic data is aggregated in 10-minute periods. The global number of requests will depend on the total number of buses and junctions involved, and it will depend on the field deployment, but this graph gives us an order of magnitude of the volume of FaaS requests to be expected.

Digital Twin of urban road intersections

Every intersection within a city varies in terms of its topology, traffic model, M-Hub program, and the specific manoeuvres that a vehicle may request priority for, resulting in a high degree of complexity. Figure 3.10 provides a few examples extracted from UNE/CEN ISO 19091[25] but there are unique scenarios and for this reason, each intersection needs to be modelled with its particular differences in terms of topology, traffic flow, traffic demand, bus stop location, crosswalk, TLC programs, bus stops, etc.

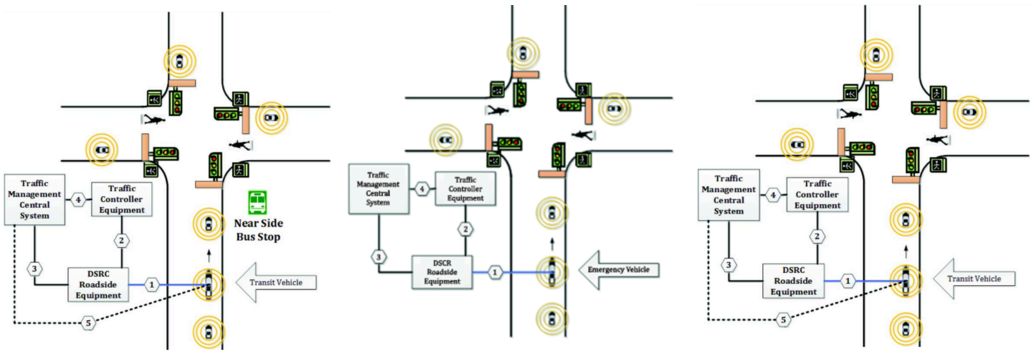


Figure 3.10. Examples of potential intersection scenarios in Transit Signal Priority (TSP). (Source UNE-CEN ISO/TS 19091)

Each intersection has its counterpart digital representation, a.k.a. Digital Shadow, holding updated historical data about its layout, current traffic status, subsystems status and alarms (M-Hub, detectors, others), etc. By incorporating a feedback actuation based on the traffic simulation results, the Digital Shadow evolves into a Digital Twin, which will act over the intersection traffic lights to grant or deny the priority based not only on current traffic status, but also on its simulated forecast predictions.

In Saturno we have a Digital Shadow of each intersection we manage, with detailed information as shown in Figures 3.13 and 3.14 . During the project we designed its simulation models to be launched from COGNIT's nodes, incorporating real traffic information from the intersections if available, as shown in Figure 3.11.

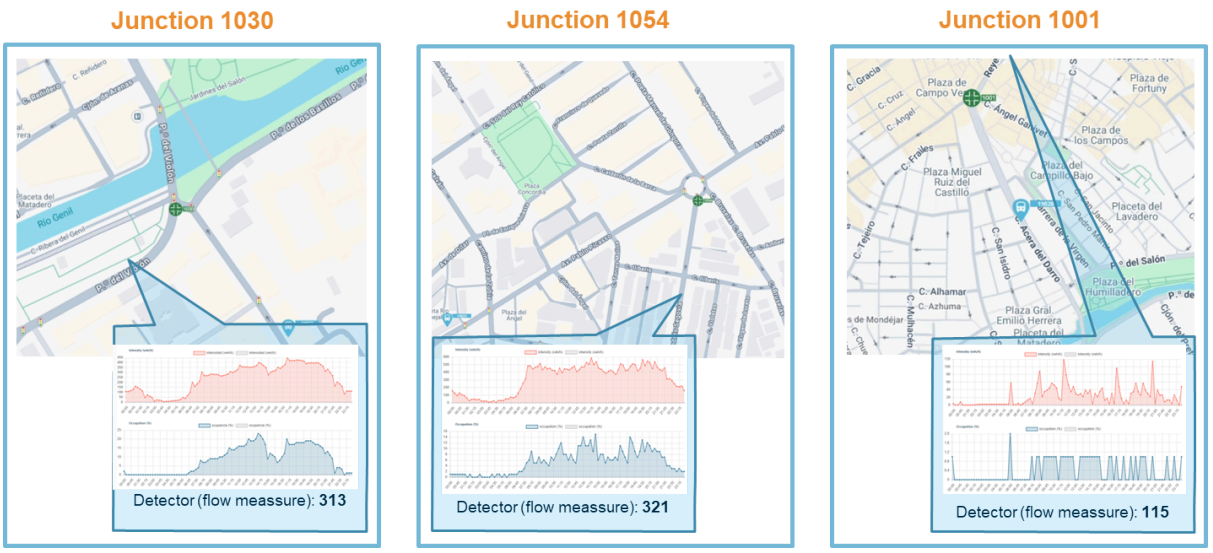


Figure 3.11. Example of the Digital Shadow of Granada's intersection in Saturno.

3.3.4 Enabling V2X communication

Our chosen approach within the project, selected among multiple possible options, has been to work with a bus priority and emergency vehicles (E-V) system based on C-V2X (cellular V2X and ETSI C-ITS-G5) connected vehicle technology, which has enabled a new perspective to the city's bus and E-V priority. All the solutions are deployed within a continuous cloud-edge infrastructure and taking advantage of the serverless infrastructure provided by COGNIT.

V2X technologies encompass communication between vehicles (V2V), vehicles and infrastructure (V2I), vehicles and pedestrians (V2P), and vehicles and the network (V2N). V2X allows vehicles to exchange important information and data, enhancing safety, efficiency, and cooperating to exchange information and avoid hazardous situations, in an overall driving experience.

This COGNIT UC1 is using V2X technology to enable real-time communications between public transport and emergency vehicles, such as ambulance, police or firefighters and traffic signals, allowing for a more dynamic and efficient traffic light prioritisation. With the use of V2X technology, public transport and emergency vehicles can communicate with the traffic light infrastructure and other vehicles in real time, enabling them to move more quickly and with greater reliability. V2X-based priority systems can reduce emissions and improve air quality, making public transportation more sustainable by reducing buses' time idling at intersections and providing a better QoS. In the case of E-V the improvement will impact directly on the safety of the E-V and its occupants.

The main reason why this Use Case needs the COGNIT Framework is that all the connected vehicles (not only bus and E-V) will generate in the near future a massive data throughput that needs to be processed in the edge with extremely low latency in order to take decisions that directly affect the safety of the users. These decisions have to be taken by

small computing devices (like TLCs) that need to offload some parts of the code with the FaaS serverless service supported by COGNIT.

ACISA aims to promote the progressive adoption of V2X technologies through this use case, which stands out because it does not require the direct involvement of car manufacturers or large-scale technology deployment. In all locations where the public bus and emergency vehicle priority system is implemented, V2X coverage is ensured through the installation of RSUs, and integrating them with ACISA's M-Hub. All these equipment can later be reused to support and enable additional V2X-based services for future urban mobility use cases.

3.3.5 Integration of standards

This Use Case contributes to global project goals by implementing standards related to V2X services deployed in the far edge of the continuum. The priority request functionality adheres to the principles outlined in UNE-CEN ISO/TS 19091, titled *"Intelligent transport systems - Cooperative ITS - Using V2I and I2V communications for applications related to signalised intersections."*[33] This standard is built upon several specifications from ETSI and SAE. We would like to emphasise the following ones:

- ETSI TS 103 301 (V2.1.1)[34]: *"Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Facilities layer protocols and communication requirements for infrastructure services"*.
- ETSI EN 302 665 (V1.1.1)[35]: *"Intelligent Transport Systems (ITS); Communications Architecture"*.
- ETSI TS 102 894-2 (V1.3.1)[36]: *"Intelligent Transport Systems (ITS); Users and applications requirements; Part 2: Applications and facilities layer common data dictionary"*.
- ETSI EN 302 636-4-1 (V1.4.1)[37]: *"Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1: Media-Independent Functionality"*.
- ETSI TS 103 097 (V1.4.1)[38]: *"Intelligent Transport Systems (ITS); Security; Security header and certificate formats"*.
- ETSI EN 302 637-2[39]: *"Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service"*.
- SAE J2735:2016[40], Dedicated Short Range Communications (DSRC) Message Set Dictionary {A, B, C}

[33] https://www.une.org/encuentra-tu-norma/busca-tu-norma/iso?c=073781
[34] https://www.etsi.org/deliver/etsi_ts/103300_103399/103301/02.01.01_60/ts_103301v020101p.pdf
[35] https://www.etsi.org/deliver/etsi_en/302600_302699/302665/01.01.01_60/en_302665v010101p.pdf
[36] https://www.etsi.org/deliver/etsi_ts/102800_102899/10289402/01.03.01_60/ts_10289402v010301p.pdf
[37] https://www.etsi.org/deliver/etsi_en/302600_302699/3026360401/01.04.01_30/en_3026360401v010401v.pdf
[38] https://www.etsi.org/deliver/etsi_ts/103000_103099/103097/01.04.01_60/ts_103097v010401p.pdf
[39] https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.03.01_30/en_30263702v010301v.pdf
[40] https://www.sae.org/standards/content/j2735_201603/

Figure 3.12 shows a bus requesting priority by sending a standardised V2X SREM message after receiving intersection data through SPATEM and MAPEM V2X messages, as defined in SAE J2735[41] Message Set Dictionary. The SREM message includes essential details such as the bus ID, ingress/egress lanes and, when available, bus delay information. The M-Hub offloads additional verifications to the serverless COGNIT Framework, which possesses all the necessary data to perform traffic simulations, and approve or deny the priority request.

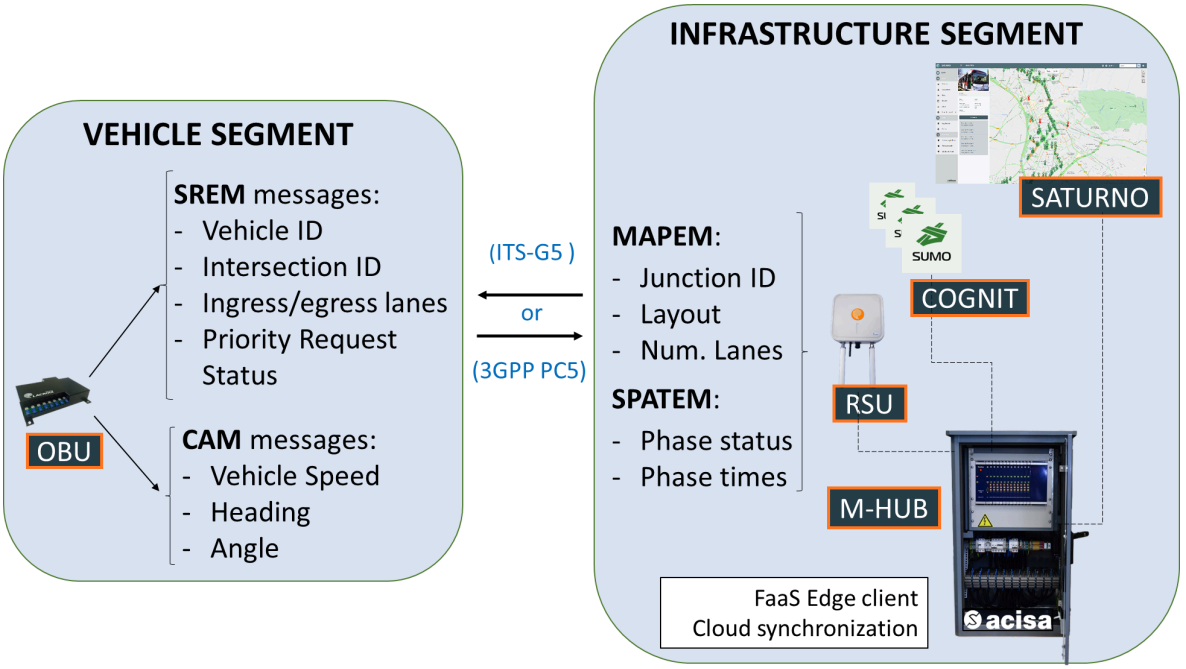


Figure 3.12. Main C-ITS messages used in TSP service.

We have included a sample of a V2X message sent from M-Hub to Saturno below:

```
None
{
    "junctionCode": "****",
    "busLine": "**",
    "busNumber": "*****",
    "delayLevel": "none",
    "minDelayLevel": "none",
    "dtIngress": "2024-10-03T17:53:31.530Z",
    "dtTransit": "2024-10-03T17:54:50.530Z",
    "dtEgress": "2024-10-03T17:55:07.530Z",
    "laneCodeFrom": 3,
    "laneCodeTo": 4,
    "priorityState": "done",
    "demandCode": 1,
    "priorityType": "minimum",
    "stoppedTime": 26,
```

[41] https://www.sae.org/standards/content/j2735_202007/

```
    "priorityLevel": "low",
    "granted": true,
    "expectedStop": false,
    "comment": "Operation done"
}
```

Figure 3.13. V2X message format

The RSUs can send information about the traffic light groups and the topology of the intersection to the M-Hub system. This information is used in the digital twin simulations that run on COGNIT. To integrate the RSU with an edge node, the use case has implemented a communication interface using a REST API and websocket.

Figure 3.14 shows a sample of the equipment installed, consisting of M-Hubs and V2X RSUs on each intersection, to communicate with any of the 18 buses equipped with a V2X On Board Unit.

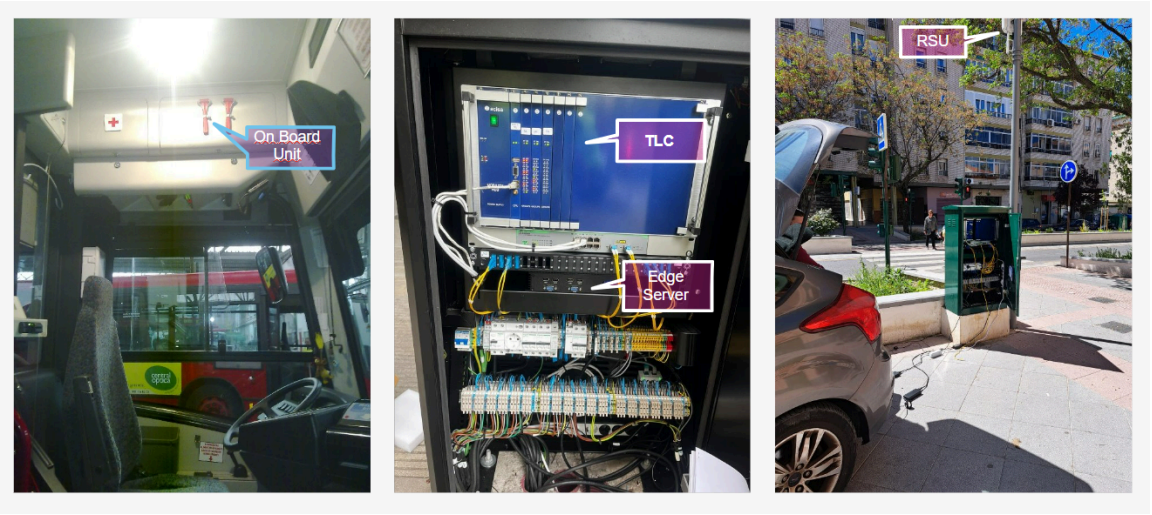


Figure 3.14. Installation of V2X On Board Units (left), M-Hub (center), and RSU (right) in the city of Granada.

Prioritisation Algorithm

The objective of the pilot is to move the priority request decision closer to the requesting vehicle in the far edge premises of a customer. M-Hub, as a client of the COGNIT Platform makes a remote FaaS request to COGNIT edge node to trigger the computational processes needed to grant or deny priority pass to the vehicle.

The main goal of the system is to avoid the stopping time of each bus at the traffic signal, thus the travel time crossing the junction will be reduced. To measure the results, each priority operation is monitored, collecting data in two scenarios: “With” and “without” the priority system enabled. In this project, the priority system is using the more advanced digital twin simulations enabled by COGNIT.

A challenge for this prioritisation schema is that the arrival time for the bus has some uncertainties. To reach the objective, the traffic light controller needs to know when the bus will arrive at the junction, in order to modify the green or red light timings. One of the main problems is the difficulty of determining the precise moment when the bus will arrive, especially if there are bus stops along the ingress path. These stops introduce a high level of indetermination in the estimated time to approach.

Although the integration with COGNIT has been deployed on field for testing, as it is a real world scenario with a critical feature as is traffic management, we limited the integration to capturing data of the requests and the results, and this needs more testing and more discussion with the city council before applying the priority to real traffic.

3.3.6 SUMO models for simulations

All the intersections under test were modeled in SUMO, and the FaaS service was deployed on all the M-Hubs to test the performance of all the M-Hub sending requests to COGNIT in the form of FaaS.[42]

With the new release, the SUMO model includes paths relevant to the priority calculations for the junction and traffic generation patterns based on the traffic observed in that area.

Simulation tool installation

We use SUMO to model and simulate several intersections in the City of Granada (Spain) and create workloads for FaaS Runtime Server. There are two main steps as prerequisites to create network scenarios and workloads for FaaS Runtime Server.

1) Install SUMO 1.2.0, Python, and Traffic Control Interface[43] (TraCI) Library to interface Python with SUMO.
2) Import Map data files of intersections inside the SUMO tool.

To install SUMO in our UC1's virtual machine we followed this procedure:

```
None
# sumo requirements
sudo zypper install python-xml
sudo zypper install unzip

# Installing sumo
zypper addrepo
https://download.opensuse.org/repositories/home:behrisch/15.4/home:behrisch.repo
zypper refresh
zypper install sumo
export SUMO_HOME=/usr/share/sumo

# Installing TRACI
pip install traci
```

Figure 3.15. Deployment of SUMO

[42] https://github.com/SovereignEdgeEU-COGNIT/use-case-1/tree/v.2.2.2/vm/model/Granada
[43] https://pypi.org/project/traci/

After installing SUMO and its requirements, we need to select the intersections and import their map data inside the SUMO tool. For this purpose we use OSMWebWizard[44] which provides a graphical interface to zoom and select specific intersections and get their map data. Indeed we can get snapshots of particular intersections and create real network scenarios using OSMWebWizard. Figure 3.16 shows an example of importing a scenario from intersections of the City of Granada and generating network scenarios using a Python script:

```
None
>> python osmWebWizard.py
```

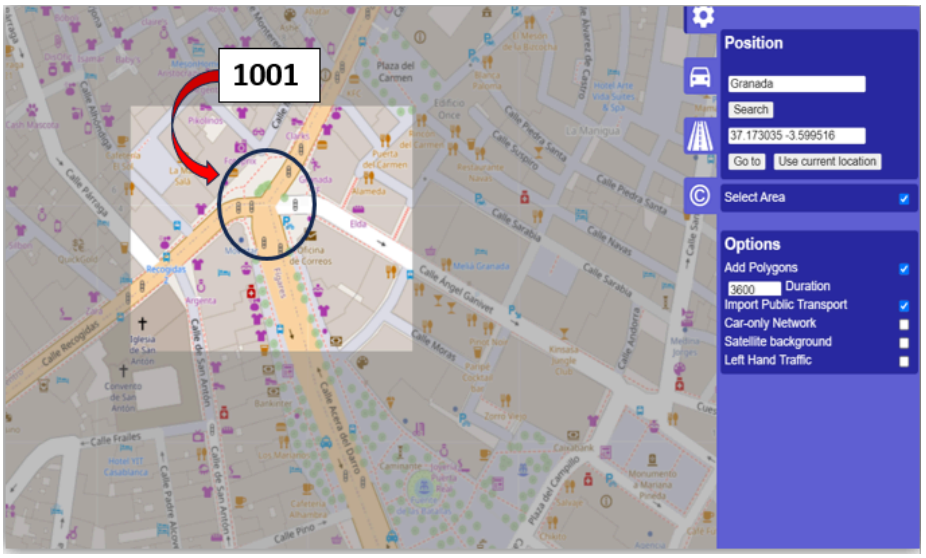


Figure 3.16. The snapshot[45] of intersection with id 1001 from City of Granada

[44] https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html
[45] In this figure, map data from OpenStreetMap. In the models, the included data map is from OpenStreetMap, please see this copyright notice: https://www.openstreetmap.org/copyright

This is a view of the network editor of SUMO for the 1001 junction:

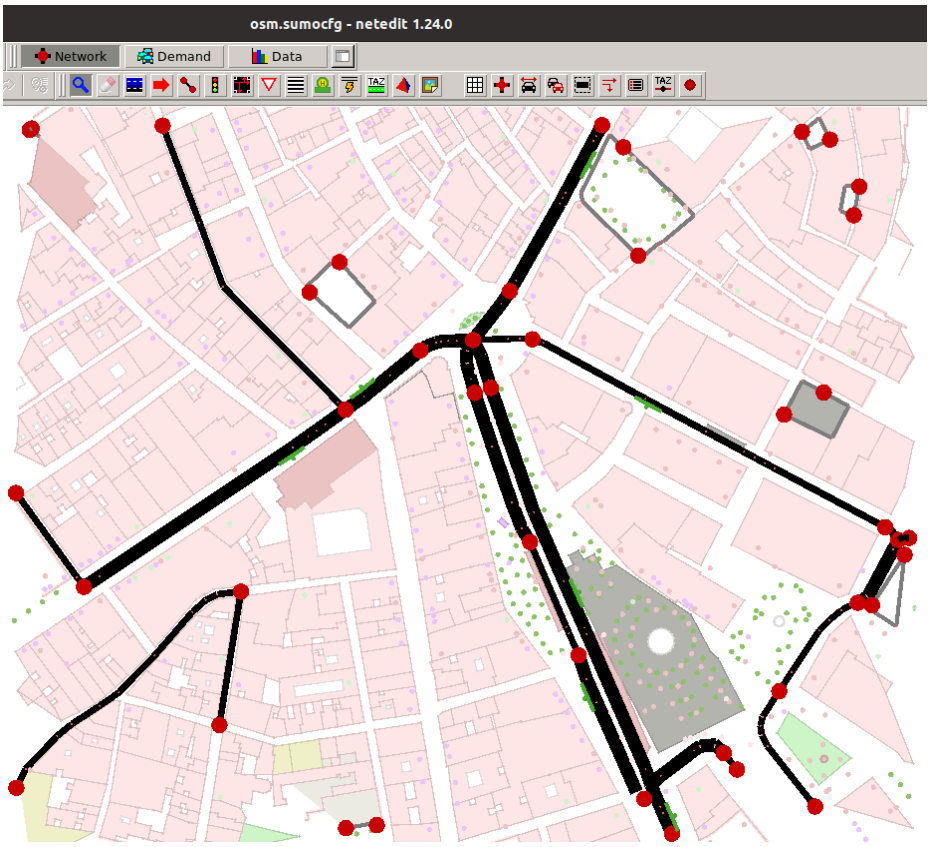


Figure 3.17. SUMO Network editor

Where the red dots represent SUMO internal network Junctions, and the black or gray lines represent the lanes involved in the simulation.

Below is an example of a frame from a graphical simulation, showing a red bus and several yellow cars

Figure 3.18. Intersection 1001 modeled both in SUMO(left) and Saturno (right)

Calling SUMO

We deployed several scripts in the image of the Serverless runtime to call SUMO and parse the results.

The script is used by the FaaS request to call SUMO and process the results of the simulation. It takes as parameters the city and junction code, and the congestion status.

Parsing the results of SUMO

The simulation generates files with statistics of the simulation. In the current phase, we opted for the timeLoss as a KPI for our priority calculations. We are taking into account the timeLoss of the bus because it is the parameter we want to minimize.

There are several files generated with the results of the simulation. For our case, the **tripinfos.xml** file contains the relevant data.

To get the timeLoss data, we can search for elements like:

```
<tripinfo id="pt_bus_121:0.0" depart="15.00"
departLane="852062315_0" departPos="12.10" departSpeed="0.00"
departDelay="0.00" arrival="61.00" arrivalLane="852062315_0"
arrivalPos="58.91" arrivalSpeed="7.90" duration="46.00"
routeLength="46.81" waitingTime="1.00" waitingCount="1"
stopTime="30.00" timeLoss="9.60" rerouteNo="0"
devices="tripinfo_pt_bus_121:0.0" vType="pt_bus"
speedFactor="1.00" vaporized=""/>
```

Figure 3.19. Simulation data of interest

The id will allow us to determine if the vehicle is a bus, because all the bus records will have a “pt_bus” prefix.

The script is used to parse the results of the simulation.
It processes the timeLoss result of the different trips and returns a value of the time loss of the whole simulation, which will be used later to make a decision about the priority.

New features in M-Hub Edge

The device-client runtime[46] of COGNIT has been deployed in the M-Hub Edge, so that upon receiving a V2X priority request, a FaaS request is generated.
The FaaS result is returned to the M-Hub Edge with a recommendation of the priority grant. Then the M-Hub Edge accepts or rejects the priority request depending on the response given by the FaaS.

DaaS integration

In this phase, some data has been identified which would be convenient to store persistently:

[46] https://github.com/SovereignEdgeEU-COGNIT/device-runtime-py

- Traffic status, i.e. level of congestion.
- Results of previous executions of the simulation, classified by context.

Traffic status

Saturno Traffic Management System from ACISA collects data from detectors, which can be processed to obtain a value of congestion for each junction:

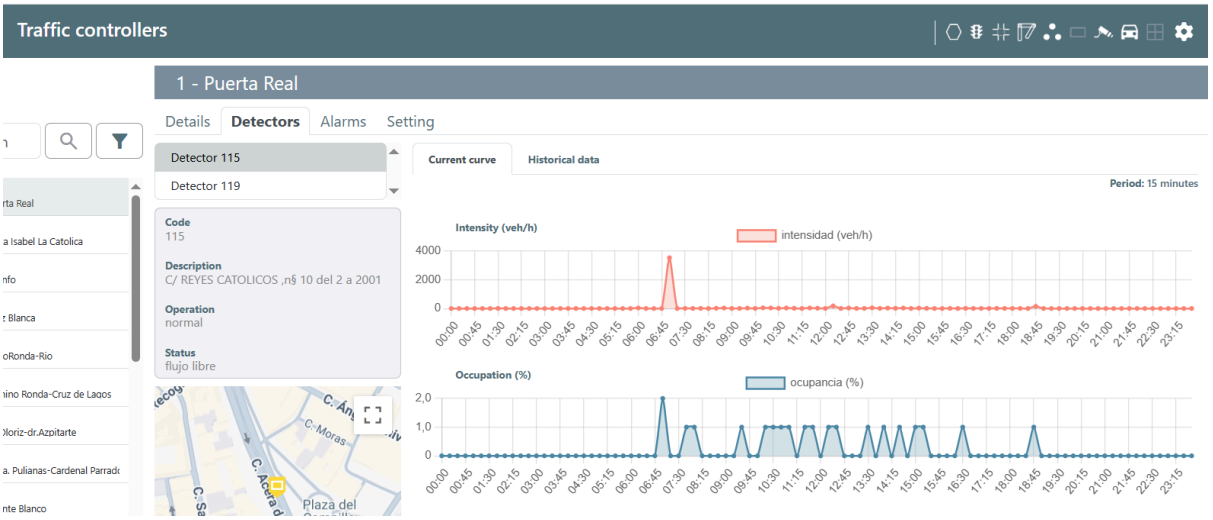


Figure 3.20. Detector data collected by Saturno

In addition, we created a flow with Nodered, so that periodically, the value of congestion of the junctions is written to the DaaS, so that it is available for the FaaS.

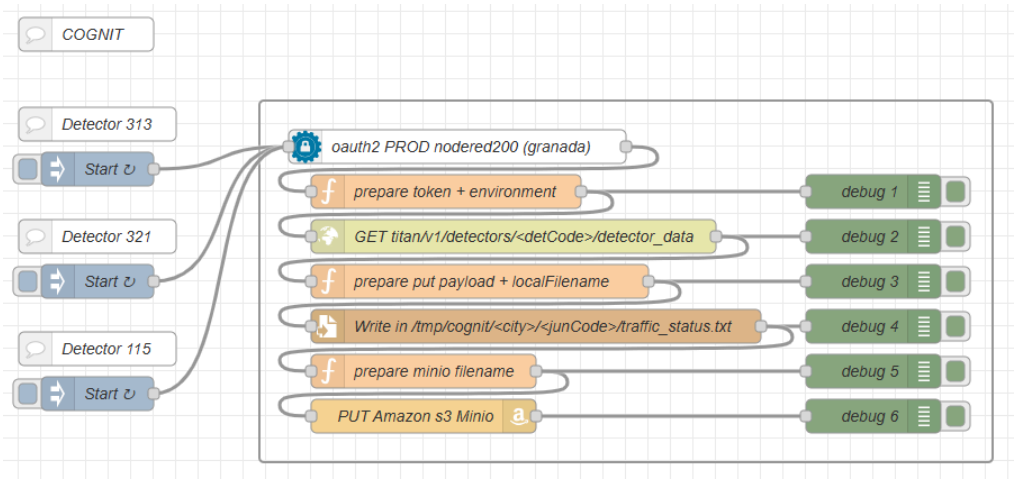


Figure 3.21. Integration between Saturno and DaaS

We are using the following format to interchange this kind of data:

File path in DaaS: smartcity/Granada/1001/traffic_status.txt

CONGESTION: LOW

```
LAST_UPDATE: 2025-08-29T19:00:01+02
```

Figure 3.22. Congestion file format

The code in the FaaS, i.e., the get_traffic_status function will collect the file {city}/{junction}/traffic_status.txt

For evaluation purposes, there is an environment variable, FORCE_CONGESTION, to force a specific congestion level and ignore the real traffic status provided by Saturno.

There is a fallback, just in case the file is not available, to consider the congestion level as MEDIUM.

Precalculated simulations

We assume that for similar conditions the result of a simulation will usually be the same. Then we need to characterize the current context of the traffic so that we can store the simulation result for such a situation, so that this simulation result can be reused when the contexts are the same.

We chose to use the following dimensions:

- Season: can get the following values: spring, summer, autumn, winter, using the following function:

```
def season(mmdd):
    if mmdd >= "0301" and mmdd <= "0531":
        return "spring"
    elif mmdd >= "0601" and mmdd <= "0831":
        return "summer"
    elif mmdd >= "0901" and mmdd <= "1130":
        return "autumn"
    elif mmdd >= "1201" and mmdd <= "0229":
        return "winter"
    else:
        raise Exception("Month/date out of range")
```

- Weekday: can get the values workday or weekend
- Hourly range: We established the following ranges, where for example (0,5) means 00:00:00 to 05:59:59:

```
(0, 5), (6, 9), (9, 12), (13, 15), (16, 19), (20, 22), (23, 23)
```

- Traffic congestion: which can take the values LOW, MEDIUM, HIGH

Using those dimensions, a filename is composed so that it is easy to check if such data is available in the DaaS. This behaviour can be overridden using the environment variable

CACHE_RESULTS. If set to true, the precalculated values are used. If not, every request will trigger a simulation.

This kind of optimization can be disabled using one of the function parameters, for example if we want to make workload tests.

Priority decision

We made an analysis of the behaviour of the junctions of the pilot and opted for timeLoss thresholds between 4.65 and 53.8 for the 38 junctions, to determine if priority should be granted.

If the value returned from the SUMO simulation is above the threshold, then it makes sense to give priority to the bus, and the priority is granted:

```
def calculate_priority(junction, result):
    return mean_timeLoss > threshold(junction)
```

Saving priority as a precalculated value

Following the same naming convention as specified above for priority calculated data filenames, a new file is created with the result of the simulation, for future use.

Overview of the FaaS for SmartCities

For the current scenario, check if equivalent simulations (same timeframe/traffic conditions) are available in the DaaS to avoid simulation time.

Otherwise, request the execution of simulations.

Calculate a decision with the results of the simulations and return to the M-Hub Edge.

The FaaS may cache calculated simulations in the DaaS so that they can be reused to serve further similar requests.

Once we reviewed in the previous sections the different elements of the logic, this is the main process of the FaaS.

Parametrization of the process

We needed an efficient way to configure the behaviour of the FaaS, and we also needed to test the behaviour efficiently under different circumstances. To facilitate this, we created different environment variables:

- **ACCESS_KEY_ID, SECRET_ACCESS_KEY**: Credentials to access the DaaS. Required.
- **CITY**: City name. Used to compose the path where the SUMO model is stored. Required.
- **JUNCTION**: String with the junction code to simulate. Used to compose the path where the SUMO model is stored. Required
- **FORCE_CONGESTION**: Usually the congestion is a value stored in the DaaS by one process running in Saturno. This environment variable is available to force a

situation of congestion for testing purposes and ignore the data provided by Saturno. Valid values are LOW, MEDIUM, HIGH. Not required. Defaults to MEDIUM.

- **CACHE_RESULTS**: When true, the result of the simulation and its context are stored in the DaaS, so the result can be reused without running the simulation, if the context is the same. The context includes values as workday or weekend, season, hourly range, and level of congestion. When false, the precalculated values are not used, every time there is a request, the simulation is run and the resulting value is stored in the DaaS for future use.

3.3.7 Lessons learned

The different partners of the project followed an incremental approach. This was an efficient way to be able to quickly test features and allow the tools and requirements to mature as we were having a clearer idea of what the processes and needs actually were.

Of course, there were many moving parts in this project (infrastructure, virtualization platform, workload management, AI orchestrator, serverless runtime, device client, and use cases), and it has been a challenge to have all those stakeholders coordinated. We had to make a special communication effort with the different partners so that everybody was on the same page.

We made an effort to share with the people managing the infrastructure, the different workflows, and approaches that we were applying to minimise the impact of changes.

Another good practice from this project has been to share knowledge, so that when similar issues needed to be solved, we didn’t need to start from scratch, but we could likely use the experience of some other partner in that area.

3.3.8. Follow up on Risks and Mitigation Plan

The project's risk analysis identified a set of technical and operational risks with moderate overall impact levels, ranging from 2 to 3 on a 5-point scale. Most risks relate to cybersecurity, communication protocols, equipment integration, and performance under real-world conditions. Proactive mitigation strategies were established for each case, including adherence to cybersecurity best practices and ISO 27001 controls, protocol adaptation, early equipment procurement, and detailed documentation reviews for V2X integration.

Overall, the project offered the use case robust risk management and contingency planning, ensuring timely deployment, stable operation, and adaptability for future evolutions of the M-Hub and its integration with the COGNIT Framework.

Nº Risk	**Potential risks**		**Contingency plan**		**Comments**
	Level (1:low; 5:high)	***Impact***	***Description***	***Responsible***	
1	3	Cybersecurity requirements put the security of the M-Hub at risk.	Use of best practices in cybersecurity, and ensure the inclusion of monitoring and control tools in edge application deployment pipelines. Use of ISO 27001 control that may apply.	Technical lead	We applied our common security measures. In addition, the deployment in Granada is in an isolated network, which reduces risks
2	2	HTTP or MQTT protocol is not efficient enough for certain devices or its implementation is not feasible.	Develop a protocol conversion server with the possibility of maintaining direct TCP/IP links to specific protocols.	Technical lead	Risk not materialized

3	3	Problems related to the integration of V2X communication systems, including cybersecurity aspects.	V2X equipment documentation shall be requested and analysed.	Project lead	No major concerns on the deployment of the V2X equipment.
4	3	Delays related to equipment procurement.	ACISA already has V2X equipment and a private cloud with the capacity to allocate the rest of the equipment.	Project lead	Equipment available on time
5	3	Implemented functionality does not give good results when applied in a real environment.	ACISA sees this framework useful in other types of real Use Cases like federated learning, urban digital twin and transfer learning related to traffic and mobility or multi-tenant smart-city infrastructure.	Project lead	In the real-world scenario we found several challenges that could be addressed in further evolutions of ACISA's priority algorithm: - Impredictability of the bus arrival once priority is requested due to traffic conditions(e.g. congestions in the road, delays at bus stop, incident, …). - The currently implemented priority bus-arrival estimation method is being refined heuristically by ACISA, although future iterations of COGNIT may support a machine-learning–based implementation.
6	3	No collaboration agreement with any city council or permission to implement the pilot test	ACISA already has agreements in real scenarios for pilot testing, but it is also possible to Install a M-Hub and RSUs in a laboratory or controlled environment.	Project lead	Not materialized

		in real city conditions is obtained.			
7	3	The results of the pilot test indicate that there are stability and scalability problems.	The equipment in the private cloud will be oversized to accommodate with enough slack the computational resources required. ACISA has an AWS account that might be used if scalability is needed. M-Hub can be virtualized, and thousands of edge nodes could be used in order to test the COGNIT Framework at scale.	Technical lead	In a very initial stage of the evolution of the deployment, some scalability issues were found during the performance tests, which were quickly addressed and solved in the following evolution of the platform.
8	2	Delays in developing the specific parts inside M-Hub to provide access to COGNIT Framework or developing the specific parts inside M-Hub to provide access to COGNIT Framework. Delays implementing FaaS or the Runtimes.	Developers will be trained as soon as the COGNIT Framework is available in order to implement and test it.	Project & Technical leads	This risk materialized and was solved adding members to the team with proper training and with a close collaboration between the M-Hub and the FaaS developers.

Table 3.2. Risks and mitigation plan

3.4. Use Case Infrastructure, Demonstration, and Validation

Demonstrations and validations of the Use Case have been conducted in a controlled environment dedicated to vehicle mobility tests and in a real junction on the street. The basic infrastructure of the testbed for the Use Case (as shown by Figure 3.3) consist of:

- At least 1 Near edge COGNIT nodes;
- 38 far edge M-Hub (M-Hub, currently in process of deploying) acting as the client and users of COGNIT;
- On-premise data center in Control Center (CC) running Saturno platform;
- Optionally, the Saturno platform may be deployed on a Public Cloud.

The number of COGNIT nodes could be scaled up within a virtualization environment in the on-premises installation. The M-Hub clients could be scaled up to thousands of distributed nodes to achieve.

The data injected by public buses and emergency vehicles will be sent to RSU which is integrated within our M-Hub. This communication is done through two different interfaces, namely, ITS-G5 (ETSI Standards) and C-V2X (3GGP Standards), wherein these interfaces are standardised by ETSI and 3GPP well-known standardisation associations in telecommunications and communications domain. Therefore, this Use Case has contributed towards achieving the implementation of a standard in the field of Edge Computing**.**

Hardware

The hardware elements necessary for the validation and demonstration of the use case:

- OBUs implementingETSI C-ITS connectivity (optionally also C-V2X) and GNSS receiver. 4G connectivity is also recommended for maintenance and configuration..
- Human-Machine Interface (HMI) connected to the OBU in which the relevant information related to the Use Case is displayed.
- RSUs installed at every intersection under test implementing ETSI C-ITS connectivity (optionally also C-V2X) and GNSS receiver. 4G connectivity is also recommended for maintenance and configuration.
- M-Hubs (COGNIT's client) integrated with the RSU, and connected via ethernet or 4G to Saturno (ACISA's Mobility platform), which is deployed either in the City's Control Centre or on ACISA's private cloud.

Software

The software elements necessary for the validation and demonstration of the use case:

- Commercial software embedded in OBU capable of sending standardised V2X messages.

- Commercial software application in RSU capable of receiving standardised V2X messages, processing them and sending them to the priority management service deployed in the M-Hub.
- New components in M-Hub to communicate with RSUs, send FaaS requests to COGNIT, and implement the mechanisms to activate priority changes in the intersection.
- New application implemented as Faas to be executed into the COGNIT's nodes. It will receive requests from M-Hubs (COGNIT's client), execute traffic simulations, and return results back to the M-Hubs.
- Saturno, ACISA's Mobility Platform deployed as a SaaS service.
- Communications systems for connection of M-Hub to control center.

3.4.1 Integration with the COGNIT Framework

The following diagram in Figure 3.23 illustrates the integration of the use case architecture with the COGNIT Framework. In this architecture, the priority evaluation functionality is executed by serverless functions provisioned through the COGNIT Framework and deployed on edge cluster nodes, which are installed at the customer's far-edge premises.

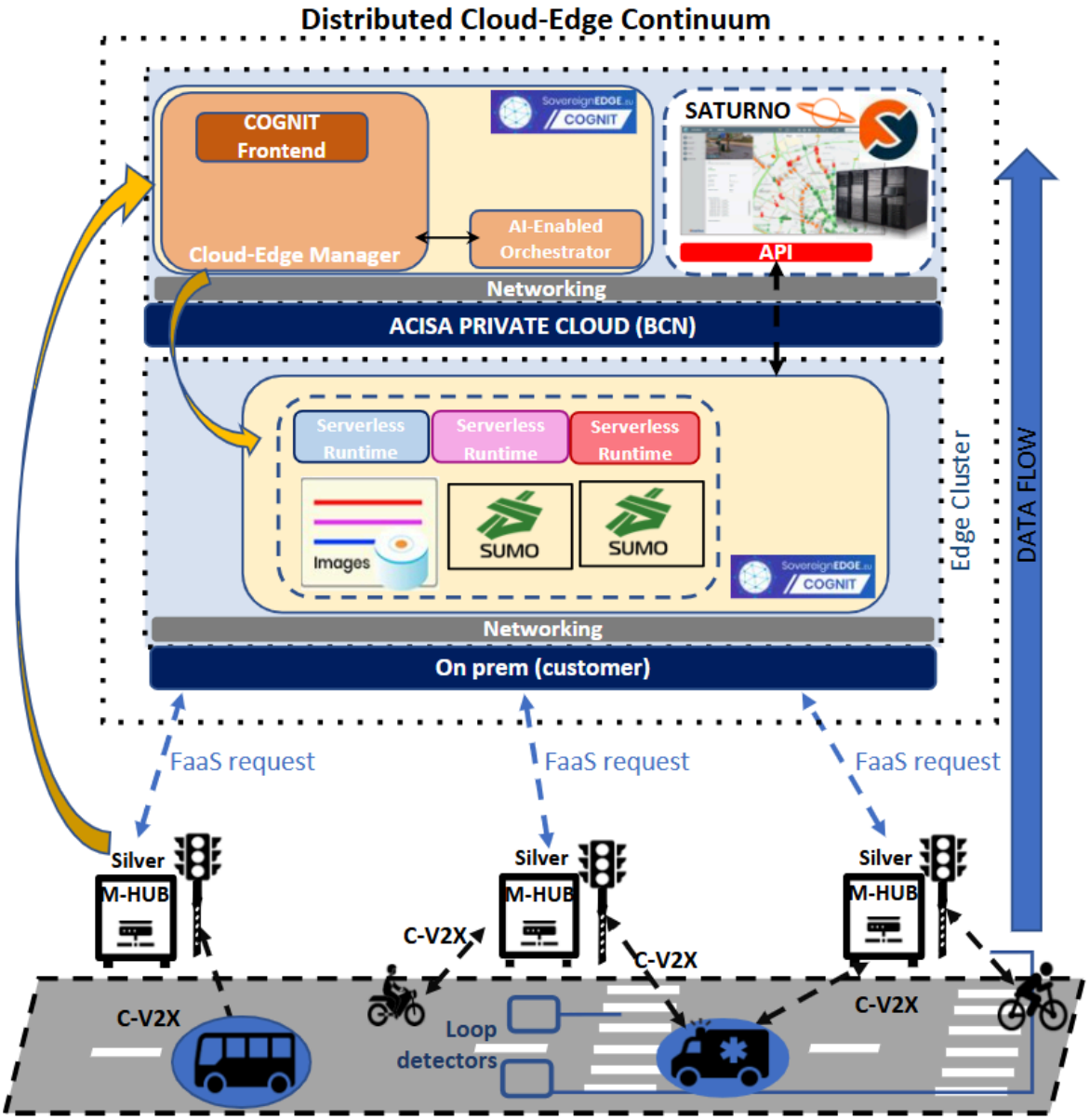


Figure 3.23. Overview of elements involved in the Smart City use case

Application Structure

Upon receiving a priority request from an authorised vehicle, the M-Hub may initiate a function call to the COGNIT FaaS service including additional contextual information to execute on-demand traffic simulation using SUMO.[47] This is some heavyweight functionality that is not possible (wouldn't make sense) to deploy on the M-Hub. Instead a FaaS request is made to remotely execute the needed simulations and obtain the results, which allows the M-Hub to make the decision to either grant or deny the requested priority. It is a function to manage a priority and needs to be executed quickly. This

[47] Simulation of Urban Mobility, an open source, highly portable, microscopic and continuous multi-modal traffic simulation package designed to handle large networks: https://eclipse.dev/sumo/

requires setting up the Serverless Runtime image with the libraries and dependencies for the simulation, the models of the junctions, and some scripts to process the outcomes, so that these elements are already available when the remote call is made, because doing this on demand for a request would be really inefficient. The context information necessary to evaluate the priority requested is available at the persistent data service in the COGNIT Framework. This information is captured by the intersection Digital Twin in Saturno, and there is a process of offloading that information to the DaaS component.

Once the priority analysis is done and communicated back to the M-Hub, it generates a response to be transmitted to the requesting vehicle. This response takes the form of another standardised V2X message known as the Signal Request Status Extended Message (SSEM), informing whether the priority has been conceded or rejected.

Our use case is focused on leveraging the capabilities of remote execution of complex simulations on the COGNIT Framework. This will provide our edge systems with valuable information to make decisions about how to deal with specific traffic situations.

Response time is an important issue because the M-Hub will receive a priority request for an upcoming bus, and there is a time constraint to make a decision. Our initial requirement was a latency not higher than 1 second introduced by the new FaaS mechanism (this does not include simulation time). According to our tests this requirement is being met.

Small to medium-sized cities such as Córdoba, Terrassa, Badalona, or Chiclana de la Frontera are increasingly adopting *“as-a-service”* solutions to overcome limited local IT resources. For these municipalities, flexibility in choosing the infrastructure on which to run their serverless FaaS (Function-as-a-Service) is an important requirement. Those cities are gradually opting for managed service providers or even public cloud deployments, benefiting from reduced operational complexity, such as Barcelona, with stricter regulatory and compliance obligations, still prefer on-premise deployments.

Being able to seamlessly select the specific location (i.e. edge nodes) where the FaaS will be run is a clear advantage of using the COGNIT Framework instead of other more ubiquitous cloud approaches from cloud services vendors. This will require the definition of policies specifying if there is a requirement about where the FaaS can be run, or where the data in DaaS can be stored.

A medium-size city in Spain may have 200 intersections on average, we may consider a maximum rate of 10 priority requests per hour per intersection, that is an upper limit of 2000 requests per hour in an average medium city. We estimate an upper limit of 20 concurrent requests by the different M-Hub clients.

COGNIT will help ensure Serverless Runtimes will be available to run SUMO simulations 24/7.

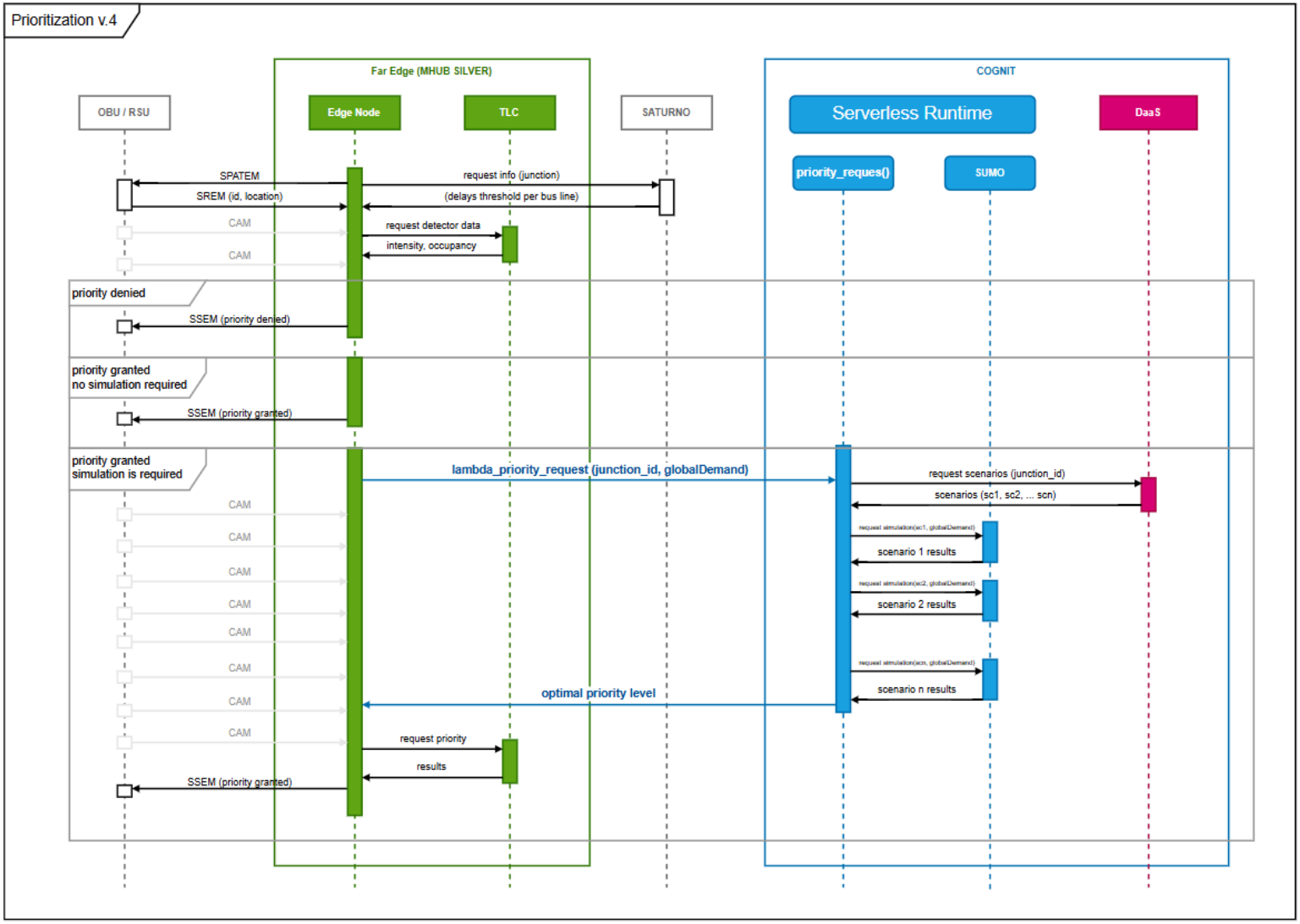


Figure 3.24. Overview of the prioritisation process, in which the M-Hubs makes FaaS requests to the Edge Cluster Frontend.

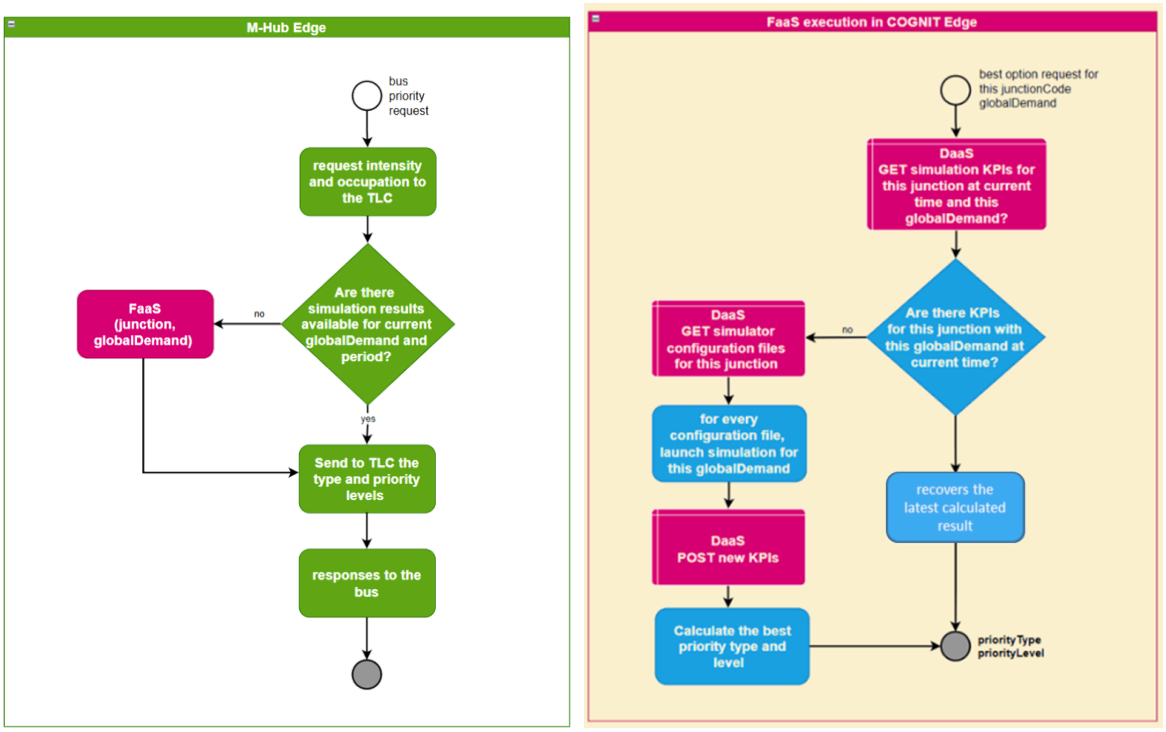


Figure 3.25. Details of the process in M-Hub and FaaS request.

Vehicle communication is facilitated by an OBU equipped with V2X capabilities and a Global Navigation Satellite System (GNSS) receiver. The OBU communicates via radio with V2X (RSUs) installed at every intersection, and these RSUs are connected to the M-Hub.

3.4.2 Edge compute node deployment

The integration with the COGNIT Framework with respect to the local Barcelona cluster has involved the following tasks:

- Deployment of the KVM hypervisor on the compute node of the Edge Cluster.
- Integration with VPN of Testbed Environment at RISE ICE.
- Enable cross-authentication between the Cloud-Edge Manager and the new Edge Cluster.
- Integration of the Edge Cluster with the Cloud Edge Manager.
- Creation of a cluster and a virtual network in the Cloud Edge Manager.
- Configuration of Service and VM templates for the Serverless Runtime.
- Deployment of the COGNIT Edge Cluster Frontend.
- Networking configurations to enable two-way communication between the Edge Cluster node and the VMs, access of VMs to the Internet to allow deployment of libraries and dependencies, and inbound connection from a public IP to the VM running the COGNIT Edge Cluster Frontend in the Edge Cluster.

3.4.3 Use Case Internal Testbed

The Use Case has set up a testbed in Granada for testing the integration of the real device in the field, with V2X sensors to receive buses or emergency vehicle detections, and running a simplistic algorithm to make decisions about priority. This algorithm will be replaced with a more elaborated one using COGNIT FaaS requests to run simulations in the upcoming cycles.

Furthermore, the following hardware has been acquired and is being deployed in Barcelona to provide local processing power to the pilot. In the current development cycle, one of the nodes has been used to deploy a COGNIT Edge Cluster Frontend:

1x Supermicro Twin rackable SuperServer SYS-120TP-DTTR Chassis 2U

Representing two nodes, each one with these characteristics:

- 2 x CPU Intel Xeon Silver 4314 16C/32T 2.4GHz.
- 128GB RAM: 8 modules x16GB DDR4-3200.
- 2 x Discs SSD de 960GB <2DWPD.
- 2 x Ports 10Gb Base-T @ motherboard.
- 2 x Ports 1Gb Base-T @ an additional board.

1x Rackable server 2U supermicro

- 2 x CPUs Intel Xeon Silver ICX 4314 16C/32T 2.4GHz.
- 128GB RAM: 8 modules x 16GB DDR4-3200.
- 2 x Discs SSD de 480GB SATA Intel D3 S4520 < 2DWPD.
- 2 x Ports 10Gb Base-T @ motherboard.
- NVIDIA QuadroRTXA5000 24GB GDDR6.

Saturno platform, previously deployed on outdated servers, has been migrated to a new cluster to ensure its integrity and availability:

4x Rackable DELL server DELL PowerEdge R660

- 2 x CPU Intel Xeon Silver 4416 20C/40T 2.0GHz.
- 512GB RAM: 16 modules x32GB 5600MT.
- 2 x Discs SSD de 480GB.
- 2 x Ports 10Gb Base-T @ motherboard.
- 2 x Ports 1Gb Base-T @ an additional board

Figure 3.26 shows the devices installed in each intersection of the Granada pilot:

Figure 3.26. Devices deployed on the pilot in Granada.

ACISA is the current traffic management contractor in the city of Granada. Figure 3.27 illustrates a map of UC1 pilot testbed, which utilized the infrastructure of Granada's public transport Line 4. This system comprises 114 intersections managed by 38 M-Hubs and 38 RSUs, along with 20 public buses equipped with V2X technology:

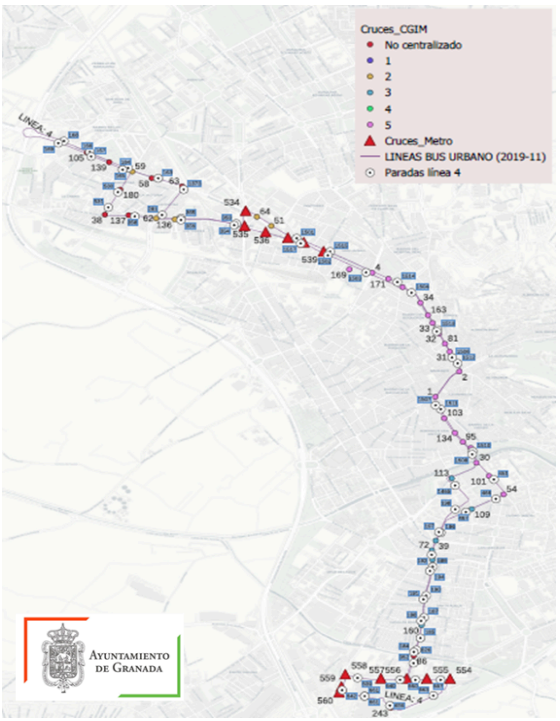


Figure 3.27. Map of Granada's public transport Line 4, including intersections and bus stops.

The current status is that:

- Every intersection is configured in Saturno, following the V2X standard.
- The basic configuration includes all incoming and outgoing lanes and the allowed movements between them.

- This configuration is automatically sent to the RSU through the Mobility Hub Edge.
- This information will be available to all authorised V2X vehicles.

In this project we have been testing the integration of the M-Hub with the V2X RSUs, and applying the algorithms needed to provide priority service in the following context:

- Line 4, consisting of a total of 38 traffic light controllers, including an RSU and an EDGE internal CPU, and 18 public buses which were equipped with the necessary OBU and antennas.
- Bus delay data for each bus is updated in Saturno every 5 minutes.

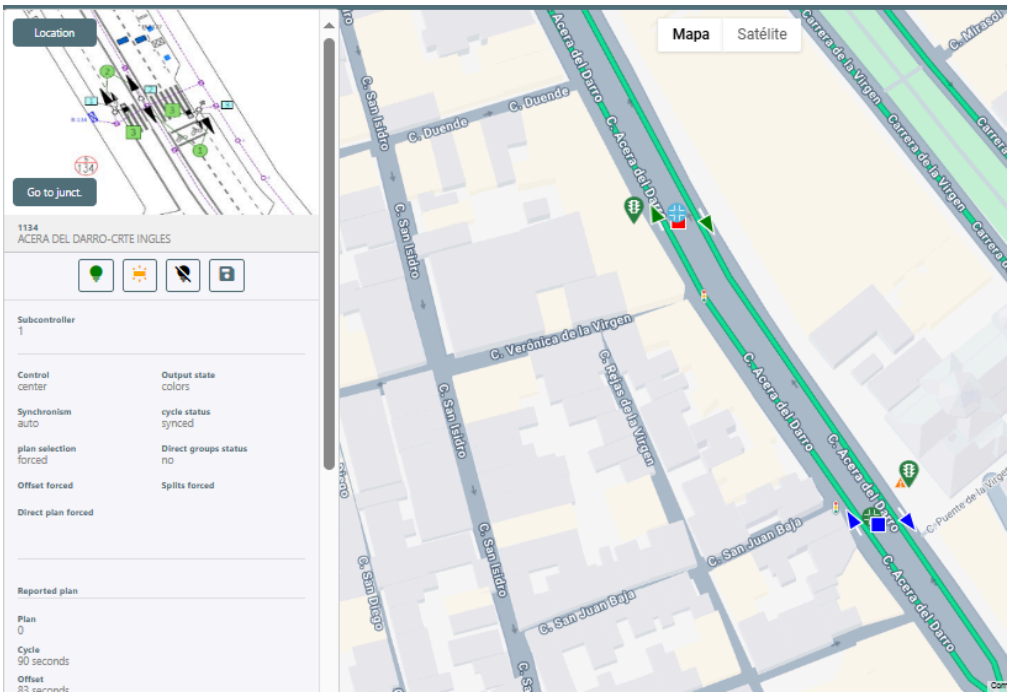


Figure 3.28. Detail of intersections used as use case 1 test bed.

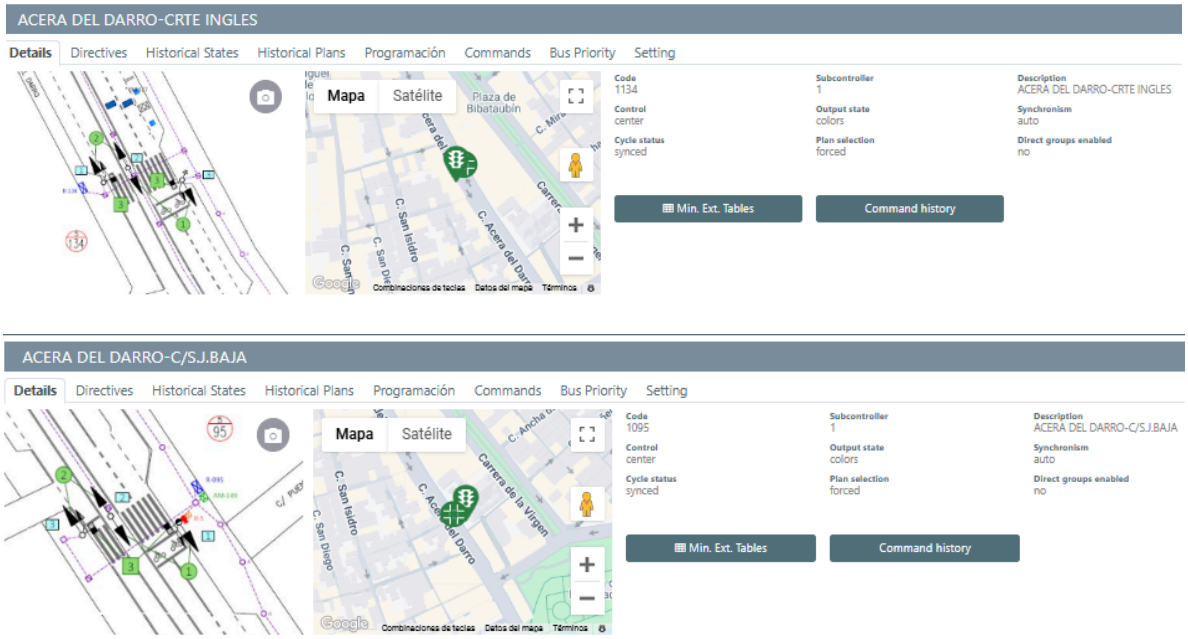


Figure 3.29. Granada's intersection 1134 and 1095 model as represented in Saturno.

3.4.4 Demo scenarios

Note: we are using a specification based upon **Behavioural Driven Development** (BDD)

A description of each specific scenario of the narrative with the following structure:

Given: the initial context at the beginning of the scenario, in one or more clauses;

When: the event that triggers the scenario;

Then: the expected outcome, in one or more clauses.

Source: https://en.wikipedia.org/wiki/Behavior-driven_development

Figure 3.30. Summary of BDD

We are considering the following scenarios:

Scenario - 1

Given I configure the environment for a flavour Smartcity.

When a FaaS request is sent from a PC.

Then, a response of performance metrics for priority is received at the device client.

And the FaaS is run in the ICE Edge Cluster.

Scenario - 2

Given I upload simulation results data to the DaaS belonging to a certain range of time.

When a FaaS request is sent from a PC that involves simulations of that range of time.

Then, a response of KPI for priority is received at the Device Client.

And the FaaS is using the result data available in the DaaS.

Scenario - 3

Given I upload simulation results data to the DaaS belonging to a certain range of time.

When a FaaS request is sent from a PC that involves simulations of a range of time not available in the DaaS.

Then, a response of KPI for priority is received at the device client.

And the FaaS is running the simulation for that range of time.

And the result of the simulation for this range of time is stored on the DaaS.

Scenario - 4

Given I configure a M-Hub environment for a flavour Smartcity.

When a bus approaches a detector.

Then, a FaaS request is sent.

And the FaaS is run in one Edge Cluster.

And a response of KPI for priority is received at the M-Hub Edge.

Validation criteria

- C-ITS Service “Traffic Signal Priority” (TSP) is tested in the pilot intersections, and basic functionality is validated in Granada.
- A junction is modelled as a Digital Twin, and is ready to run simulations and obtain results in the form of KPIs.
    - TSP, enhanced with a Digital Twin model is tested and basic functionality is validated. Edge node is able to launch the requested simulations by the FaaS.

Current Project Status: Validation Complete

The project has successfully met all defined validation criteria. As the preceding sections have thoroughly documented the work performed and the underlying infrastructure, this section will focus exclusively on the functionality that has been rigorously tested and validated.

Key Validated Capabilities

- The M-Hub deployed on field in Granada is receiving V2X events when a bus is detected by the sensors.
- The M-Hub is sending requests to the COGNIT Serverless Runtime using the COGNIT Device Client.
- The request is initially received by the COGNIT Frontend and derived to the corresponding compute node in an edge cluster. This functionality is out of the scope of this use case, as is responsibility of the WP2.
- The compute node executes the FaaS requested by the M-Hub. This involves running the SUMO simulation, getting the results, parsing and aggregating them, and returning a recommendation about granting or not the priority.
- The FaaS is leveraging the persistence provided by the DaaS in several aspects:
    - Fast access to the congestion state of the junction: Saturno is frequently updating this state in the DaaS and the Faas has this data available as needed.
    - Using previously calculated simulation results: When input the conditions are the same, the simulation would probably get quite similar results. Then the system is keeping the results of the simulations so that they can be reused when needed or desired.
    - To achieve this, the simulation results need to be stored in the DaaS when the simulation is run.

  This functionality would reduce the demand of the execution of simulations when similar results are expected.

- The FaaS is returned the priority grant or deny response, which is passed to the M-Hub so that it is taken into account for the traffic management.
- To be completed: Validation of the deployment of the edge cluster and the execution there of FaaS.

3.5 Technology Readiness outlook and conclusion

ACISA has a long-standing experience in traffic management systems, enabling centralized and coordinated control of field traffic controllers. Building on this expertise, we incrementally introduced innovative technologies, culminating in the deployment of V2X communications between public vehicles and their integration into traffic management workflows. Initially, we captured raw data from detectors and performed isolated tests, including a digital twin simulation, which highlighted scalability limitations for large-scale deployment. Through the COGNIT project, we advanced to TRL 5 by implementing the TSP service in controlled real-world scenarios and deploying Function-as-a-Service (FaaS) and edge nodes within an operational traffic management environment. This allowed us to validate a scalable, adaptive system capable of offloading computationally heavy workloads from traffic controllers while considering efficiency, energy consumption, latency, and geolocation. The real deployment in Granada strongly suggests that the solution is robust, resilient, and close to market readiness.

To further advance the TRL, ACISA has submitted a proposal under Horizon Europe[48]. The project aims to scale COGNIT to a full-city environment, addressing remaining challenges such as large-scale resource management, integration of AI based solutions, and optimization of traffic flow for multiple vehicle types. Successful implementation will increase confidence in city-wide deployment, strengthen interoperability, and enhance system efficiency, providing strong motivation for future EC funding.

COGNIT has significantly benefited the use case by providing an adaptive, scalable, and energy-efficient traffic management use case.. It added value by enabling real-time V2X data integration, offloading computational workloads, and improving the management of complex computational infrastructure. The developments achieved can be further enhanced through city-scale deployment, integration with additional smart mobility services, and potential adaptation to other industry verticals, such as logistics, emergency services, and public transport management. The project demonstrates how advanced data-driven traffic solutions can be transferred and applied across different urban mobility contexts, unlocking new opportunities for innovation and service optimization.

[48] Call HORIZON-CL4-2025-04-DATA-02 (OMNI AI)

4. Use Case #2: Wildfire Detection

Across Earth's ecosystems, wildfires are growing in intensity and spreading in range. From Australia to Canada, the United States to China, across Europe and the Amazon, wildfires are wreaking havoc on the environment, wildlife, human health, and infrastructure. The costs in human lives and livelihoods can be counted in the number who perish in the flames, or contract respiratory diseases from the toxic smoke, or in the number of towns, homes, businesses, and communities affected by fire. Not only can wildfires reduce biodiversity, but they also contribute to a climate change feedback loop by emitting huge quantities of greenhouse gases into the atmosphere, spurring more warming, more drying, and more burning.

More than 60000 forest fires occur every year in Europe alone, for a total estimated loss of 2 billion euros and between 1 and 2 million hectares[49]. Approximately 96% of all wildfires within the European Union are attributed to human actions[50]. The European crisis is, therefore, fundamentally rooted in land management deficiencies and human negligence.

Climate change functions primarily as an amplifier, increasing the risk and extent of fires by creating warmer, drier conditions that intensify drought. This climatic shift effectively lowers the threshold for an ignition to grow into a large, destructive event, exacerbating the impact of the prevalent human-caused sparks. European trends demonstrate a lengthening of wildfire seasons, with fire-prone areas expanding beyond the traditional Mediterranean regions into Central and Northern Europe. Furthermore, there is an increasing number of days categorized as "extreme" fire danger, thus wildfire will be a challenge the entire world and Europe in particular will surely face in the following years.

While prevention mitigates long-term risk, operational success in fire suppression is determined by the speed and efficacy of the initial response. The ability to contain a wildfire successfully depends critically on the response taken during the first few moments of the incident. The first crew arriving at a fire must be efficient in their initial efforts to stop the wildfire below the targeted acreage (often ten acres). The first intervention is so pivotal that fires contained during this phase contribute minimally to the overall burned area. Empirical studies consistently demonstrate a direct and critical relationship between delayed response and escalated damages. Longer fire department response times are associated with an increase in total burned area and a higher probability of extreme flame damage.

Analysis shows that while prompt intervention is necessary to minimize fatalities, for minimizing the probability of total property loss, **each minute becomes more critical at higher response times**. This finding is explained by the exponential growth curve inherent in fire behavior. When a response is already delayed, an additional minute of delay allows the fire to grow disproportionately larger and more intense, crossing critical thresholds that fundamentally change the fire’s manageability.

[49]https://joint-research-centre.ec.europa.eu/projects-and-activities/natural-and-man-made-hazards/fires_en
[50]https://joint-research-centre.ec.europa.eu/jrc-news-and-updates/2023-among-five-worst-years-wildfires-europe-2024-provides-some-relief-2024-11-19_en

When it comes to fighting wildfires, technology has very clear limitations. This is because controlling wildfire behaviour is highly dependent on the prevailing weather and fuel conditions, and accessibility. It is often only a change in weather that can help bring a wildfire under control. Therefore, the limits and appropriateness of suppression strategies and tactics and the associated suppression resource types should be overcome by new IT and AI applications to:

- Reduce the latency time between fire detection and fire extinction chain of operations.
- Improve fire behaviour predictions and associated smokes and particulate dispersion.
- Improve the safety of firefighters during field operations.

Nature 4.0 develops IoT devices for monitoring environmental parameters with applications in the agricultural, forestry and marine fields, exploiting the most modern technologies. The application of IoT technologies to remote areas poses unique challenges involving needs for low power consumption, low-maintenance and often low-cost solutions. The company's flagship product is the TreeTalker® (patent no. 102019000013362 released on 07.21.2021), capable of measuring in real time the water consumption of trees, the growth of biomass (diameter) and the health status of the leaves through spectral indices and innovative processes (**figure 4.1**).

As a follow-up Nature 4.0 developed the TreeTalker Fire (TT-Fire) device for real-time monitoring of fire occurrence and behaviour predictions. The device has been successfully used by the Civil Protection agency of Puglia Region in Italy for their operations of fire prevention and suppression (figure 4.1).

Figure 4.1. The two images show the TreeTalker on the right and the TreeTalker Fire developed during the Ofidia2 project

The possibilities offered by the COGNIT framework allowed a redesign of the TreeTalker Fire to improve the existing capabilities and add new features, in particular to reduce false positives. The new device TT-Fire, shown in **figure 4.2**, is able to detect real-time flame events by using a novel sensor based on an IR camera, with a distance range of 100 m. At the same time, gaseous compounds (CO2, O3 and particulate PM1-2.5-10) are measured. The devices send the data via 4G connection, given the possibly large amount of information that should be exchanged during a relatively short amount of time. Real-time images are collected during fire occurrences to evaluate fire intensity and behavior. The

real-time collected data streams are activated in a cascade of events that could involve all wireless sensors of the network with a rapid increase of required computation resources. Such particular features require a new approach to cloud computing services and edge integration, which is the core of the COGNIT EU project.

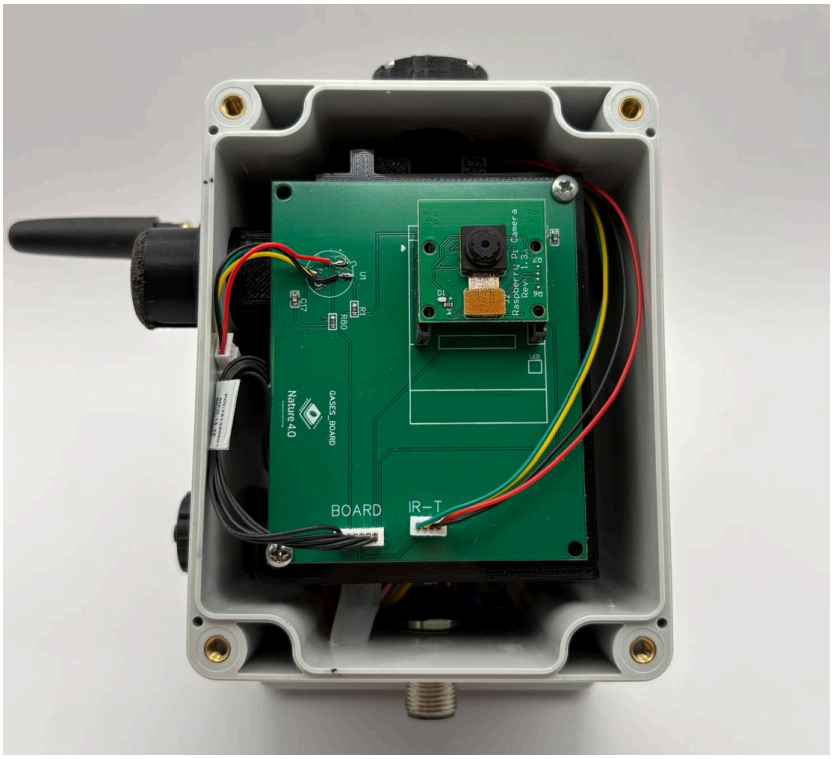


Figure 4.2. Image of the new version of the TreeTalker Fire developed during the COGNIT project

4.1. Reference Scenario

The wildfire detection use case explores the use of IoT technologies for fire detection in forests. The early detection of wildfire is of utmost importance to obtain a timely intervention from civil protection and firefighters to minimise damage in terms of forestry resources and human lives. However, developing and deploying a reliable sensor network in the forest is challenging due to the strict power constraints and the lack of strong connectivity, as well as the cost of maintenance in remote areas. Furthermore, a false alarm resulting in an unnecessary intervention also leads to an increase in costs.

For these reasons, TreeTalker Fire is being designed to collect data from multiple sensors to assess the presence of fire. The collected parameters and data are:

- Carbon dioxide concentration in air (ppm).
- Ozone concentration in air (ppm).
- Particulate matter PM10, PM2.5, PM1 (µg/m3).
- Air temperature (°C).
- Air relative humidity (%).
- Foliage temperature.

- IR camera images with a 32x24 resolution.
- RGB camera images.
- Geographic coordinates, which have been hardcoded in the device.

Sensors installed in forests can be used for the first detection of fire flames and combined with the TreeTalkers Cyber to collect useful metrics for wildfire risk assessment. The operating principle of the TreeTalker Fire device, which evolved during the COGNIT project, is to use all sensors except the RGB camera for a first assessment of the environment and to use the RGB camera as confirmation if needed. The gas sensors, as well as the measurements of air temperature and humidity, allow detection of a potential fire even if it is not in sight, while the infrared (IR) camera is a more direct sensor if the fire is in sight. The confirmation by the RGB camera provides a clear understanding of the situation to the civil protection agencies to help coordinate the intervention and reduce false positives. The access to images, together with the other parameters, helps tracking the spread of a wildfire. While for most of the sensors it is sufficient to set a trigger threshold, image recognition requires a suitable machine learning algorithm that exceeds the power and computing resource capabilities of IoT devices, thus it makes sense to offload it. The TreeTalker Fire is equipped with a 4G module to gain Internet access and be able to exchange information at a suitable speed. The TreeTalkers can be scattered in the forest while the TreeTalker fires are distributed along forest paths and forest borders, as well as recreation areas, to account for the 96% of fire outbreaks attributed to human actions (as referenced in the introduction above).

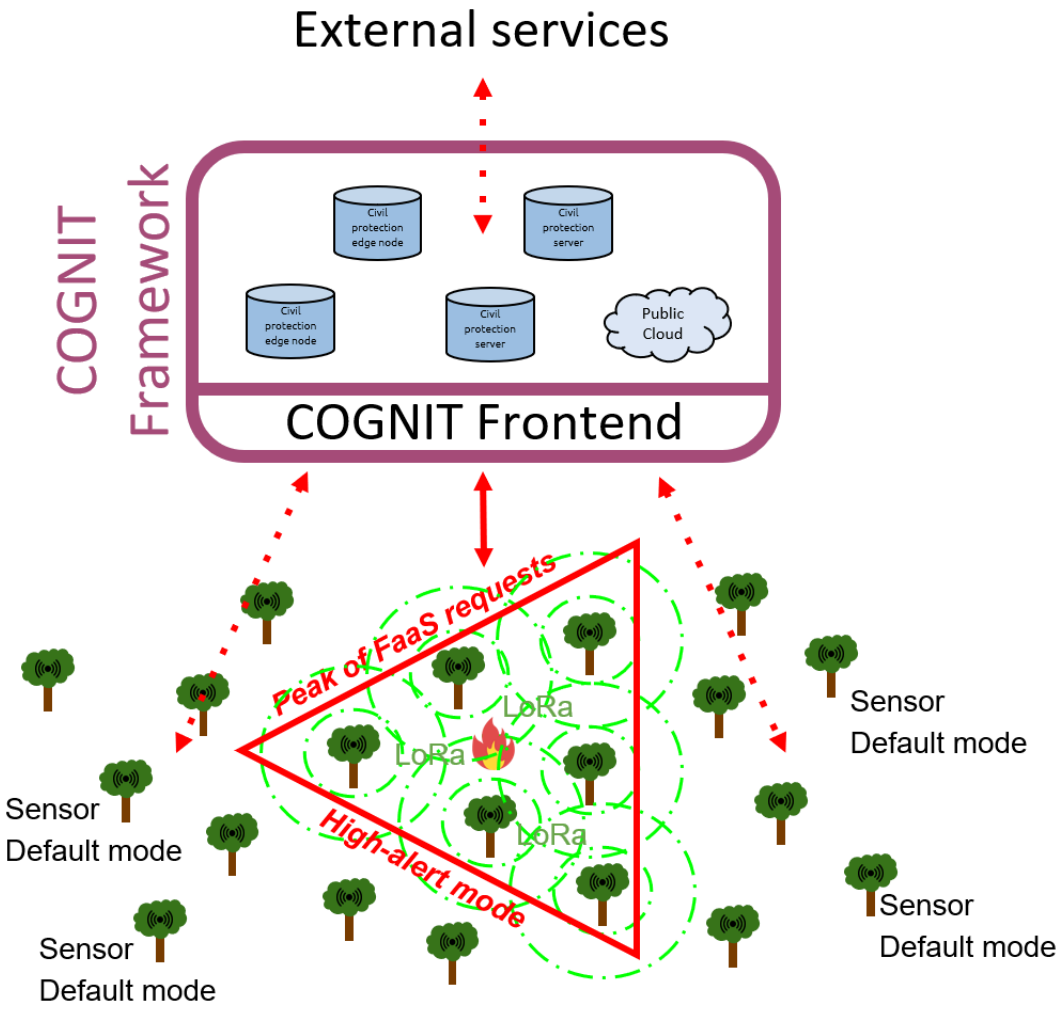


Figure 4.3. Schematization of wildfire use case scenario

The device routine consists of monitoring its surroundings every hour with all the sensors except the camera in order to save energy. In most cases no flames will be detected, and the device will just send the data to an external database. Collected data are combined with the ones from TreeTalkers for further environmental analysis and for fire risk assessment. The described behaviour represents the “Default mode” representing a preemptive approach to wildfires. When the collected parameters may indicate the presence of a flame, the “High alert mode” is triggered. In this mode, the camera is turned on, and the measurement frequency is increased to one measurement per minute. In “High alert mode”, a FaaS request is sent to COGNIT for each measurement to run the image recognition algorithm and confirm the presence of a flame. In case the presence of a flame is confirmed, the data is sent to civil protection. The FaaS requests are performed until 10 consecutive requests return a negative answer, after which the device exits the “High alert mode” and will make no further requests for a given period to avoid unnecessary resource utilization in case of false positives.

When a device enters the “High alert mode”, and every time a FaaS request returns a positive result, a distress signal is sent to the nearby devices using LoRa technology. It is a LPWAN (Low Power Wide Area Network) technology with a range up to 15 km line of sight and widely tested with the TreeTalker devices. All devices within reach wake up and collect data, including camera images, and trigger a FaaS request to COGNIT. If the request returns a positive result (a flame has been detected), the device enters “High alert mode”, otherwise it returns to sleep.

When there is presence of fire, the number of devices involved and the subsequent number of FaaS requests can rapidly increase, asking for a fast response to the sudden peak of requests. Moreover, despite the implementation of wildfire risk assessment, the actual emergency can hardly be forecasted, requiring high scalability and adaptability.

For all these reasons, allocating a given number of resources based on the maximum capabilities requested by the application would be inefficient, causing a waste of computational resources most of the time. However, all the resources required must be timely available when needed to minimize the damage of wildfire events. COGNIT can manage the resources, making them available when needed, allowing for efficient management and an effective reaction to request spikes, reducing the response time. The machine learning model, along with all the required libraries for the function execution, is saved into the VM image associated with the service. This approach aims to accelerate execution and conserve bandwidth. The captured image, properly resized, is the argument of the machine learning function; resizing the image on the device allows saving bandwidth.

Our use case aims to demonstrate the capability of the COGNIT Framework to respond to an unforeseen sudden peak of requests and to provide sufficient hardware resources when required by the application.

Figure 4.4 below shows an example of a possible wildfire prevention network and all its possible components. In forests, Internet connections can be unavailable or unreliable. If the area is well covered by a public LTE network, a 4G connection can be used by the devices; otherwise, one possible solution for providing connectivity is a private LTE network. The target market for TreeTalker Cyber Fires comprises public institutions seeking a digital solution to facilitate forest management and preservation; in addition to providing the devices with a reliable and fast connection for image transfer and supporting fire control efforts, a private LTE network could benefit them in case of emergencies, such as finding missing people in the forest.

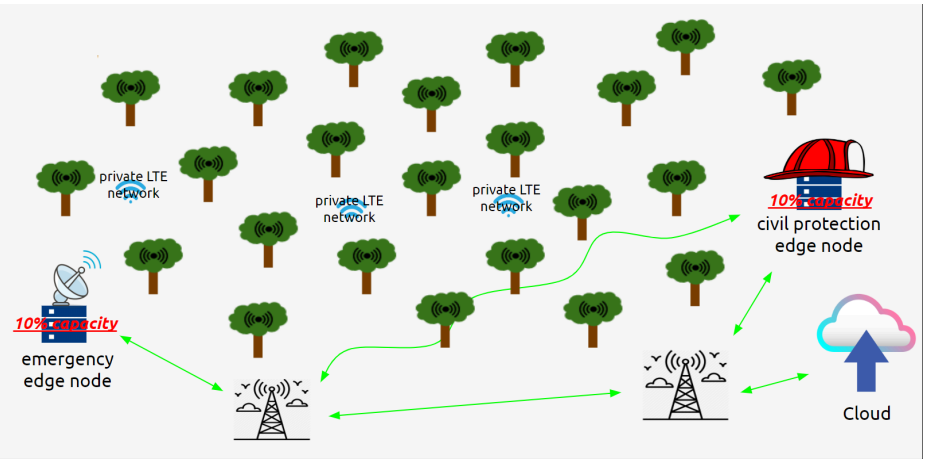


Figure 4.4. Example of a possible implementation of the wildfire early detection network

A private LTE network also increases the reliability of the entire network in case of weak Internet signals. During a fire, the signal strength could decrease due to interference or an increase in data transferred, but also as a result of fire damage to the infrastructure itself. If the connection to the public cloud is severed, a private LTE connection will keep the device network functional.

For reliability reasons, some edge nodes could be deployed at strategic points to allow data analysis at the edge. One of these edge nodes could be deployed in the local office of the forestry corps, while others could be added depending on the forest structure. Edge nodes can be equipped with satellite connections to allow information to be sent even when a public LTE network is not available.

The described configuration not only increases the overall reliability of the system, but the possibility of performing data processing locally also decreases the application response time and the bandwidth required by the system from the public network, particularly during a wildfire in remote areas where network resources are already scarce.

Reducing the volume of data transfer is especially beneficial for satellite connections, as it cuts both transfer time and costs. While the number of edge resources depends on network size, we suggest deploying enough capacity to support devices entering 'high alert' mode within an hour; this should be calculated based on average device distribution and wildfire spreading speeds.. Increasing the hardware resources and the number of edge nodes improves the performance of the network, but also increases the costs; moreover, the edge resources are under-exploited during normal operation in the default mode. The COGNIT Platform manages the compute resources and offloading requests of the described device network, maximising its effectiveness and timely providing resources for image recognition, where and when needed, reducing the overall costs.

The working assumptions regarding the constraints for the function execution mainly concern latency for the wildfire application, in which timely intervention is essential.

4.1.1 Unique Challenges

Implementation of the scenario described above presents a number of unique challenges and novelties, including:

- The devices and communication gateways are located in highly remote areas, which means that minimising maintenance and energy use are highly prioritised.
- Due to energy conservation measures, the devices will rarely make offloading requests during normal/default operation; however, in the unpredictable event of a suspected fire, many nearby devices could potentially wake up within a very short amount of time, causing a sharp peak of heavier-than-usual FaaS offloading requests to the edge.
- To successfully provide fire containment, all needed resources must be made available and managed in an effective manner, to assure the best response possible to the threat, by the responsible organization (reliability).

4.1.2 Future opportunities

Nature 4.0 works on several devices involving environmental monitoring and the COGNIT project provides several features that could be leveraged by Nature 4.0 future projects.

The lightweight C-client developed under WP3 is a feature of COGNIT that is absolutely needed to apply FaaS technology and cloud-edge continuum management in our field of work and is not provided by most of its competitors. While the final version of TreeTalker Fire has enough memory to run the Python client, several of our devices could not integrate FaaS services due to lack of computational resources. Moreover, we continuously iterate on our devices to optimize the needed microprocessor resources to subsequently reduce the power consumption and microprocessor costs, as well as the costs associated with battery changes in remote environments by increasing battery duration. Thus, the COGNIT Framework marks a significant step in bringing cloud edge continuum and FaaS technology to IoT devices.

The described features represent the prerequisites for the use of a FaaS service by IoT devices, but the management of the cloud-edge continuum could offer so much more for future projects than FaaS. Over the last year solutions to seamlessly connect LTE devices placed in remote areas by satellite connection have been released in selected countries. Nature 4.0 is thrilled by this new opportunity and is planning to experiment with it in some of the projects it is involved in, along with other satellite solutions it is testing. Moreover, the application of private LTE networks is expected to be a key opportunity for devices such as TreeTalker Fire in the near future, increasing the overall reliability of the system. It is easy to see how this technology integrates perfectly with the concept of cloud-edge continuum with edge resources connected to the private LTE network and the COGNIT Frontend accessible through the private network and a satellite access point, along with the standard network to increase system reliability.

Finally, the possibility to include specific constraints for the functions being offloaded to COGNIT could be implemented in our existing network and expanded with further options. The possibility to select the minimum energy consumption is truly interesting, and in future it could support some additional requirements such as minimum carbon footprint

or lower cost when consumption-based servers are included in the network to provide resources during peaks of requests or particularly challenging tasks.

The COGNIT Framework offers an interesting open source solution for the effective use of distributed resources and to provide FaaS service in owned servers without using externally managed, commercial solutions and shows even bigger potential to be applied in the future to coordinate the resources in the private-edge continuum that will be boosted by upcoming solutions.

4.2. Objectives and validation criteria

The key objective is to test how COGNIT deals with an unforeseen/sudden and overwhelming peak of requests. Specifically, in default mode the data are analysed and saved without particular constraints in terms of time or criticality. On the other hand, during a wildfire event, low-latency features and advanced data analysis capabilities become important—this scenario causes a rapid change in service level requirements. Moreover, a clever use of local and remote resources is required to ensure a prompt response during a high-intensity scenario such as wildfire containment.

One additional goal of UC2 during the COGNIT Project was the improvement of the TreeTalker fire device and the implementation and testing of different technologies in order to build the best possible device in terms of capabilities and reliability through the new possibilities available by the integration with the COGNIT Framework.

The main addition in this direction was the image recognition feature. The feasibility of performing image recognition directly on the device was assessed but ultimately rejected. This decision was driven by three primary technical limitations:

- **Power:** The processors required for the task demonstrated a high power draw.
- **Time:** The elaboration speed was deemed relatively slow.
- **Memory:** A significant memory footprint was required for concurrent image storage and main program execution.

Moreover, sending the image provides the operator with a more comprehensive understanding of the situation that helps the implementation of the most effective intervention strategy as the event evolves. The full utilization of cloud computing for image recognition makes COGNIT even more valuable for UC2, elevating the timely execution of the function and the smart monitoring and use of the cloud resources from auxiliary features to indispensable components of the system.

The validation criteria to demonstrate the successful integration of the COGNIT Framework with the new and improved version of the TreeTalker Fire device are:

- Identify and develop a lambda function that should be offloaded to the COGNIT cloud-edge.
- Integrate the COGNIT Client with existing applications, and install it on a TreeTalker Fire device.

- Demonstrate a TreeTalker Fire device offloading data analysis and image segmentation to the COGNIT Framework.
- Simulate a wildfire event on a virtual TreeTalker Fire network, based on a realistic deployment scenario, to test the ability of the COGNIT Framework to adapt to sudden peaks of FaaS requests.

The final goal to be achieved by the end of the project is to install a pilot deployment of a TreeTalker Fire network in a controlled, real environment near Nature 4.0 headquarters. The pilot will be composed of a batch of 20 TreeTalker Fire and TTCyber. The production of the devices is currently ongoing.

4.3 Summary of activities done during the project

The COGNIT Project was designed to be developed in cycles and thus, the use case activities were accompanied by a continuous effort to adapt and modify their application to support the new features and requirements introduced by the COGNIT Framework development.

UC2 followed three main branches:

- The development of a new version of the TreeTalker Fire.
- The programming of a virtual simulation for scale tests.
- The deployment of a local edge node to test the integration with the COGNIT framework described in section 4.4 of this document.

All three branches were updated during the project to reach the best possible solution.

4.3.1 Selection of the image recognition algorithms and integration with COGNIT

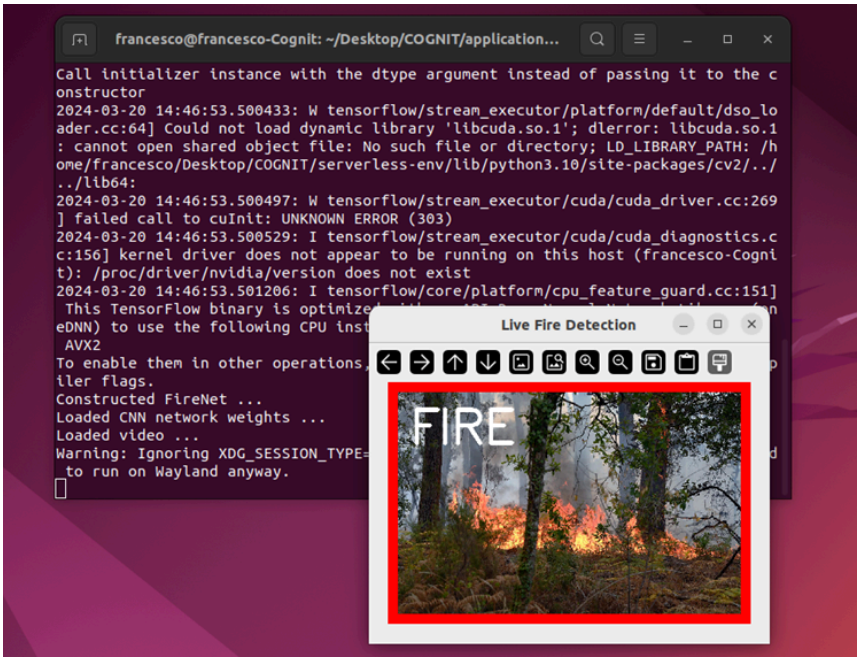


Figure 4.5: Execution of the fire image recognition function with visual rendering.

The first step in defining the UC2 application was to conduct a thorough research regarding pre-trained fire image recognition machine learning models to decide if it was necessary to train a model specifically for the application or if there was an applicable solution. The constraints were:

- It should be written in Python, being the main programming language for our applications as well as for the COGNIT Project.
- It should be released under a permissive license. This requirement allows the free use of the software for commercial purposes, the future improvement of the software and is fully aligned with the project's spirit.
- It should perform well in recognizing forest fires.
- It should be compatible, if possible, with TensorFlow Lite, to potentially allow it to run on edge devices.

Among the Machine Learning (ML) solutions available, a model based on the Inception V1 Convolutional Neural Network (CNN) was identified as meeting all stated requirements[51]. The supporting research systematically compared different CNN architectures to define a reduced-complexity version with minimal performance degradation. While the parent network was used for multi-class object recognition, the optimized version avoids reliance on temporal scene information (like flame flicker or motion), making it highly suitable for recognition of flames in static images. Moreover, for each machine learning algorithm two versions were provided; the one named "SP" was trained to localize the position of a flame inside the image while the second one was trained for binary classification, with a subsequent increase of accuracy. Following testing, the "InceptionV1-OnFire"[52] was chosen. Its architecture is reported in Figure 4.6.

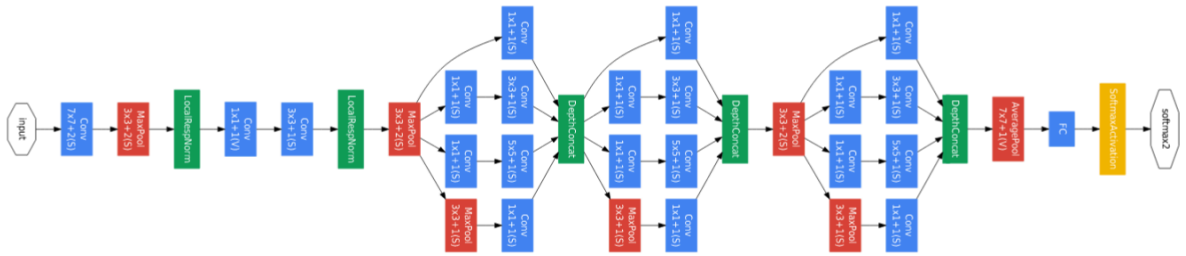


Figure 4.6. InceptionV1-OnFire convolutional neural network architecture

The reduced complexity compared to the original models allowed for a reduction in the execution time and the required computational resources. This feature is valuable for wildfire detection applications, in particular when the function must be run for multiple nodes at the same time.

[51] Experimentally defined Convolutional Neural Network Architecture Variants for Non-temporal Real-time Fire Detection (Dunnings, Breckon), In Proc. International Conference on Image Processing, IEEE, 2018.
[52] https://github.com/tobybreckon/fire-detection-cnn/tree/master

The original model was adapted to run in a single function that takes a list of integers as argument (the single image) and executes the model to return a boolean depending on the result or a string in case of errors in executing the function.

```python
def fire_presence_detection(im: list[list[list[int]]]) -> str | bool
```

Furthermore, the function was adapted to print every message to Linux's "/dev/null" because the virtual machine (based on the UC2 VM image) has no standard output.

```python
import sys
f = open('/dev/null','w')
sys.stdout = f
```

The nested list of int (rows, columns, and RGB values) is then converted into a numpy uint8 and the model's weights are loaded directly from a folder in the virtual machine. Finally, the result of the model is calculated and returned.

Another change performed was to reduce the libraries (dependencies) required by the ML algorithm to the bare minimum to make it as lightweight as possible. The software was tested on an Ubuntu virtual machine on different images to assess the performance. Compatibility issues with existing libraries were fixed to make it run, selecting the most recent compatible versions as requirements (tensorflow==2.8.0, Pillow==9.0.1, protobuf==3.12.4).

In order to run the function on a COGNIT serverless runtime a dedicated VM image for the Use Case was created. The virtual machine configuration was adapted to fulfill the use case requirements by increasing the default VM memory size to 3.072 GB and the disk size to 8 GB to allow the use of TensorFlow, moreover the VM template CPU mode was updated to "host-passthrough" to support SSE4.1 instructions. After the mentioned steps, the libGL library could be installed through the following commands:

```
zypper install Mesa-libGL1
zypper install libgthread-2_0-0
```

Finally, the directory containing the model's weights was uploaded to allow the function offloading. The directory structure used for the application is illustrated in Figure 4.7:

```
FireUC
├── requirements.txt
└── model
    ├── checkpoint
    ├── inceptiononv1onfire.data-00000-of-00001
    ├── inceptiononv1onfire.index
    └── inceptiononv1onfire.meta
```

Figure 4.7. Directory structure of uploaded files.

The full code, as well as all the requirements to set up the COGNIT image and run the image recognition function, are uploaded to the public "COGNIT Framework" GitHub under the "use-case-2" folder[53] which is the repository maintained by use-case-2 during the project. Further information about the software can be found in D5.9.

4.3.2 New version of the TreeTalker fire

The original TreeTalker Fire shown in **figure 4.8** was a device used in the Ofidia2[54] project for wildfire prevention and it was originally equipped with:

- Gas sensors to measure the concentration of O_3, CO_2 and PM $(2.5,\ 5\ and\ 10)$;
- TRH (air Temperature and Relative Humidity) sensor;
- Flame detection sensor (with a range of approximately 150 m);

Figure 4.8. Images of the original TreeTalker Fire

While the original configuration obtained some interesting results during the project, it lacked a way to discern false positives, in particular when triggered by the flame detection sensor, resulting in false alarms and potential deployment of civil protection resources to confirm the presence of a fire and their associated costs. The COGNIT project provided the perfect tools and opportunity to improve the device's capabilities by redesigning it using

[53] https://github.com/SovereignEdgeEU-COGNIT/use-case-2
[54] https://www.interregofidia.eu

the most recent technologies, the main one being the addition of a camera for fire presence confirmation. During the project different technologies and solutions have been tested in order to find the best in terms of capabilities and reliability.

Different solutions were tested to find the best combination:

- 3 NB-IoT and 4G modules (SIM7020, SIM7022 and A7268E).
- 2 camera models (esp32-CAM and Arducam) .
- 3 processors (STM32L433RCTx, esp32 and Raspberry Pi Zero).
- 2 temperature sensors (single point temperature sensor and MLX90640).
- different gas concentration sensors.

Most of the electronics required by each component have been designed internally in order to reduce the cost and obtain a tailored solution. Different versions of the device during its development cycle are shown below:

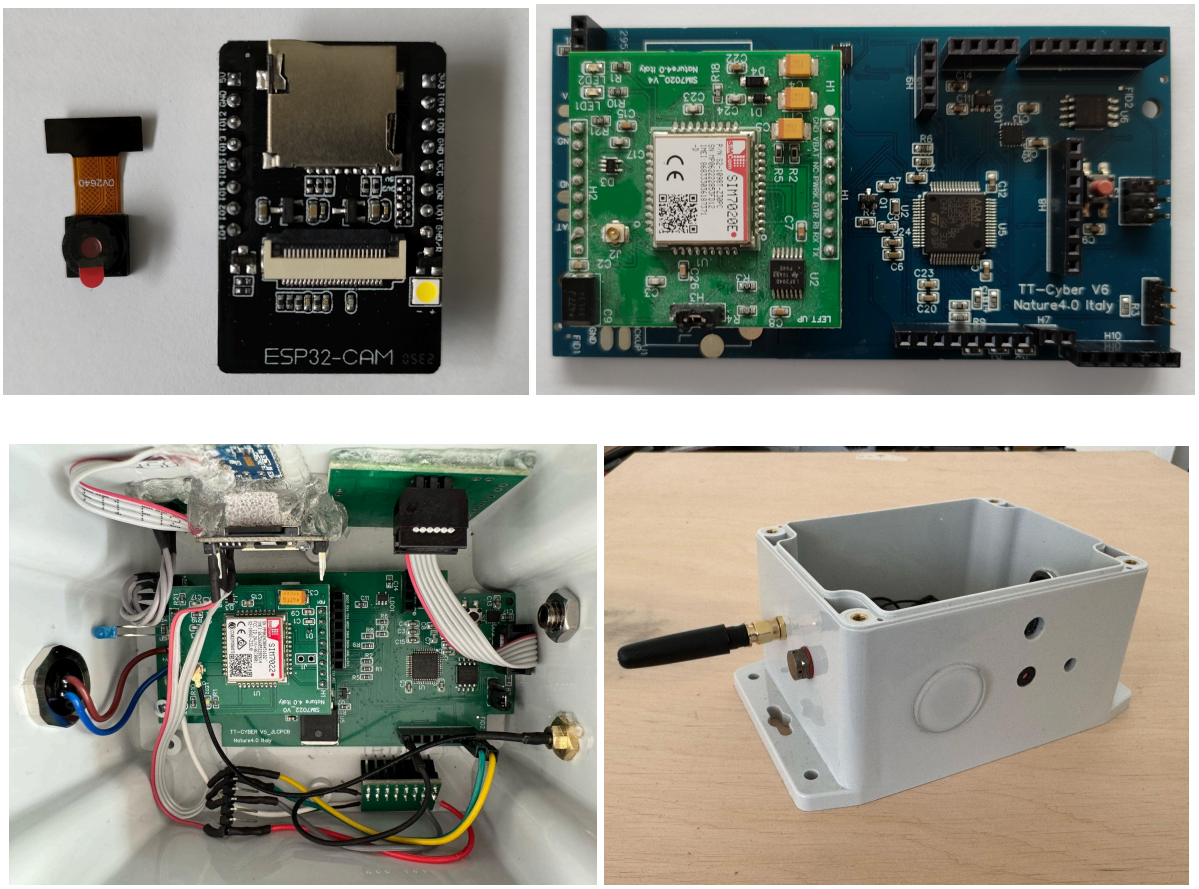


Figure 4.9. Images of different TreeTalker Fire prototypes

Every sensor and module was interfaced and tested and the selection of the definitive components was performed taking into account the performances as well as market considerations.

The final set up is the following:

- Carbon dioxide concentration in air (ppm).
- Ozone concentration in air (ppm).
- Particulate matter PM10, PM2.5, PM1 (µg/m3).
- Air temperature (°C).
- Air relative humidity (%).
- Foliage temperature.
- IR camera with a 32x24 resolution.
- RGB camera for image acquisition.
- Geographic coordinates hardcoded in the device.

Compared to the previous solutions, a foliage temperature sensor has been added at the top of the device to provide further information for the wildfire risk assessment and the flame sensor has been replaced by a IR camera matrix of 32x24. It allows discerning between the flame and the surroundings, thus providing valuable information compared to the flame sensor that returns a single average value for the entire field of view. Moreover, the RGB camera has been added to confirm the presence of a flame. The final device is shown in figure 4.10 is composed of 4 layers.

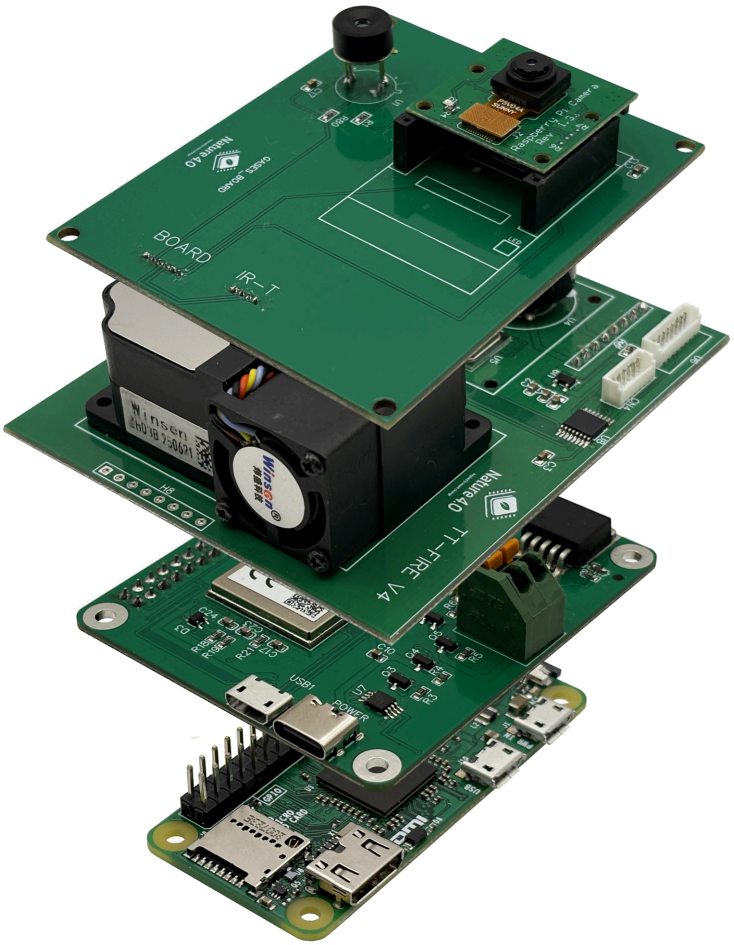


Figure 4.10. 3D exploded view of the printed circuit boards composing the TreeTalker FIre, from the top: the optical sensor board, the gas sensor board, the Internet module and the Raspberry Pi Zero

The upper one contains the optical part and is equipped with a 5 Mpx module for Raspberry Pi. The module was preferred over the ESP32-CAM for the higher resolution and the Raspberry Pi Zero has been chosen to handle the increased complexity and memory usage while maintaining the power consumption as low as possible. The other optical sensors equipped are the foliage temperature sensor placed on top of the device and the IR camera, exposed to the outside through a protected opening on the transparent lead to allow the infrared radiation to reach the sensor. The middle board contains the gas sensors and is connected with the outside by two channels: one larger lateral intake protected from rain for the particular matter sensor and a small tube that passes through the bottom side of the box. The particular matter sensor is equipped with a small vent that speeds up the air recirculation. The sensors are enclosed in a 3D printed volume to reduce the response time to reach the external gas concentration. The sensors have been selected by testing different solutions available on the market and taking into account the cost and the resolution. The gas concentration is primarily used as a threshold for flame detection and secondarily as an absolute measure of air pollution. The two sensor boards are connected by I^2C and UART to the Raspberry Pi Zero that provides the power and runs the main program for sensor acquisition and function offload. The Internet connection is provided by a custom made A7268E module. A custom-made RAK3172 module is used to switch on and off the Raspberry Pi to spare energy and to send and receive LoRa distress signals.

During default mode operation the device wakes up periodically and collects data from the sensors. The presence of fire is detected by comparing collected data with preset thresholds for gas sensors, and comparing the temperature of the IR sensor pixel representing the highest value with the threshold, as well as the percentual difference with the average value of the IR image pixels. If no flame is detected the device shuts down and waits for the next cycle. Otherwise, the high alert mode is triggered and a distress signal is sent by the RAK3172 to nearby devices. In the meantime a RGB image is collected by the device, it is reduced to a 224x224 format (the resolution used by the machine learning algorithm) and a FaaS request is sent to the COGNIT Framework to execute the function. The reduction in the size of the image device side helps to reduce the amount of data that must be sent with a subsequent reduction in send time and bandwidth. If a flame is detected, the high alert mode persists, the device remains active and sends a FaaS request every minute until 10 consecutive requests return a false result. A flowchart of the device logic is reported in **figure 4.11**.

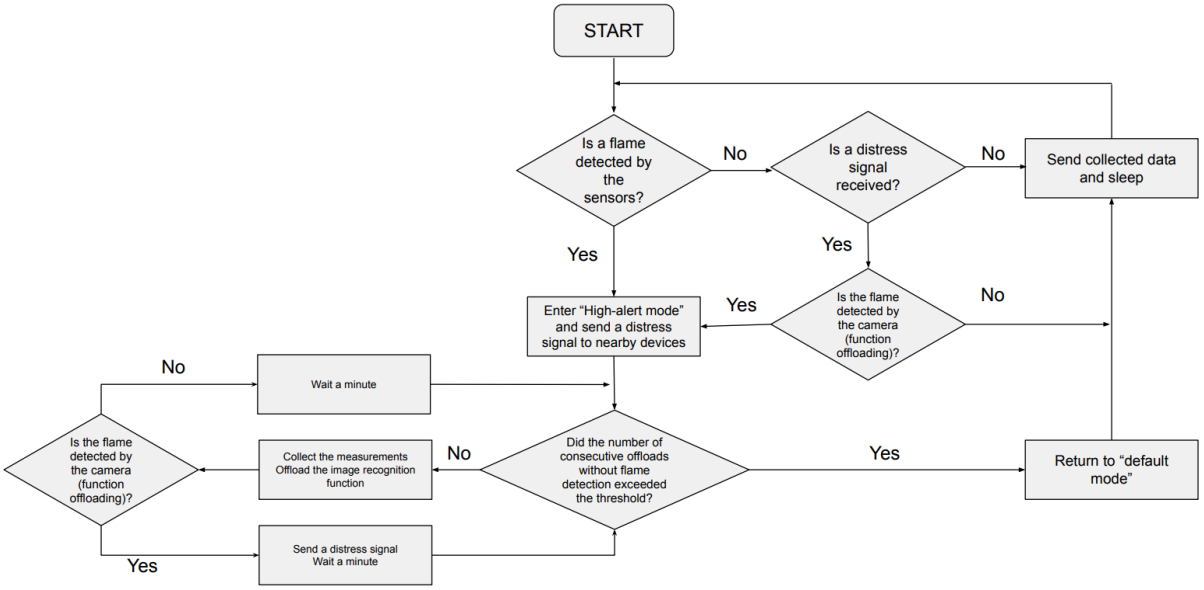


Figure 4.11. Flow chart of wildfire sensor network logic.

The described logic was written in Python, which is made possible with the Raspberry Pi Zero available as the main board. An example of the strings sent during the default mode is the following:

None

```
22-10-2025 10:08:41;9F25A018;19;74387;485;0;1000;1000;382;1961;7629;1768;3948
22-10-2025
10:08:44;9F25A018;17;1287;1289;1298;1191;1228;1280;1330;1207;1287;1420;1772;1692;1740;1745;1860;1732;1749;1748;1851
;1744;1735;1752;1807;1750;1718;1712;1783;1655;1628;1615;1723;1474;1360;1311;1238;1187;1199;1232;1343;1375;1403;1449
;1677;1717;1730;1729;1797;1784;1775;1762;1817;1799;1753;1745;1791;1787;1728;1716;1740;1729;1660;1655;1717;1637;1293
;1342;1466;1322;1243;1267;1378;1296;1426;1489;1706;1689;1696;1711;1840;1760;1766;1777;1864;1752;1762;1780;1842;1746
;1732;1741;1799;1686;1637;1697;1708;1592;1391;1278;1319;1318;1344;1280;1338;1351;1487;1486;1683;1720;1721;1681;1799
;1824;1774;1764;1803;1817;1794;1753;1796;1790;1749;1742;1757;1739;1736;1629;1687;1652;1312;1306;1388;1266;1284;1308
;1374;1261;1403;1492;1752;1684;1692;1726;1842;1768;1769;1796;1842;1778;1780;1800;1828;1776;1760;1758;1803;1712;1684
;1710;1769;1617;1406;1288;1410;1384;1319;1273;1336;1319;1461;1507;1738;1739;1719;1702;1794;1819;1802;1770;1801;1812
;1779;1789;1801;1799;1803;1737;1772;1750;1712;1671;1687;1707;1384;1385;1433;1314;1226;1265;1360;1280;1557;1650;1758
;1673;1711;1733;1840;1790;1771;1804;1848;1779;1778;1807;1820;1766;1770;1771;1821;1730;1672;1698;1763;1592;1432;1378
;1375;1309;1251;1247;1293;1323;1613;1658;1717;1716;1710;1725;1807;1813;1795;1778;1831;1810;1811;1763;1810;1819;1782
;1761;1789;1761;1710;1669;1710;1716;1377;1444;1451;1305;1272;1250;1311;1312;1592;1670;1758;1696;1697;1739;1855;1809
;1812;1789;1848;1782;1782;1795;1831;1750;1765;1790;1824;1717;1708;1715;1796;1654;1427;1435;1436;1434;1286;1258;1318
;1302;1564;1615;1754;1732;1743;1727;1819;1821;1803;1777;1821;1803;1804;1765;1819;1800;1802;1766;1801;1786;1739;1684
;1740;1717;1350;1419;1563;1515;1310;1300;1346;1234;1448;1553;1747;1704;1710;1735;1850;1804;1795;1813;1844;1790;1771
;1799;1847;1768;1743;1787;1832;1760;1716;1737;1781;1671;1440;1406;1570;1550;1366;1254;1286;1264;1435;1517;1712;1726
;1728;1735;1806;1822;1798;1784;1823;1815;1797;1785;1824;1798;1794;1760;1789;1796;1745;1727;1713;1703;1410;1472;1551
;1424;1356;1308;1279;1243;1418;1553;1748;1679;1691;1733;1857;1778;1802;1811;1840;1776;1770;1807;1867;1761;1765;1796
;1845;1729;1682;1753;1783;1677;1417;1393;1439;1422;1367;1283;1245;1316;1471;1533;1723;1703;1715;1731;1826;1808;1804
;1795;1819;1809;1812;1757;1817;1790;1786;1775;1809;1775;1721;1718;1735;1730;1373;1390;1446;1319;1368;1341;1312;1283
;1446;1611;1750;1665;1687;1738;1820;1766;1796;1783;1855;1764;1763;1815;1827;1755;1754;1782;1807;1749;1732;1703;1784
;1647;1376;1340;1407;1401;1393;1309;1328;1325;1468;1591;1730;1720;1724;1732;1803;1813;1825;1781;1829;1820;1796;1752
;1827;1819;1785;1771;1796;1778;1749;1710;1731;1687;1354;1379;1467;1361;1390;1372;1276;1261;1479;1606;1750;1674;1695
;1750;1830;1763;1777;1792;1840;1774;1744;1761;1850;1758;1749;1768;1850;1729;1701;1731;1782;1693;1393;1354;1422;1395
;1435;1342;1246;1291;1549;1615;1728;1692;1691;1718;1802;1801;1796;1777;1811;1796;1786;1739;1811;1783;1757;1731;1783
;1779;1740;1679;1708;1713;1348;1386;1497;1362;1402;1384;1312;1226;1519;1660;1752;1660;1647;1744;1825;1752;1767;1795
;1848;1751;1776;1755;1824;1732;1720;1820;1817;1701;1707;1705;1761;1647;1470;1400;1476;1410;1462;1400;1338;1339;1548
;1612;1707;1678;1689;1679;1784;1790;1781;1767;1791;1808;1772;1746;1788;1753;1760;1713;1768;1754;1709;1633;1697;1683
;1366;1412;1551;1419;1395;1405;1402;1301;1530;1645;1758;1633;1642;1725;1838;1743;1738;1778;1826;1717;1748;1748;1824
;1727;1705;1733;1795;1704;1640;1649;1803;1576;1440;1325;1482;1461;1433;1397;1360;1324;1515;1546;1687;1659;1681;1664
;1802;1772;1771;1737;1791;1778;1753;1724;1755;1773;1748;1697;1715;1705;1693;1648;1645;1690;1490;1471;1534;1390;1424
;1408;1461;1301;1471;1569;1694;1607;1640;1693;1821;1721;1742;1739;1801;1708;1693;1754;1822;1684;1696;1719;1774;1678
;1627;1638;1690;1535;1609;1464;1482;1415;1434;1368;1438;1364;1417;1547;1666;1633;1617;1641;1767;1768;1741;1720;1757
;1725;1716;1681;1716;1716;1710;1660;1708;1685;1648;1589;1632;1368
```

The first string is composed of:

- Date and time of reception of the message by the server - **22-10-2025 10:08:41**
- Device serial number - 9F25A018
- String type for string recognition by softwares - 19
- Time in milliseconds from the beginning of the cycle - 74387
- O3 concentration in digital number - 485
- Particulate matter 1; 2.5 and 10 in ug/m^3*1000 - 0;1000;1000
- CO2 concentration in ppm - 382
- air temperature (°C*100) and relative humidity (%*100) - 1961;7629
- Foliage temperature in °C*100 - 1768
- Battery voltage in mV - 3948

The second string contains:

- Date and time of reception of the message by the server - **22-10-2025 10:08:41**
- Device serial number - 9F25A018
- String type for string recognition by softwares - 17
- Time in milliseconds from the beginning of the cycle - 74387
- IR temperatures matrix in °C*100

During the design of the device and upon its completion some tests were performed in a controlled environment, in particular regarding the IR sensor, the RGB camera and the function offload to COGNIT. An example of an infrared image is reported in **figure 4.12**.

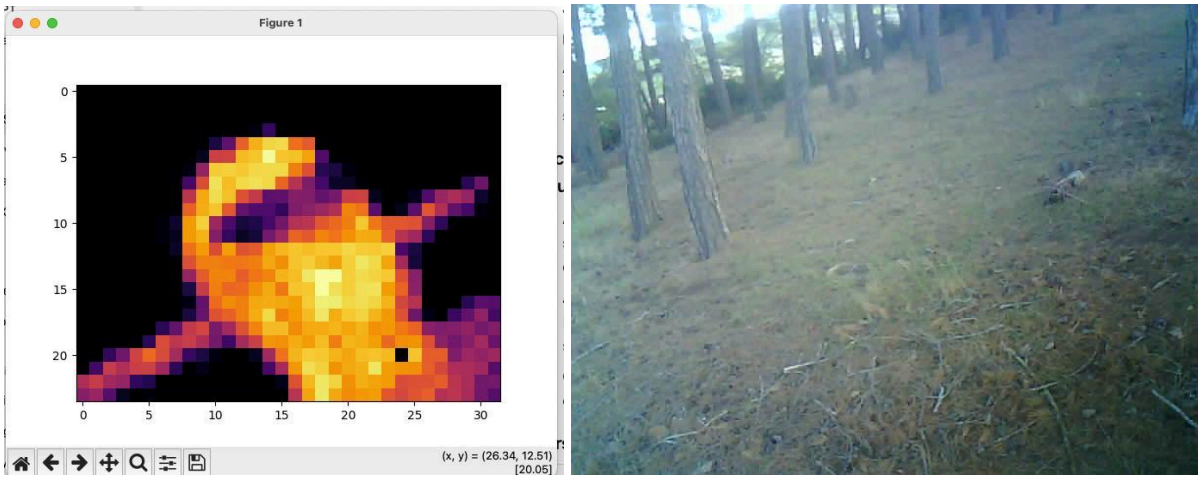


Figure 4.12: Infrared camera image captured during laboratory tests (left) and an image of a test device installed in a forest (right)

Some tests were also performed on the esp32-CAM along with the IR camera on a controlled fire. The results can be seen in **figure 4.13**.

Figure 4.13: Test conducted on a controlled flame at 10 m, 50 m and 100 m with the esp-32CAM and the infrared camera

Tests show the effect of the sunlight on the IR sensor and the esp32-CAM resolution limits when bound to the STM32L4 setup. However, the beneficial effect of shade in order to

discern the fire from the environment is also clear; shaded environments correspond to the main condition present in forests. Moreover, the latest version of the device greatly improved the camera resolution compared to the esp32-CAM. The upgrade will help both the machine learning algorithm and the operator to confirm the presence of fire. Also, it is worth pointing out that in all three cases the device would enter high-alert mode due to the clear difference between the hotter pixel and its environment. This would wake up the nearby devices that could have a better view of the flame, and make the TreeTalker Fire that spotted the flame increasing its measurement frequency to one per minute increasing the probability to capture a better image and recognize the fire presence.

4.3.3 Virtual simulation of the TreeTalker Fire network

During the development of the project, two simulations were implemented, each based on a different principle. The first simulation was based on the statistical propagation of a forest fire, while the second focused on the physical (or geographical) propagation of a fire. These simulations were created in order to stress-test the scalability of the COGNIT framework and evaluate its behavior under realistic but unpredictable conditions.

The first simulation consists of a grid of NxN squares, each with M devices inside for a total of NxNxM devices that is equivalent to an evenly distribution of the sensors in every direction. At the start of the simulation, a fire is detected at the center of the square matrix, and in each cycle, the following happens:

- All active devices contact the COGNIT Framework, sending an image of a forest fire. A device is considered active if the fire spread to it in a previous cycle.
- All devices within K tiles from the edge (as demonstrated in **figure 4.14**) belonging to an active device have a P percent chance of having fire spread to them where P depends on the distance from the center. Two possible events can be triggered:
    - The fire spreads, and the devices in the newly activated tile send an image of the forest fire to the framework.
    - The fire does not spread, thus the device sends an image of a forest (not on fire).
- The overall result of the function offload is reported on a map.
- The probability (P) of fire spreading is increased.

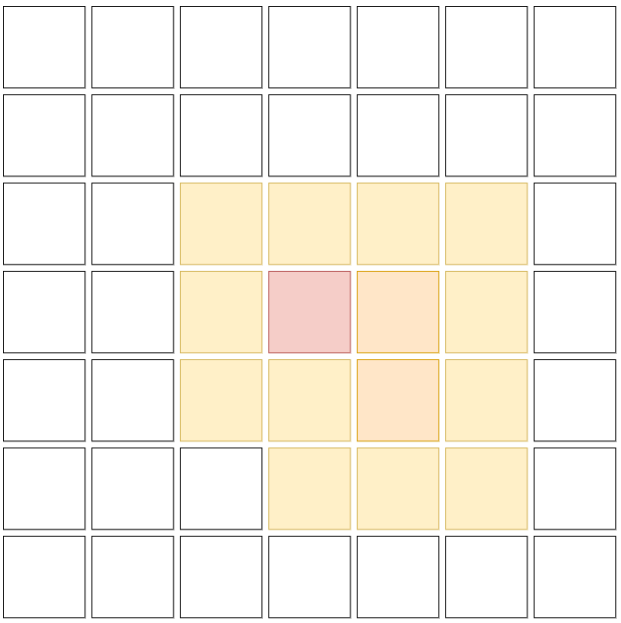


Figure 4.14: Sample 7x7 (NxN) grid with the initial fire (red) at the center, two more tiles on fire (orange), and all tiles within 1 (K) tile from a fire (yellow).

This process (summarized in **figure 4.15**) continues until all tiles in the grid are burned. To enhance realism, all active devices send their requests asynchronously and instantiate separate COGNIT Device Client instances, emulating large-scale concurrent reporting during an actual fire event.

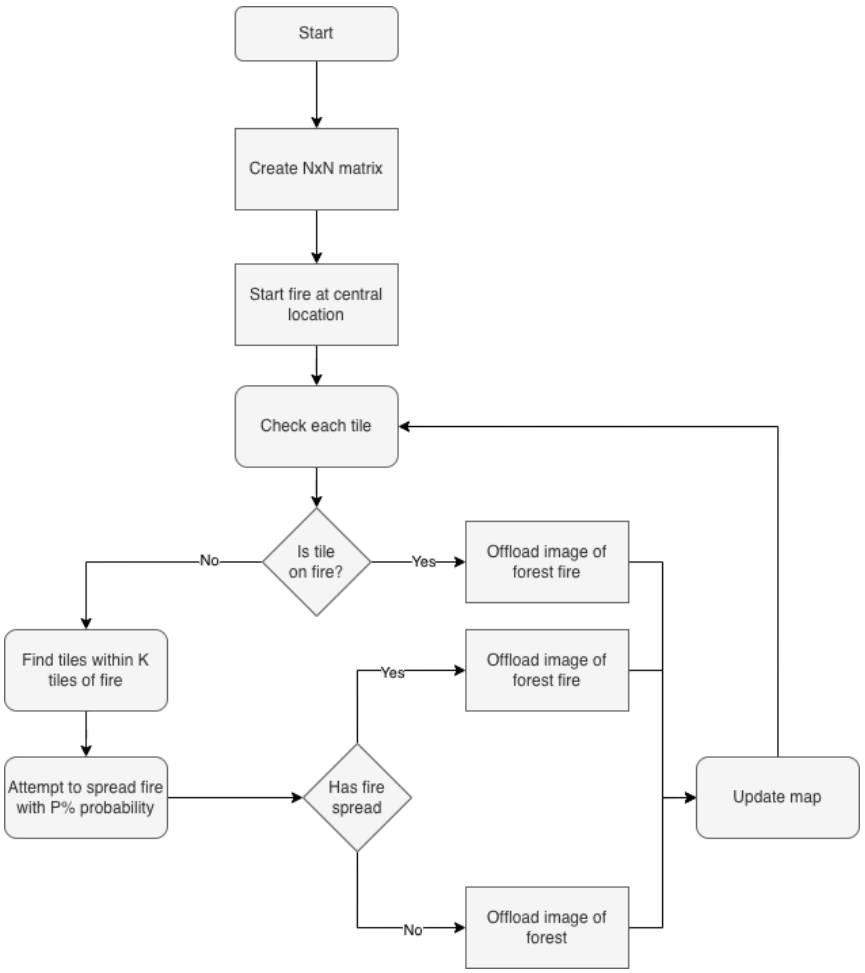


Figure 4.15: Flowchart of the statistical simulation.

The second simulation adopts a geographical perspective and encodes basic physical assumptions about fire dynamics. In the absence of wind, fire is assumed to spread radially and uniformly. In the presence of wind, the spread becomes directionally skewed, with progression biased along the wind vector. Based on these assumptions, the simulation starts by selecting a national park and placing N devices at random inside the park. Afterwards a fire is started in a random point in the park and each cycle it spreads, in an elliptical manner, with the initial point as one of the foci, and the semi-major axis oriented towards the wind. The increase in size of the ellipse in each cycle depends on the imposed wind speed. A larger ellipse spreads out to illustrate the smoke and the wake up signal, in other words, where devices might wake up but not pick up any fire on the visual and infrared cameras. Just like in the previous simulation, each device that is within the “fire” ellipse will send an image of a forest fire to the COGNIT framework, while devices in the “smoke” ellipse send out an image of a forest (not on fire). Again, just like the previous simulation, the requests are sent in an asynchronous manner, and each device creates its own device client in the COGNIT Framework to better simulate what would happen during a real-world scenario. A flowchart of the geographical algorithm can be seen below:

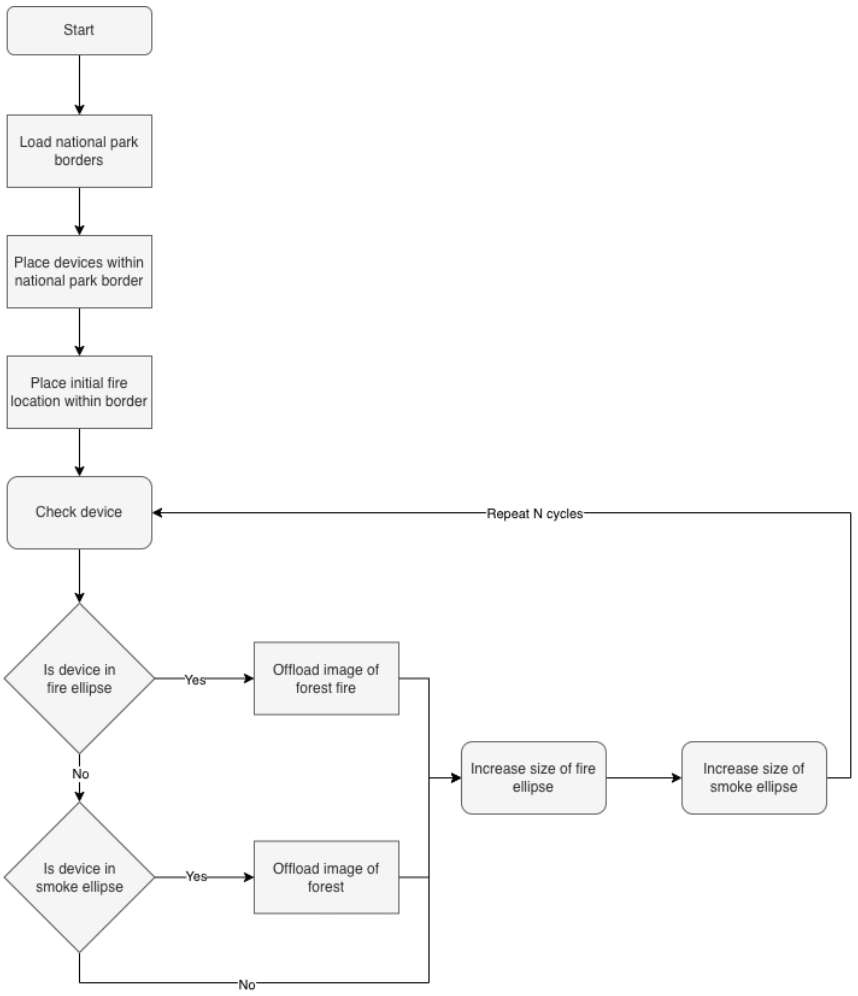


Figure 4.16: Flowchart of the geographical simulation.

The geographical simulation has been updated with multiple national parks in Italy, as well as the possibility of selecting wind speed and direction in a more interactive approach, to better serve the public demo shown at the Open Source Summit (OSS) in Amsterdam in August of 2025. Additionally, the location of the devices can be made random or more evenly spread and the area in which the devices are distributed can be restricted, which is especially useful when simulating real-world deployment of devices along known hiking routes.

The geographical simulation was chosen to publicly demonstrate UC2, being more representative to a real-world scenario both in terms of fire spread and load on the COGNIT infrastructure.

The interface of the geographical simulation can be seen in **figure 4.17**. In the left portion of the screen, the map is shown, where the lightly shaded blue area is the national park, the green dots are the sensors and the red dot is the initial fire location. The green area around the devices is the field of view of the sensors. If there is an interception between the field of view and the ellipse representing the fire, the high alert mode is triggered. On the top right, a graph showing the total number of activated sensors can be seen while on

the lower right simulation settings can be adjusted; the map (or national park), the wind direction and the wind speed. Once everything is configured, the simulation can be started by pressing the green “Start” button at the bottom right of the screen.

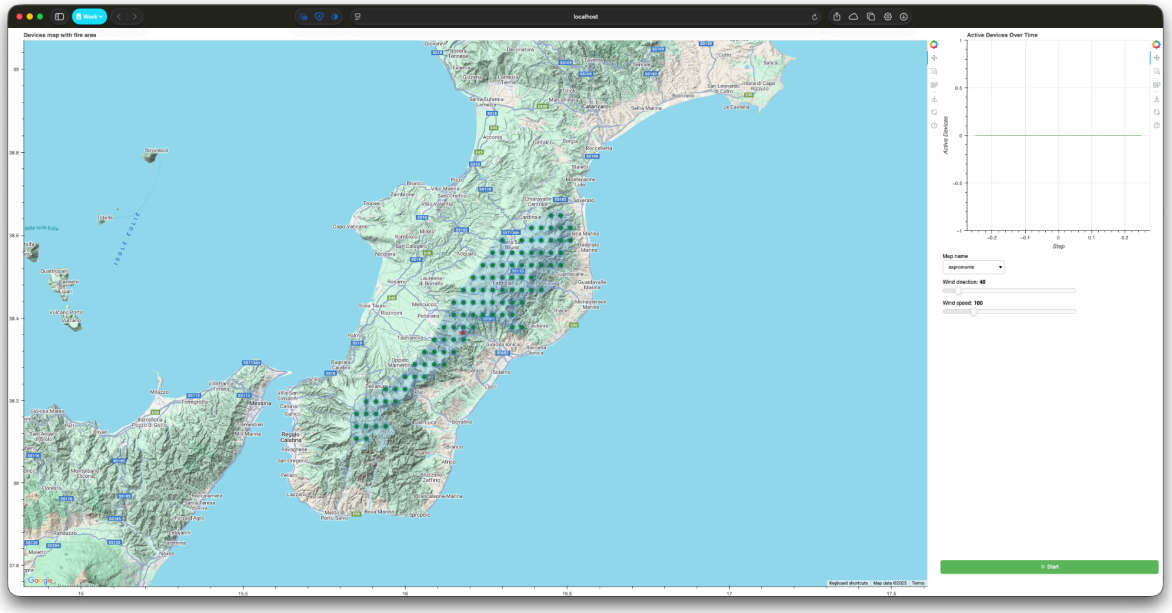


Figure 4.17: Initial dashboard of geographical simulation.

After a few cycles, the dashboard reports the fire spreading as can be seen in **figure 4.18**. On the left side of the dashboard, the map shows two ellipses, an orange inner one, where the fire is, and an outer yellow one, where its effects can be perceived but it can not be confirmed. In particular, every device that has detected a flame per result of the FaaS offloading to the COGNIT Framework is shown in red (devices have sent a fire image when their light-green circle intersects with the fire circle). On the right of the screen the graph of active devices shows the simulation history (devices that have seen a fire per the result of the function offload), which consists in a continuous increase of the number of active devices. This history of active devices shows how important the scalability of the COGNIT framework is for this use-case, as once a fire starts more and more devices will start coming online very quickly.

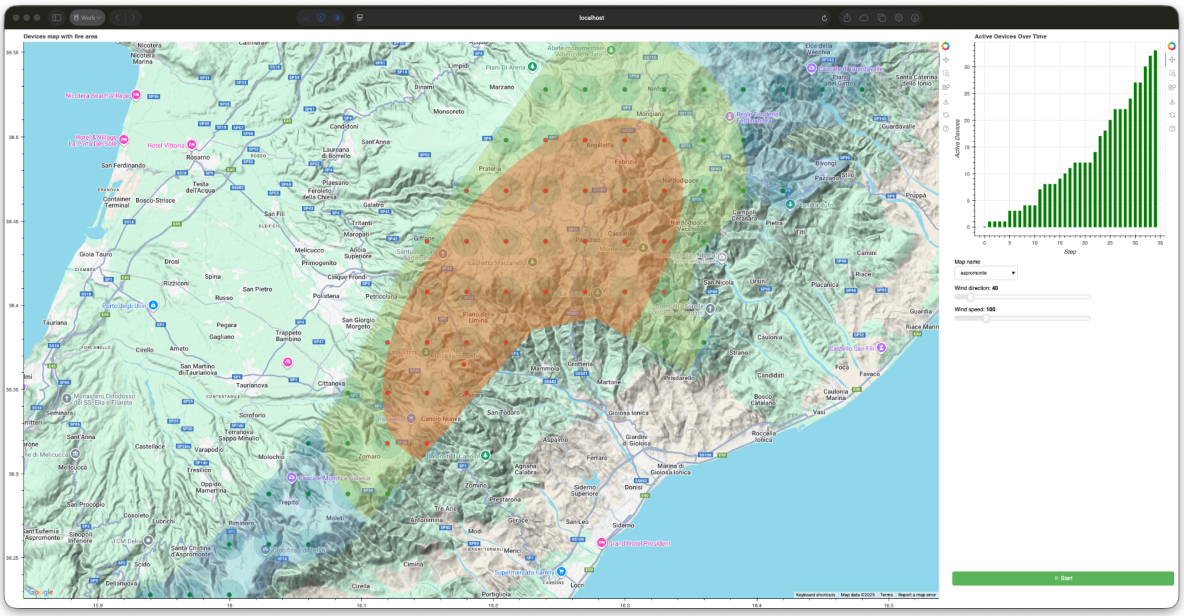


Figure 4.18: Geographical simulation dashboard after some cycles.

4.3.4 Lessons learned

During the COGNIT Project the TreeTalker Fire was developed and improved throughout the project's development cycles. It was the first attempt to apply machine learning algorithms to one of Nature 4.0's IoT devices and the COGNIT Framework proved to be a valuable asset. We were able to test different technologies, communication modules and different approaches to machine learning and image transmission and to reduce the execution time of the function by over 75% by offloading it. The challenges we faced during the project encouraged us to dive into the technology currently available on the market, particularly among edge solutions. We thus discovered new possibilities that will be applied in future, the most exciting being satellite communication coupled with edge resources. The research and development will continue, but the smart management of cloud and edge resources, as well as the clever use of them through FaaS, will be an important factor for Nature 4.0 to take into account in its future development, in particular the possibility to use COGNIT to seamlessly decouple compute power from 24/7 available services. Some of the specific COGNIT functions we would like to continue to integrate in the future are reported in paragraph 4.1.2.

4.3.5. Follow up on Risks and Mitigation Plan

Nº Risk	Potential risks		Contingency plan		Comments
	Level (1:low; 5:high)	*Impact*	*Description*	*Responsible*	
1	3	Insufficient connectivity	The location of the test could not provide a good NB-IoT or 5G coverage. Possible mitigation strategies could be testing COGNIT features using a virtual TT-Fire network, deploying a private 5G or NB-IoT access or changing the location of the test, can mitigate the risk.	R&D team	The software to test the behaviour of a virtual TT-Fire network was programmed mainly to perform scalability tests and simulations of the network behaviour. A 4G connectivity has been chosen for the device providing a better sensor data transfer speed than NB-IoT and being present in the site that was selected as potential pilot.
2	1	Issues in integrating COGNIT Client with the TT-Fire existing application	COGNIT Client is a new application and should be integrated with the existing device firmware. The risk can be mitigated through continuous collaboration and confrontation with other project partners and the required changes can be directly implemented and tested iteratively being the software directly managed by Nature 4.0.	R&D team	The device went through multiple cycles of development during the project for improving its features and to integrate and leverage the functionalities offered by COGNIT. The synergy with the partner of the consortium definitely accelerates the integration of new functions through the cycles.

3	2	Functionalities tested in laboratory does not perform as well in the field	Every detail will be thoroughly tested at different levels more and more similar to the real case scenario with an increase in complexity. However, both hardware and software will be modified in a short time if an issue arises, being the development of the devices totally managed internally.	R&D team	The functionality test performed during the project directly resulted in the improvement of the TreeTalker Fire device. The ability to rapidly implement changes led to various upgrades throughout the development lifecycle, successfully culminating in the verified and enhanced current configuration detailed in the project deliverable.
4	1	Issue in running the image recognition algorithm on the TT-Fire device	Compared to the first version some upgrades will be made on the platform such as the addition of cameras. Performing image recognition tasks directly on the microcontroller unit (MCU) will help assess and compare power consumption offloading the task to COGNIT architecture or performing it on the platform. The MCU will be upgraded in order to grant the needed resources.	R&D team	The design of the device evolved during the research project. Empirical testing revealed that performing the machine learning algorithm directly on the device was not viable due to unacceptable trade-offs in energy consumption, execution speed, and overall costs. Furthermore, the operational requirement for the operator to access the image prior to intervention made local processing functionally redundant. Consequently, the image recognition task was entirely delegated to the cloud. This decision fully mitigated the original risk by moving the resource-intensive task to a dedicated platform, making the use of the COGNIT framework even more integral and pivotal to the final operational configuration.
5	3	Low memory on the device to run the COGNIT Client, the image	As in the previous cases, the MCU will be chosen taking into account the memory and computational power requirements. Also, the	R&D team	Different microprocessors and solutions have been explored during the project and the current configuration definitely has the capabilities to execute the required tasks.

		recognition algorithm and the firmware	microphyton and the C++ programming languages will be both supported.		
6	3	Delays related to equipment procurement	During recent years, electronic supply shortages create difficulties in finding particular electronic components on the markets. The risk can be mitigated by securing the amount of components needed by the project as soon as the product is defined.	UC Leader	All the components needed for the project development and tests could be acquired.
7	4	Unable to obtain permission for installing a private deployment supporting 5G or NB-IoT	The private deployment supporting 5G or NB-IoT is a nice feature for reliability purposes and for the possibility to guarantee coverage in a challenging environment. However, it has to obtain the country's permission to be installed. It is an additional feature being the main connection assured by public networks, moreover the country test location can be changed in order to obtain permissions. The risk can be mitigated starting to ask permission in advance assuring long times for fulfil possible requirements.	Admin	The deployment of a private LTE network must be deemed optional for the wildfire use case. Its deployment is a possible solution to on site insufficient Internet signal strength. The potential pilot sites for TreeTalker Fire installation presents a strong 4G signal so the installation of a private LTE network was not deemed necessary.

4.4. Use Case Infrastructure, Demonstration, and Validation

A local testbed comprising hardware and software components to test and demonstrate the relevant features of the application as well as its interactions with the COGNIT Framework has been set up at Nature 4.0 headquarters.

The application is deployed on a network of devices that periodically scan their environment for signs of fire. These devices act as clients, offloading an image recognition function to the COGNIT Framework. The frequency and number of concurrent requests increase significantly during high-alert mode, which is triggered by wildfire events. The offloaded function utilises a convolutional neural network (CNN) to detect forest fires. If a fire is confirmed by the algorithm, additional devices in the area are alerted.

Geographically distributed Edge Clusters play a crucial role in enhancing network reliability by mitigating risks associated with network instabilities during wildfire events, thereby reducing potential points of failure. Moreover, they integrate very well with possible enhancements such as private LTE networks or satellite-LTE networks currently offered by some companies, which are expected to have a larger adoption and diffusion in the near future. Finally, the management of multiple clusters with different capabilities is important for this case study because it is representative of the situation of most civil protection infrastructures that could have different servers and resources geographically distributed, in particular some servers with lower capabilities in in the deployments near the main forests, and some servers in the headquarters farther away but with higher capabilities.

A more general description of the reference scenario can be found in section 4.1 of this document.

4.4.1 Use case demonstration and validation

COGNIT is a key component for the management of the wildfire prevention network, offering the possibility of managing and optimising local and remote resources to obtain the best results possible under the provided constraints. The devices will be mainly located in remote areas; therefore, it is fundamental to spare as much energy as possible to reduce the routine maintenance required. The possibility of offloading data analysis is valuable because it allows the device to save more energy for data collection. Moreover, the devices will operate in default mode most of the time, characterised by low priority tasks and low hardware requirements, while during the high-alert mode, both hardware requirements and the offloading frequency drastically increase. However, the activation of the high-alert mode is rare; thus, the use of FaaS reduces the average costs and helps avoid wasting computational resources.

The high-alert mode requires more computational resources and latency control to guarantee a certain level of service during emergencies. COGNIT can help manage these peaks of request seamlessly.

The demonstration of the COGNIT implementation in the UC2 scenario is divided into two parts: a virtual sensor network and the deployment of the COGNIT client on an actual device.

It is impossible to test the reaction of an entire sensor network to a fire in a physical environment because of safety concerns, logistical limitations, and the scale required for a realistic physical test. Therefore, for scale-related demonstrations, a virtual simulation of the network is preferred.

On the other hand, the deployment of the COGNIT client on real devices is essential to test the interaction between them and the framework, and to demonstrate the use case feasibility. The laboratory test on a fully operational device shows the successful use of the COGNIT framework by a TreeTalker Fire device.

The demonstration and validation are based on three different elements:

- At least one working prototype able to offload the function to the COGNIT Framework;
- One or both simulations to reproduce the behaviour of the device during wildfire events and show the features of COGNIT, in particular its scalability during unforeseen peaks of requests;
- An Edge Cluster to test its integration and use with the main testbed.

The Edge Cluster is connected to the project testbed in RISE ICE, Luleå, to demonstrate the behaviour of an Edge Cluster managed by COGNIT that is a key feature of the system for UC2.

The validation and demonstration is performed in two steps:

1. The device is forced in high alert mode by a combination of a heater and a printed image shown to the camera, to trigger the sending of a FaaS request to the COGNIT Framework every minute, to demonstrate its successful interaction with the COGNIT Framework;
2. The virtual network is used to represent a realistic installation of TreeTalker Fire sensors. A flame is placed at a random point. The simulated network is composed of 200 devices that progressively switch to the high alert mode. Their status change is monitored and two different requirements are imposed on latency to the COGNIT Framework, one for the devices that are in high alert mode (below TBD seconds) and one for the devices that have been woken up by the nearby devices (below TBD seconds). The RISE testbed and the Nature 4.0 edge node are provided as available clusters, one representing a nearer edge node and the second one representing a central server, to closely mimic the reference scenario.

The final version of the demonstrator makes use of the virtual network to show the potential of COGNIT Framework for a complete TreeTalker Fire network distributed in a realistic configuration, moreover a pilot of 20 devices has been installed to test the system of a controlled real environment near the Nature 4.0 headquarters.

COGNIT has multiple features that are currently leveraged by the UC2 and demonstrated through the described scenarios, the main ones being:

- **Scalability**: The computational resources required by the network during the default mode are very low while they dramatically increase during the high-alert mode. The scalability of the system and the possibility of responding to a sudden peak of requests are crucial in this application, and it is the main challenge that

COGNIT should face in UC2. Fires are sudden and unpredictable; nevertheless, the framework guarantees service when resources are required and the smartest use of the available resources.

- **Integration**: In contrast to other FaaS solutions, the possibility of installing the COGNIT Framework on-premise and using owned resources, or resources under the civil protection organisations is very important since real-time data on wildfires are considered sensitive information. The existence of an on-premise open-source framework compared to the commercial ones allows to make use of this technology even in projects where the application of current solutions would be impossible. Moreover, the synergy with edge nodes placed in local branches makes COGNIT a valuable solution for the smart management of computational resources of geographically distributed organizations such as those responsible for wildfire containment.
- **Control over latency**: a wildfire can ignite within minutes and propagate at speeds of up to 20 km/h (5 m/s). The faster the response, the lesser the damage. UC2 can definitely use the low-latency feature of COGNIT in a fast-changing scenario, such as wildfires, to reduce the impact of the event.

COGNIT Framework offers additional interesting features that could be applied to undergoing and future projects by Nature 4.0. The potential of COGNIT technology from a broader point of view is described in section 4.1.2.

4.4.2 Use Case Internal Testbed

The use case internal testbed has all the components needed to test a basic implementation of a TreeTalker Fire network and its integration with COGNIT. It is composed of the TreeTalker Fire device, the Nature 4.0's edge node and the virtual simulation software.

The TreeTalker fire device has been presented in detail in 4.3.2.

The virtual simulation software is described in section 4.3.3 and its code can be found in use-case-2[55] github folder (D5.9).

Finally, an on-premise server has been deployed in Nature 4.0 headquarters to test the possibility of integration between on premise and cloud resources and to be used as an edge node for the UC2 reference scenario. The hardware components are as follow:

- 2 x CORSAIR VENGEANCE LPX DDR4 RAM 64GB (2x32GB) 3200MHz.
- 2 x SAMSUNG MZ-77E4T0B/EU 870 EVO SSD 4 TB.
- 2 x Crucial P3 Plus SSD 2TB PCIe Gen4 NVMe M.2 SSD.
- 1 x AMD Ryzen Threadripper PRO 5975WX processor 3.6 GHz 128 MB L3.
- 1 x NVIDIA GeForce RTX® 4090 24GB.
- 1 x NVIDIA RTX 6000 Ada 48 GB.

During the project the server has been configured to be integrated with the RISE testbed.

[55] https://github.com/SovereignEdgeEU-COGNIT/use-case-2

4.5 Technology Readiness outlook and conclusion

UC2 started from a previously existing concept, bringing it to the next level by redesigning it completely, leveraging the features offered by COGNIT. Numerous configurations were tested during the project and several new sensors were subsequently integrated such as the IR and RGB cameras. The device is currently operational and has been tested in a laboratory. By the end of the project a small pilot is expected to be installed in a nearby forest, placing the TreeTalker Fire solidly in TRL 5.

The application of the COGNIT Framework to wildfire detection improved the effectiveness of the solution by offering FaaS functionalities while ensuring the quality of service required by a highly sensitive use case such as this one, improving latency and scalability. Moreover, the possibility to run it on premises allows it to apply the solution in sensitive fields such as disaster management by civil protection. National organizations often own bigger and smaller datacenters across the nation with different computational powers and consider the collected data as sensitive, the COGNIT Framework is therefore ideally suited for such applications.

In addition, COGNIT offers features that Nature 4.0 would like to use in the future or would like to see implemented as broadly discussed in 4.1.2. In particular, the C-client could be applied to other IoT products to improve their interaction with the cloud, and Nature 4.0 would be thrilled to integrate the COGNIT edge management with private LTE networks and satellite solutions to offer a more resilient and flexible network for its devices.

Extended COGNIT features and capabilities which would be of interest for Nature 4.0 include: Minimizing carbon footprint or minimizing cost, when external consumption-based subscriptions are used to cope with an increase of demands. Seamlessly delegating resource intensive tasks and high reliability tasks to specialized machines, reducing the overall cost and environmental impact. Nature 4.0 is looking forward to making use of these in the future, and offer the enhanced capabilities to its customers with the help of COGNIT.

5. Use Case #3: Energy

Supporting the energy transition in Europe requires allowing the wide and open accessibility of energy data. Due to diminishing fossil fuel resources (i.e. oil, gas, carbon) and the current geopolitical context, it is critical to develop solutions for individual energy independence of a household and/or small energy clusters. In order to do that it is necessary to develop methods for monitoring, predicting, and managing both energy production and consumption in a given environment. For this, the implementation of smart edge applications that make use of advanced AI/ML algorithms is a clear need.

Current deployments in the energy sector are limited by edge applications being deployed in resource-constrained environments (i.e. energy meters) and the very high security requirements for distribution system operators (DSO), both are usually resulting from centralised architectures based on private direct current (DC) infrastructures being used by the DSOs. Advanced AI/ML applications would benefit from being able to leverage, in a secure way, additional resources across the cloud-edge continuum that are much closer to the data sources and edge/IoT devices on the ground.

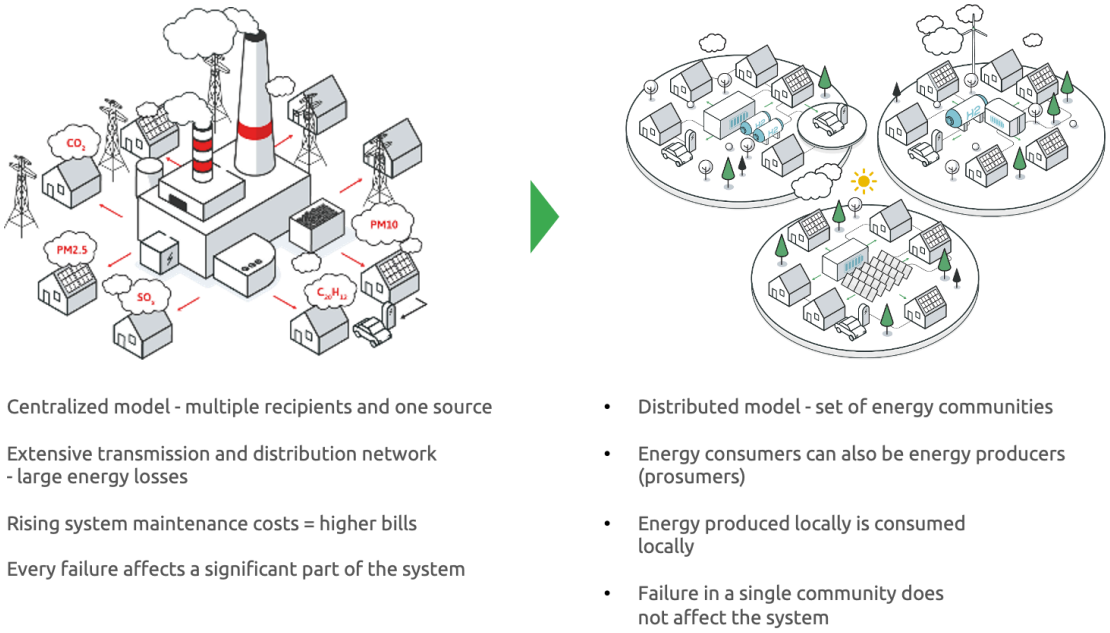


Figure 5.1. Transformation of the European energy sector.

This use case is exploring the scenario of using smart electricity meters to optimise green local energy usage in a household context, in which energy consumers are also energy producers (*prosumers*). The use case is developed and tested by evaluating its goals in Poland. In most Polish locations, the current energy system is carbon-intensive and centralised, which means that there are only a few locations in the country where energy is generated. As a consequence, electricity must be transmitted over long distances through the transmission and distribution network, resulting in high losses. In addition, bottlenecks and disruptions in the network have the potential to affect huge areas and populations. The energy industry of the future will be based on distributed systems, relying on renewable energy sources (RESs) and energy storage solutions. This highly distributed

model of the energy network features many small producers of energy, aiming to reduce costs, risks, and greenhouse gas emissions, and to eliminate transmission energy losses through more energy produced and consumed locally. To make this a reality, there is a strong need to manage both energy consumption and production to optimise usage of local energy. Electricity meters, already at the interface between the building and the power grid, are ideally positioned to manage such distributed smart energy systems.

This Use Case will demonstrate the capabilities of edge computing to support the ongoing transformation of the energy sector. It is moving from a hierarchical, centralised structure towards a more decentralised and distributed way of managing energy assets and networks.

Phoenix Systems develops the open source, real-time operating system Phoenix-RTOS, designed specifically for resource-constrained platforms, such as IoT devices. They are developing next-generation electricity meters, leveraging the Phoenix-RTOS platform, with the ambition to turn it into an Energy Assistant, capable of running user applications directly on the electricity meters and managing appliances relevant to household energy balancing. Phoenix-RTOS stands out for its reliability, flexibility and scalability of software projects for low resources platforms. The transformation of the European energy sector brings new challenges and requirements that electricity meters equipped with Phoenix-RTOS can meet.

Atende Industries develops the *besmart.energy* cloud platform, which allows energy communities, such as energy clusters or energy cooperatives, to balance energy and increase self-consumption rates, thereby reducing electricity costs for their members. Built-in high accuracy weather forecasts allow to predict production from PV or wind farms. Artificial intelligence algorithms can predict the energy demand of energy community members, enabling active control of the demand side.

5.1. Reference Scenario

In this scenario, the smart electricity meters run a user application to manage important appliances and energy assets installed (from grid topology perspective) behind the meter, adjusting and optimising operations in real time, according to user preferences. These appliances and assets include energy storages, photovoltaic (PV) installations, electrical vehicle (EV) chargers, and heating, ventilation, and air conditioning (HVAC) systems. By empowering electricity meters with apps, connected to services equipped with advanced algorithms (and eventually pre-trained AI models) making decisions about end-devices' parameters , they turn into highly personalised Energy Assistants. Running user apps is possible thanks to the Phoenix-RTOS (Real-Time Operating System), which offers all needed mechanisms for effectively and safely partitioning between user apps and Distribution System Operator (DSO) software/firmware, meeting the legally relevant requirements of the Measuring Instruments Directive (directive 2014/32/UE).

Ultimately, this approach leads to cost savings because of more effective usage of energy and lowering overall demand for coal energy, as an example.

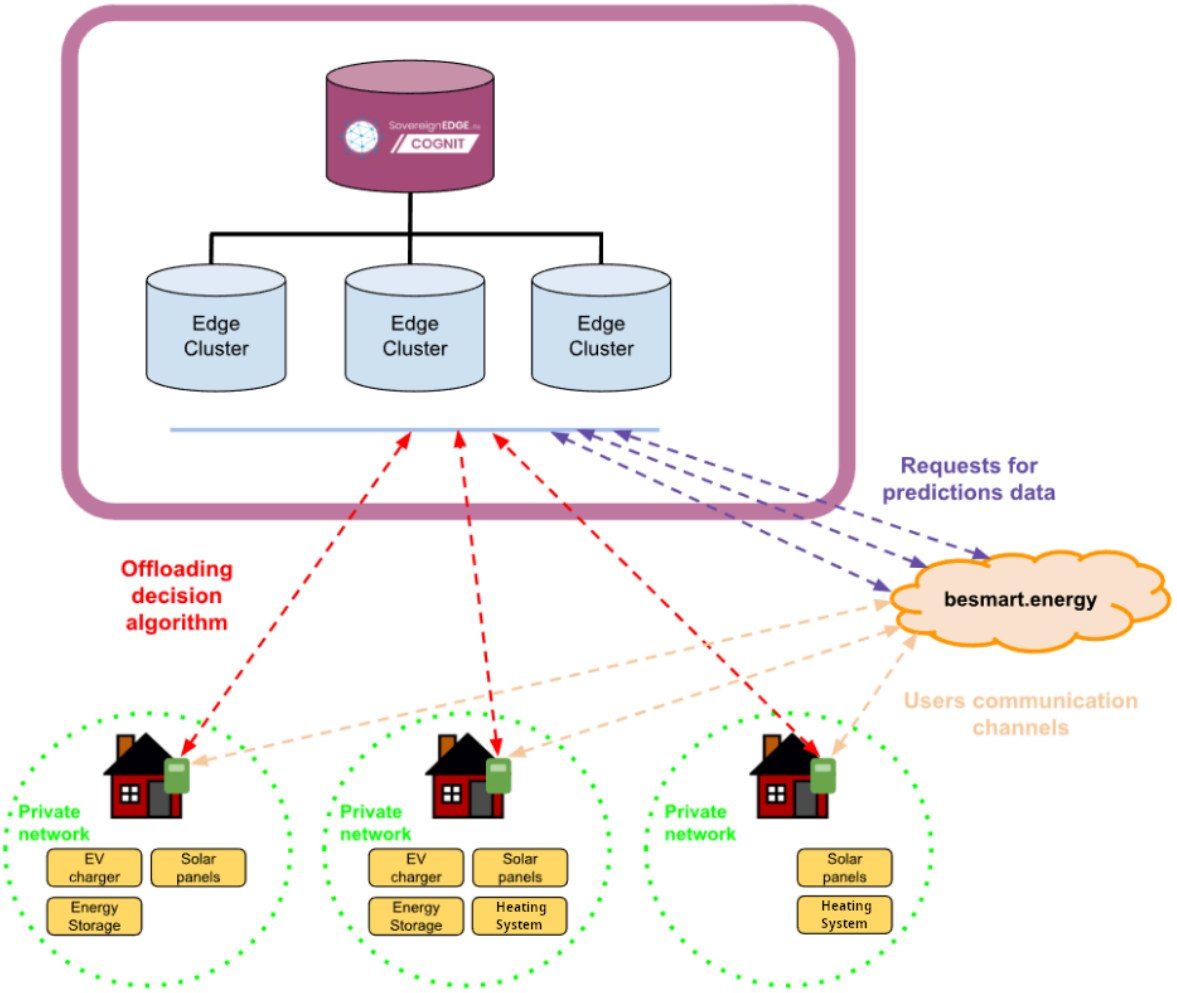


Figure 5.2. Architecture for the Energy Use Case.

The Use Case involves the following system components:

- **User** – stakeholder.
- **Devices Controller** – actuating devices.
- **Electricity Meter** – current data provisioner, manager of actuators, involves COGNIT Device Runtime.
- ***besmart.energy*** – a cloud platform for smart energy systems.
- **Edge Cluster Frontend** – entry point for the devices to offload functions (functions are executed in Serverless Runtime) and getting results back.
- **COGNIT Frontend** – main entry point for the COGNIT services.

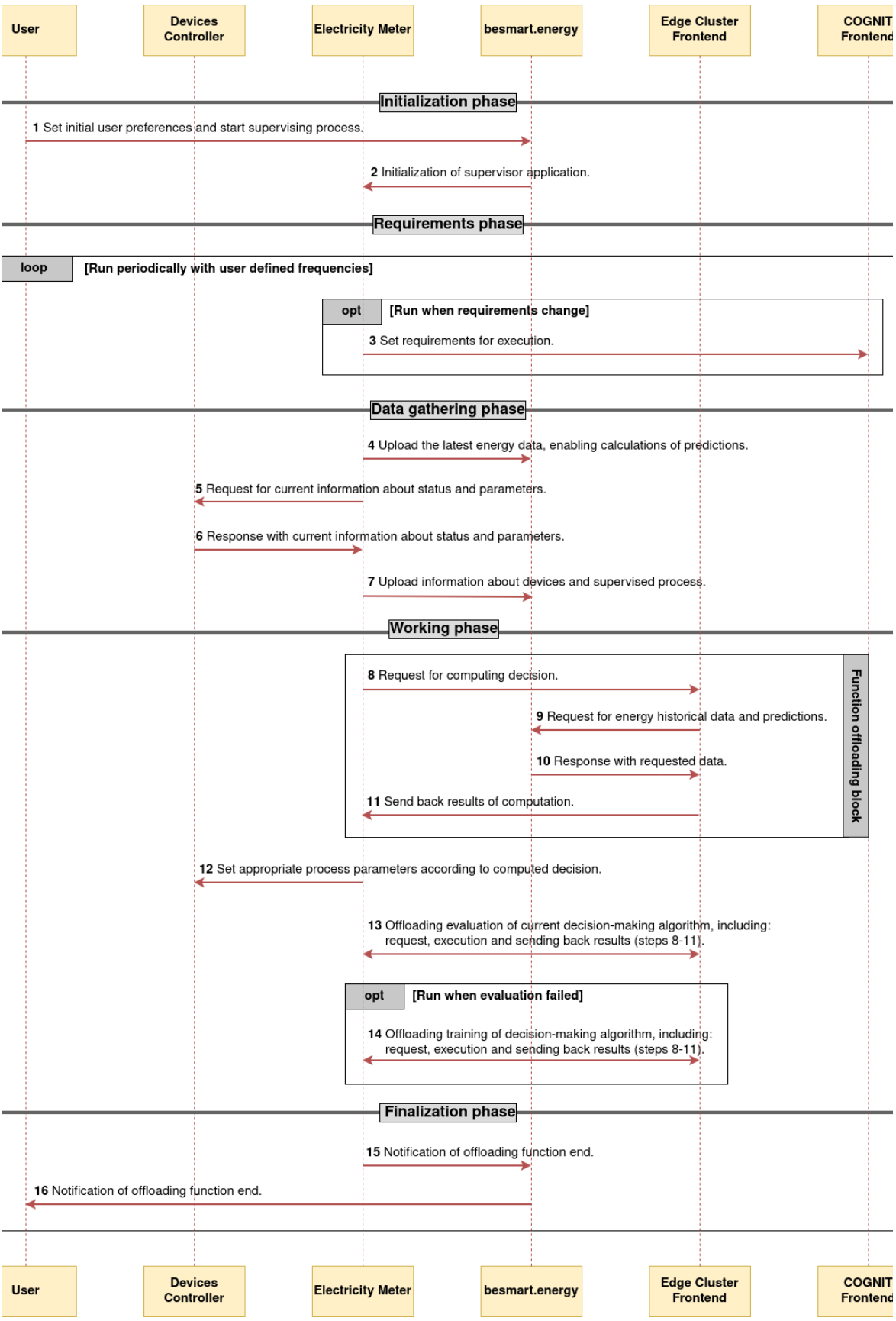


Figure 5.3. Process diagram for a single energy meter.

The simplified communication scheme between the individual components for a real-life scenario within one household is introduced in Figure 5.3.

1) The user of the besmart.energy platform sets preferences (e.g. EV car departure schedule, desired home temperature, etc.) and initiates the supervision process. For the demonstration application purposes, the user interface is the system terminal. The besmart.energy platform can be expanded in the future by adding a dedicated panel.
2) The besmart.energy platform communicates with the user's electricity meter, starting the appropriate application process and feeding it the initial user preferences.
3) The Electricity Meter communicates with the COGNIT Frontend to set execution requirements for functions to be offloaded. This step is also performed when supervision requirements change, necessitating an update.
4) The Electricity Meter uploads the latest energy consumption and production data to the besmart.energy platform’s database, enabling periodic generation of future predictions, so they are always up to date.
5) The Electricity Meter requests the current status and settings of the devices from the Devices Controller.
6) The Devices Controller responds with current information to the Electricity Meter.
7) The Electricity Meter uploads information about devices managed under the supervision processes to the besmart.energy platform’s database. The concept is that this will allow the user to view the current device status in a dedicated panel on the platform.
8) At user-defined intervals, the Electricity Meter requests the execution of the decision-making function to the Edge Cluster Frontend.
9) The Edge Cluster Frontend requests predictive data (energy consumption, energy production, and weather forecasts) from the besmart.energy platform.
10) The Besmart.energy platform responds with the required data to the Edge Cluster Frontend.
11) The Edge Cluster Frontend processes the decision-making function and responds with the results to the Electricity Meter.
12) The Electricity Meter sets the parameters computed by decision-making function to the Devices Controller.
13) At regular intervals, the Electricity Meter communicates with Edge Cluster Frontend to execute the evaluation function. This step mirrors steps from 8-11, but here, the Edge Cluster Frontend requests both historical real and predictive energy data.
14) If the evaluation results indicate that the decision-making algorithm is not accurate enough, the training function is offloaded. This is triggered by a request from the

Electricity Meter and executed by Edge Cluster Frontend, using data from the besmart.energy platform (similar to steps 8-11) .

15) Once the offloaded function is completed, the Electricity Meter notifies the besmart.energy platform of the result.

16) Besmart.energy platform sends a notification to the User about new results from any type of function that influences the supervision process.

The supervision process is continuous and is intended to ensure the optimal energy performance of household appliances. Taking that into consideration, steps 3 to 16 are repeated constantly at different frequencies. Additionally, what is not shown in the diagram, the user has the option to trigger the offloading of the decision-making (steps 8-11) or training function (step 14) manually, as needed.

5.2. Objectives and validation criteria

Implementation of the scenario described above presents a number of unique challenges and novelties, including:

- The devices (electricity meters) have very limited computing resources, e.g. only 500 kB of memory available for the entire user partition.
- Highly dynamic and varying processing and offloading needs that require relatively frequent offloading (every few minutes, depending on user preferences).
- Maximisation of local energy usage.
- Scaling the number of Serverless Runtime instances, since there will be many smart meters making concurrent requests.
- Sufficient computation resources needed for building and training AI models processing data for Use Case functionality.

To sum it up, the goals of this Use Case scenario are to:

- Transform the regular energy meter into the Energy Assistant.
- Develop AI/ML algorithms for decision making in terms of managing energetically important appliances.
- Maximise local energy consumption by using the energy meter as an actuator, based on data from the *besmart.energy* platform and the result of the energy optimization algorithm executed through the COGNIT Framework FaaS.

5.2.1. Objectives

Main aspects of demonstration and validation are to test performance of the COGNIT Framework:

- In the case when there are large amounts of concurrent requests in the area of the same local edge node;
- In case of dynamic changes in Serverless Runtime performance requirements, due to changes in user preferences.

5.2.2. Validation criteria

The validation methodology is based on a set of test scenarios, each consisting of running a defined number of smart energy meter application instances over a fixed period of time. Each application instance employs the COGNIT Framework to offload functions relevant to the specific requirements of the Energy Use Case. These functions include: the AI model training function, the AI-driven decision-making function for end-devices management and the evaluation function. In addition, each application instance may differ due to varying end-user preferences, such as the offloading frequency, or due to distinct energy consumption and production patterns. The detailed test scenarios are defined in section 5.4.

The main metric used for assessing the results of the test scenarios is the **efficiency rate** of offloading actions, defined as:

$$\eta = \frac{\lambda_s}{\lambda_a} * 100\% \quad (5.1)$$

where:

η - efficiency rate of offloading actions made by single application instance,

λ_s - number of offloading actions completed successfully,

λ_a - general number of offloading actions initiated.

Three types of efficiency rates are considered:

η_T - efficiency rate for offloading the training function,

η_A - efficiency rate for offloading the AI-based decision-making function,

η_E - efficiency rate for offloading the evaluation function.

The **resultant efficiency rate** for all application instances within a given scenario is calculated as:

$$\eta_{res} = \frac{\sum_{i=1}^{n} \lambda_{si}}{\sum_{i=1}^{n} \lambda_{ai}} * 100\% \quad (5.2)$$

where:

η_{res} - resultant efficiency rate of offloading across all instances within the scenario,

n - number of instances in the given scenario,

i - index denoting a specific application instance,

λ_{si} - number of successful offloading actions for instance i,

λ_{ai} - total number of offloading actions for instance i.

Similarly, the following resultant efficiency rates are defined:

$\eta_{T,res}$ - resultant efficiency rate for offloading the training function,

$\eta_{A,res}$ - resultant efficiency rate for offloading the AI-based decision-making function,

$\eta_{E,res}$ - resultant efficiency rate for offloading the evaluation function.

An offloading action is considered successful if a relevant response is received within a predefined timeout period after the offloading request is initialized. Three types of timeout parameters are used:

T_T - timeout for offloading the training function,

T_A - timeout for offloading the AI-based decision-making function,

T_E - timeout for offloading the evaluation function.

Each test scenario focuses on assessing specific aspects of the framework. For every scenario, target η_{res} values are defined, which should be achieved to validate the framework performance (refer to section 5.4 for details).

5.3 Summary of activities done during the project

Data collecting and processing

The aim of this use case is to explore the use of smart energy meters to optimise green local energy usage in household context. To better understand the issue, a modern electricity meter was installed in one household in Poland. This household is a real representative for the reference scenario, as it takes part in the energy exchange not only as a consumer, but also as a producer of renewable energy, thus becoming a prosumer. The user's private setting includes a photovoltaic array connected to a battery energy storage, an electric vehicle charger and an electric heating facility controlled by a wireless system.

The data from this environment are sent to the database and presented in `besmart.energy` platform, so they can be accessed from everywhere. The electricity meter collects the data related to the energy consumed from the grid and the energy returned to the grid from the local network every 15 minutes. It also enables connecting to appliances controllers and communicating with them about parameters/settings and available data.

We achieved successful integration of the following metrics for regular data queries:

- Energy returned to the local grid by the PV inverter.
- Storage state of charge.
- Maximum limit of storage charging and discharging power.
- Temperature from each sensor.
- Temperature setting for each heating zone.
- State of the switch in each heating device.

To analyse the data and use it to create a decision algorithm required pre-processing, which included the following steps:

1. Unification of units: The energy meter counts the total energy consumed/returned, expressed in kilowatt-hours, hence it is necessary to reproduce the instantaneous values in kW. Similarly, it is needed to transform the percentage level of storage charge into its current capacity in kW.
2. Unification of time base/frequency: By querying device controllers, only current values can be obtained, therefore the data gained this way is at irregular intervals. To compare data from different sources, they must refer to the same time steps. Using aggregation and interpolation, it is possible to obtain values for points with regular frequency, e.g. every 15 minutes as for energy consumed/returned signals or every one hour, as this is the standard unit of time for energy transactions.
3. Unification of time ranges: Because of required research and work, the collecting of different data started at different time. There were also some technical issues regarding the energy meter, it was not sending data for some time. Moreover, there may always be some errors on any device that make reading impossible. Therefore, the ranges of the raw series did not overlap 100%, so it was necessary to supplement the missing data. In the case of single points, interpolation can be used, but for longer periods of time, more complex methods like machine learning models trained on available data are preferable.

This part of the research work also included getting familiar with the communication protocols (like e.g. Modbus TCP) that the mentioned appliances use.

Research on modelling appliances

A comprehensive understanding of the operation of critical household devices within the local energy network is essential for optimizing their management. The implementation of an appropriate model is pivotal for the development of a decision-making algorithm and the creation of an environmental simulator. To gain insight into current modeling methodologies, a thorough literature review was conducted.

Energy storage

The primary focus of this research was on energy storage systems, with particular emphasis on battery solutions—encompassing their operational mechanisms, key parameters, modeling, and charging/discharging management. Special attention was given to studies in which storage systems were employed in residential settings to enhance the consumption of energy generated by photovoltaic (PV) panels, analogous to the scenario under investigation in this study. A relatively straightforward battery storage bidding model was developed, featuring a constant charging power limit, as it proves to be well-suited for integration into optimization problems.

State of energy in storage at time t can be described by the following relation:

$$soe_t = soe_{t-1} + (\Delta t * P_{ch,t} * \eta - \Delta t * P_{dis,t}) \quad (5.3)$$

subject to

$$P_{ch,t} \leq \alpha_{ch,t} * P_{max} \quad (5.4)$$

$$P_{dis,\,t} \leq \alpha_{dis,\,t} * P_{max} \quad (5.5)$$

$$soe_t \leq C_{max} \quad (5.6)$$

where:

soe_t is state of energy (Wh);

Δt the time lapse between moments $t - 1$ and t

η the energy efficiency of charging storage;

C_{max} the battery energy capacity (Wh);

P_{max} nominal charging/discharging power (W);

$P_{ch,\,t}$ (W) and $P_{dis,\,t}$ (W) charging and discharging powers (both always assumed positive); $\alpha_{ch,\,t}$, $\alpha_{dis,\,t}$ - limitations of those powers in range <0, 1>.

It is essential to avoid discharging energy storage systems below a specified threshold, not only to ensure an adequate energy reserve in the event of grid failure or unavailability, but also to preserve the system's longevity. A shallower depth of discharge correlates with a higher number of operational cycles before the system requires replacement. Furthermore, the optimization of storage management should incorporate a constraint on the number of charge/discharge cycles, ideally limiting it to no more than one cycle per day.

In the baseline model, that is the most commonly employed battery storage model in the power system economics literature, both $\alpha_{ch,\,t}$, $\alpha_{dis,\,t}$ are set to 1. However, to more accurately capture the battery's charging characteristics, we formulated a reduction in charging power as:

$$\alpha_{ch,\,t} = (C_{max} - soe_t) / (C_{max} - SOE_{CC,CV}) \quad (5.7)$$

where:

$SOE_{CC,CV}$ denotes the state of energy at which the constant-current (CC) switches to the constant-voltage (CV) charging scheme (Wh).

Introduction of (5.5) ensures a reduction in charging capacity after the transition to constant-voltage mode during battery charging. This modified charging scheme imposes a stricter limit, with the charging power linearly decreasing from P_{max} at $soe_t = SOE_{CC,CV}$ to zero at $soe_t = C_{max}$. This model is referred to as the linear CC-CV model.

Electric vehicle

The operating principles of batteries in electric vehicles (EVs) are analogous to those in battery energy storage systems (BESS). Consequently, a similar linear CC-CV battery model, as described in the previous section, was employed. The key distinction is that the EV battery can only be charged when the vehicle is parked at home and connected to a charging source. In contrast, the battery discharges during operation, providing power to the vehicle while driving.

Heating

Heating constitutes the largest share of energy consumption in a household. The heating source determines how electricity is converted into raising the temperature inside a home. In the case of our testing environment, we are dealing with heating mats and electric heaters, which can only work with full power. They can be switched on or off by a controller based on current temperature in the room measured by sensor and referenced temperature, set by the user. It can be assumed that the air in the room heats up directly as a result of the device operating at a specific power and efficiency for a given time.

The more challenging issue is heat loss through the floor, walls, doors, windows, roof and other surfaces that separate indoors from outdoors. Each element of the building has its own heat transfer coefficient value related to the material it is made of, which measures how much heat it allows to pass through, but one has to also consider its area and thickness. There is also another component of ventilation losses, which occurs when hot air inside the building is replaced by colder outside air through ventilation or infiltration. It is impossible to create a model that would take into account all physical dependencies, so we decided to approximate the losses using one parameter.

The temperature inside the house at time t can be determined as follows:

$$T_t = E_t / h \tag{5.8}$$

$$E_t = E_{t-1} + s_t * \eta * P_{heat} * \Delta t + \Theta * (T_{out,t} - T_{t-1}) * \Delta t \tag{5.9}$$

where:

T_t is temperature inside (K);
E_t energy accumulated in household (J);
h heat capacity (J/K);
s_t state of heating device ($\epsilon \{0, 1\}$);
η heating efficiency;
P_{heat} power of heating device (W);
Θ heat loss parameter (W/K);
$T_{out,t}$ temperature outside (K).

Development of a Smart Energy Meter simulator

One of the challenges of the R&D activities was to develop a simulator of the Smart Energy Meter (SEM simulator). The purpose of the SEM simulator is twofold. Firstly, it should be the energy-metering part of the household simulation. Secondly, it should be lightweight and scalable enough in order to easily perform scalability tests of the COGNIT framework.

The following requirements were established for the SEM simulator component:

- Scalability – one of the primary goals of the simulator is to test the behaviour of the COGNIT Framework in the situation of a vast number of requests from a

considerable number of devices. Thus the SEM simulator should be easy to build and run with multiple instances of the simulator processing different scenarios.
- Portability – the simulator should be working both on a PC with Linux OS as well as on a device with limited resources running the Phoenix-RTOS system. This enforces usage of C as the language of the implementation.
- Compatibility with meter message API – the meter message API is an interface provided by SEM for applications running on it. This requirement ensures that the application that is developed to run on SEM can use the same interface when paired with the SEM simulator.
- Flexibility – the simulator should take as input the predefined scenarios of energy usage and/or power grid state.
- Interface for household appliances – the SEM simulator should be able to attach simulators of different appliances (photovoltaic, energy storage, heating system, etc.), tracking the energy consumption and production that they create. This is a crucial feature for simulating the overall status of the Energy Use Case scenario.
- Interface in Python – this provides simplicity of integrating the SEM simulator to various applications. In particular creating the application that uses COGNIT Device Runtime in Python.

Through the process of development of the SEM simulator process a few challenges emerged. These included the following topics:

1. **Format of the scenario input**. The possible structures of definitions of the scenario were analysed. In particular, many considerations referred to whether the SEM simulator should allow continuous (linear, or curved) fluctuations of voltage and current rather than allowing only discrete changes every quant of time (1 second). The continuous changes seemed as a solution with many advantages, since it reproduces the real life situation more accurately, but for the time being it was decided to stick to a simpler solution with discrete changes as a good-enough option for current purposes. Apart from that, different file formats were analysed to find the best fit.

2. **Interface between SEM simulator and simulators of appliances**. The appliances should provide the SEM simulator with the information about the energy consumption and production that they create. These parameters can change based on a schedule defined in the scenario, or dynamically throughout the simulation (e.g. caused by a change of a device setting triggered by a result from the decision algorithm). In real situations the energy consumed by a device is a result of complicated physics. These may involve different phenomena such as voltage drops, but these complex simulations are out of the scope of the project. Thus it was decided that the appliances should provide the SEM simulator with the data on the current (in Amperes).

   Moreover, appliances are not independent from each other, as e.g. functioning of the energy storage depends on the amount of energy produced by photovoltaic panels. This requires creating a “gateway” between the SEM simulator and appliances simulators that handles this dependency logic and prepares the data for the SEM simulator.

Implementation of appliances simulators

For each type of household device previously described in terms of its modeling approach, a corresponding simulation module has been developed and implemented. These modules, written in Python, are seamlessly integrated with the SEM simulator, resulting in a comprehensive and coherent household simulation framework.

The simulation operates on predefined scenarios derived from real-world user data, ensuring realistic and representative system behavior. Furthermore, the system provides a high degree of flexibility – the number of different devices as well as the parameters of each single device are configurable. User preferences related to the expected indoor temperature and planned EV usage can also be freely adjusted. A fragment of the scenario file, illustrating how device parameters are defined, is presented in the code block below.

JSON

```
    "STORAGE_CONFIG": {
        "max_power": 12.8,
        "max_capacity": 24.0,
        "min_charge_level": 10.0,
        "charging_switch_level": 60.0,
        "efficiency": 0.85,
        "energy_loss": 0.0
    },

    "EV_CONFIG": {
        "0": {
            "max_power": 6.9,
            "max_capacity": 40.0,
            "min_charge_level": 10.0,
            "driving_charge_level": 80.0,
            "charging_switch_level": 75.0,
            "efficiency": 0.85,
            "energy_loss": 0.0
        }
    },

    "HEATING_CONFIG": {
        "heat_capacity": 3.6e7,
        "heating_coefficient": 0.98,
        "heat_loss_coefficient": 300.0,
        "temp_window": 0.75,
        "heating_devices_power": [8.0, 8.0]
    },
```

Each of the device simulators is equipped with a Modbus server (implemented using PyModbus library), enabling the household simulation, executed on a PC or RPi, to be attached to a smart energy meter running the demo application (as described in the

'Application (COGNIT Device Runtime in C) section) and the COGNIT Device Runtime. In our laboratory demonstration setup, the SEM uses the RS485 serial to communicate with device simulators that are launched on a RPi.

Development and implementation of baseline decision algorithm

We developed a simplified decision-making algorithm designed to optimize the self-consumption of a single prosumer, based on a predefined set of rules. The primary objective was to assess the functionality of the COGNIT Frontend in executing a function within our demonstration and to establish a baseline for evaluating an AI-driven version of the algorithm, with the goal of determining whether its implementation enhances self-consumption.

The primary task of the proposed algorithm is to determine the optimal control settings for end-user devices over a forthcoming time interval, with the objective of minimizing energy drawn from the electrical grid while accounting for user preferences. A key operational priority is the utilization of green energy—preferably directly generated by photovoltaic (PV) systems or previously stored in a battery energy storage system (BESS).

During the course of the project, the algorithm evolved from making decisions based solely on real-time data from the previous time step to incorporating forecasts of energy demand, PV production, and outdoor temperature relevant to the target time step. It is assumed that idealized models of home heating dynamics, energy storage systems, and electric vehicle (EV) battery charging/discharging are available, enabling the algorithm to accurately compute energy distribution across end-user devices during each control cycle.

In its final implementation, the algorithm receives input on the current status and configuration of end-device controllers, including:

- State of charge (SoC) of the energy storage system,
- SoC of EV batteries,
- Operational status of heating devices,
- Measured indoor temperature.

Additionally, it incorporates user-defined preferences such as:

- Scheduled departure times for EVs,
- Desired indoor temperature,
- Time horizon (Δt) for which the control parameters will be applied.

At the beginning of execution, the algorithm retrieves forecast data via the besmart.energy API, including:

- Predicted energy consumption by non-controllable appliances,
- Forecasted PV generation,
- Expected outdoor temperature.

Prediction errors are not considered; all forecasts are treated as ideal representations of future conditions. Consequently, the energy demand from non-controllable devices must be met precisely, without modification.

To meet the forecasted energy demand, the algorithm allocates available energy in the following order of priority:

1. Energy generated by the PV system is used first.
2. If PV generation is insufficient, stored energy in the BESS is utilized.
3. Any remaining deficit is covered by importing energy from the grid.

Subsequently, the algorithm verifies heating requirements. If the predicted indoor temperature at the end of the cycle is expected to fall below the user-defined threshold by more than ΔT, the heating system is activated regardless of green energy availability.

The next priority is EV charging. For each vehicle currently parked at the residence, the algorithm assesses whether battery charging is needed to ensure readiness for the scheduled departure. If additional green energy remains after these primary needs are satisfied, it is allocated for optional heating. Specifically, if excess renewable energy (including energy stored in the BESS above a defined high SoC threshold) is available, it is used to preheat the home, provided that the resulting temperature does not exceed the desired value by more than ΔT. This approach allows surplus green energy to be stored in the form of thermal energy, effectively reducing future grid consumption while maintaining user comfort.

Any further surplus energy, once heating demands are met, is directed toward charging both EV and stationary batteries. Only when storage systems reach their maximum capacity or charging is limited by nominal power constraints is the remaining energy exported to the public grid.

This control strategy is designed to maximize momentary self-consumption of renewable energy, while also addressing user-defined constraints and comfort preferences.

Research, development and implementation of AI-based decision algorithm

A comprehensive literature review was conducted to examine current applications of artificial intelligence (AI) methods in the domain of Home Energy Management Systems (HEMS). Particular emphasis was placed on studies in which the proposed systems incorporated components that closely align with our experimental testbed, namely: a renewable energy source (specifically photovoltaic panels), a battery energy storage system (BESS), a heating, ventilation, and air conditioning (HVAC) system, an electric vehicle (EV) with a charging station, and uncontrolled electrical loads.

A key challenge identified during the review was the limited number of publications in which the proposed solutions had been validated in real-world implementations, rather than solely through simulations or in virtualized environments.

Based on the findings of the literature review, reinforcement learning (RL) was selected as the most suitable and promising approach for optimizing the control of devices with partially or entirely unknown operational characteristics. In this machine learning

paradigm, an agent interacts with an environment in order to maximize a scalar reward signal. The agent aims to learn an optimal policy, which defines the most effective strategy for selecting actions in response to each possible environmental state. During the learning process, the agent seeks to balance exploration (gathering information about previously unknown states or actions) with exploitation (leveraging current knowledge to maximize reward). After each action is executed, the environment transitions to a new state and provides feedback in the form of a reward. This iterative process continues until the agent converges on a policy that maximizes the expected cumulative reward over time.

The architecture of the proposed solution is illustrated in Figure 5.4. In this framework, the reinforcement learning agent interacts directly with a Smart Energy Meter (SEM), which serves as the central control unit for managing all appliances within the smart home. The SEM is responsible for monitoring energy production and consumption, acquiring real-time measurements, and setting operational parameters for the controllers of end-use devices.

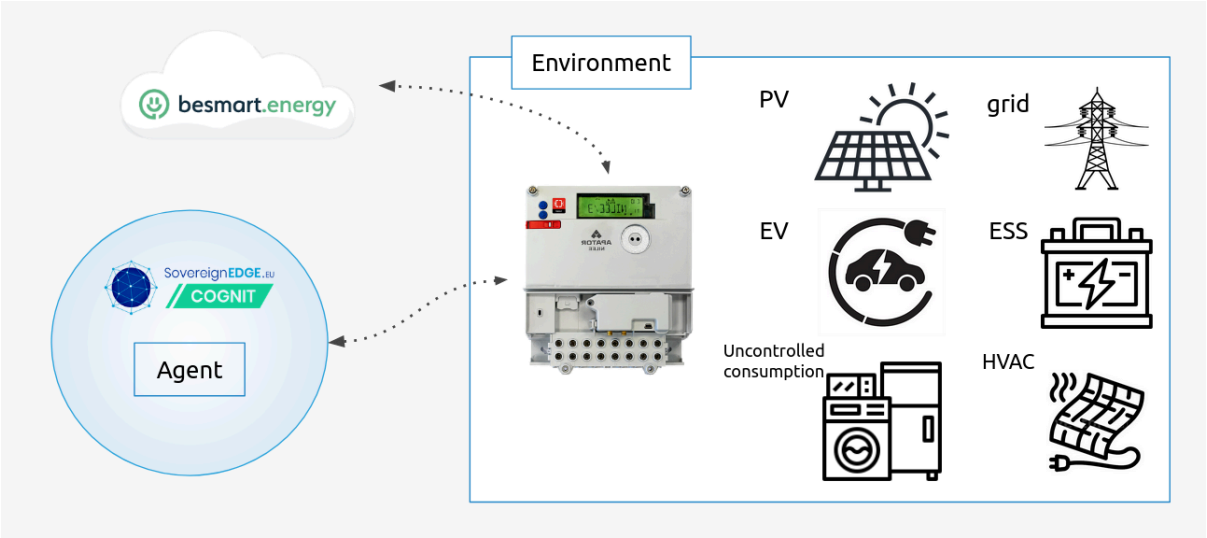


Figure 5.4. Scheme of HEMS with AI-based decision algorithm.

Due to the limited computational resources and memory capacity of the SEM, it is not feasible to deploy trained AI models or perform complex computations locally. Therefore, the action-selection logic of the agent is offloaded to the Serverless Runtime, which ensures scalability and computational efficiency.

The SEM also interfaces with the besmart.energy platform to retrieve forecast data, including predictions of energy generation, energy consumption, and weather conditions. These forecasts enable the agent to make informed decisions by considering anticipated future states of the environment.

Within the defined environment, all key components previously identified—namely renewable energy sources, energy storage systems, HVAC, EV charging infrastructure, and both controllable and uncontrollable loads—are represented. In particular, uncontrollable (non-shiftable) loads, such as microwaves, refrigerators, and personal computers, must be supplied with energy immediately upon demand; interruptions or delays in their operation are not permitted. In contrast, controllable loads, such as HVAC systems, offer a degree of

flexibility in both the timing and amount of energy consumed, provided that user-defined constraints (e.g. temperature comfort ranges) are satisfied.

The considered use case involves the HEMS performing an optimal 24-hour ahead scheduling of all appliances, with a user-defined scheduling resolution. For $\forall t = 1, 2, ..., t_N$ (N - the number of cycles in 24-hour slot, e.g. $N = 24$ for 1-hour resolution), the state of the environment comprises the eight following variables:

- h_t hour of the day,
- E_t^{PV} predicted energy generation from PV panels,
- E_t^{cons} predicted energy consumption of uncontrollable loads,
- T_t^{ind} indoor temperature,
- $T_t^{pref,low}$ distance of the indoor temperature from the lower limit of the comfortable temperature,
- $T_t^{pref,up}$ distance of the indoor temperature from the upper limit of the comfortable temperature,
- T_t^{out} forecast of outdoor temperature,
- SOE_t^{BESS} energy level of energy storage system,

And three variables for every EV in household with identifier i:

- k_t^{EVi} whether EV is parked at home,
- t_t^{EVi} time till EV scheduled departure,
- SOE_t^{EVi} energy level of EV battery.

For brevity, s_t is adopted to describe the environment state at time slot t, i.e.

$$s_t = \left(h_t, E_t^{PV}, E_t^{cons}, T_t^{ind}, T_t^{pref,low}, T_t^{pref,up}, T_t^{out}, SOE_t^{BESS}, k_t^{EVi}, t_t^{EVi}, SOE_t^{EVi}, ...\right) \quad (5.10)$$

It is assumed that the indoor temperature at a given time step $t + 1$ depends solely on the indoor temperature, the power input to the HVAC system, and the outdoor weather conditions at the current time step t. Additionally, based on a linear battery model, the energy level of the BESS at time $t + 1$ is determined only by the current energy level and the charging or discharging power at time t. A similar assumption is made for the EV battery: its charge level at the next time step depends on the current energy level and the charging power—if the vehicle is available at home—or the driving power otherwise.

Given these assumptions, the system dynamics can be effectively modelled as a Markov Decision Process (MDP). In this formulation, the next state s_{t+1} is fully determined by the current state s_t and the action a_t taken by the agent, without any dependency on prior states or actions. This Markov property enables the use of reinforcement learning algorithms that are well-suited for environments with such memoryless transitions.

The objective of the RL algorithm is to maximize self-consumption of locally generated energy, while ensuring that:

- Indoor temperature remains within a user-defined comfort range,
- The state of charge of both the BESS and the EV battery remains within their operational limits.

To achieve this, the agent seeks to determine an optimal policy for selecting values from the action set A, which includes:

- T^{set} temperature setting on the HVAC controller in range $\left[T_{min}^{HVAC}, T_{max}^{HVAC}\right]$,
- p^{BESS} BESS charging/discharging power in range $\left[-P_{max}^{BESS}, P_{max}^{BESS}\right]$,
- p^{EVi} EV i battery charging power in range $\left[0, P_{max}^{EV}\right]$.

After performing an action at the state s_t, the environment will move to a new state s_{t+1} and return a reward r_t.

The reward function is formulated as the sum of the negative energy exchange between the household and the external grid and the dissatisfaction with HVAC, BESS and EVs related to the consumer's preferred comfort and appliance's operation characteristics. The total reward is expressed as:

$$R_t = -\left(\beta^E \left|E_t^{grid}\right| + \beta^{HVAC} c_t^{HVAC} + \beta^{BESS} c_t^{BESS} + \beta^{EV} c_t^{EV}\right), \quad (5.11)$$

where β^E, β^{HVAC}, β^{BESS}, β^{EV} are the coefficients of reward elements, added to balance the targets of self-consumption and comfort. c_t^{HVAC}, c_t^{BESS}, c_t^{EV} are the cost functions for the HVAC, BESS and EVs, respectively. E_t^{grid} is the energy exchanged with the grid, which can be calculated as:

$$E_t^{grid} = E_t^{PV} - \left(E_t^{cons} + E_t^{HVAC} + E_t^{BESS} + E_t^{EV}\right), \quad (5.12)$$

where E_t^{HVAC} represents the energy consumption of the HVAC and E_t^{BESS}, E_t^{EV} represent the energy charging/discharging of the BESS and all EVs, respectively, at time t. Our goal is for E_t^{grid} to ideally achieve a value of 0, as we want to consume all green energy produced internally and not draw energy from the grid.

The cost function of the HVAC is defined as follows:

$$c_t^{HVAC} = max\left(T_t^{min} - T_t^{ind}, 0\right) + max\left(T_t^{ind} - T_t^{max}, 0\right). \quad (5.13)$$

The agent receives a penalty for the consumer's thermal discomfort that is defined as the deviation of the indoor temperature from the preferred temperature range $\left[T_t^{min}, T_t^{max}\right]$ at time t.

Next, the cost function of the BESS is expressed as:

$$c_t^{BESS} = max\left(SOE_{min}^{BESS} - SOE_t^{BESS},\ 0\right) + max\left(SOE_t^{BESS} - SOE_{max}^{BESS},\ 0\right). \quad (5.14)$$

In this case, the dissatisfaction occurs when the SOE_t^{BESS} becomes lower than SOE_{min}^{BESS} (undercharging) or greater than SOE_{max}^{BESS} (overcharging).

Finally, the cost function of the EV in defined as:

$$c_t^{EV} = \sum_i c_t^{EVi} \quad (5.15)$$

$$c_t^{EVi} = max\left(SOE_{min}^{EVi} - SOE_t^{EVi},\ 0\right) + max\left(SOE_t^{EVi} - SOE_{max}^{EVi},\ 0\right),\ if\ t \in \left[\omega_{i,arr},\ \omega_{i,dep}\right] \quad (5.16)$$

$$c_t^{EVi} = max\left(SOE_{dep}^{EVi} - SOE_t^{EVi},\ 0\right),\ if\ t = \omega_{i,dep} \quad (5.17)$$

$$c_t^{EVi} = 0,\ otherwise\,. \quad (5.18)$$

Similar to the operation of BESS, the penalty results from the SOE_t^{EVi} exceeding the allowable range specified by SOE_{min}^{EVi} and SOE_{max}^{EVi}. Unlike the BESS reward function, it occurs only when the EV is available at home, so for time t between time of EV arrival $\omega_{i,arr}$ and departure $\omega_{i,dep}$. Additionally, if SOE_t^{EVi} is lower than consumers preferred SOE_{dep}^{EVi} at the time $\omega_{i,dep}$, the dissatisfaction cost increases due to the insufficient level of energy in the EV battery.

When jointly controlling all the appliances at time slot t, the HEMS agent intends to maximise the expected return it receives over the future. In particular, the return is defined as the sum of the discounted rewards, i.e. $R = \sum_{n=0}^{\infty} R_{t+n}$, where a discounting factor $\gamma \in [0,\ 1]$ is used to explain the relative importance of the current and future rewards.

Training function for AI model

To solve the MDP defined in the previous section and derive the optimal policy π under which the agent operates, an algorithm based on Deep Deterministic Policy Gradients (DDPG) was initially employed. DDPG is an actor-critic reinforcement learning method well-suited for environments with continuous state and action spaces.

In this approach, the agent architecture is composed of two neural networks:

- The actor network, which maps a given state s_t to a specific action a_t,
- The critic network, which evaluates the quality of the chosen action by estimating the expected cumulative reward associated with the state-action pair $(s_t,\ a_t)$.

The critic network is trained using experience replay and target networks to ensure stability and convergence. It estimates a *Q*-value function that guides the policy gradient updates of the actor network. The actor network, in turn, adjusts its parameters based on the gradient suggested by the critic. This coordinated learning strategy enables the agent to refine its policy over time.

The DDPG algorithm was trained using real-world data collected from a smart energy meter, which recorded energy consumption and photovoltaic production at 1-hour intervals over a period of several months. Historical weather forecasts for the same location and time frame, obtained from a numerical weather model, were used to provide outdoor temperature inputs. Each training episode covered a 24-hour period, with the episode's starting hour selected at random to improve generalization. Initial values for indoor temperature, BESS charge level, and EV battery level were randomly sampled from normal distributions within operational limits.

Transitions between states were simulated using models of the household heating system, battery storage, and EV charging systems, developed in previous project phases.

Figure 5.5 presents example learning curves obtained for various configurations of the reward function's weighting parameters. While the results demonstrate that the DDPG algorithm is capable of convergence, they also indicate instability in training. In several instances, the final episode rewards were lower than those achieved in earlier training stages, suggesting inconsistency in policy optimization.

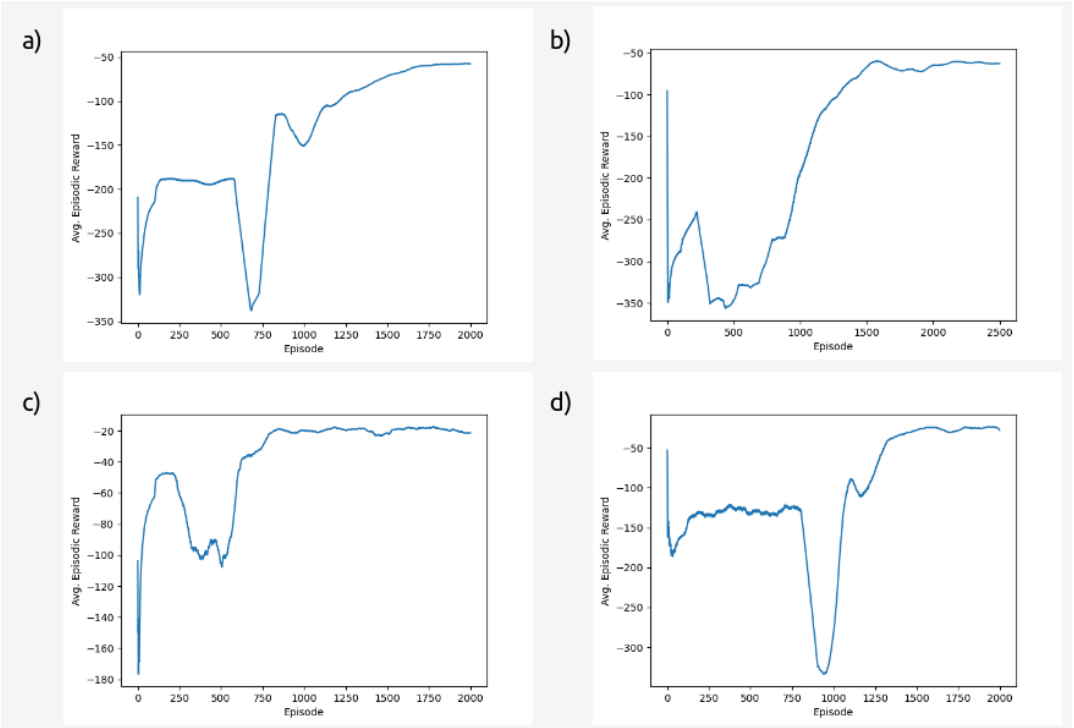


Figure 5.5. Learning curves of DDPG-based HEMS algorithm for different coefficients of individual reward function factors: a) $\beta^{E} = 0.3, \beta^{HVAC} = 1.5, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; b) $\beta^{E} = 0.3, \beta^{HVAC} = 2.0, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; c) $\beta^{E} = 0.1, \beta^{HVAC} = 1.5, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; d) $\beta^{E} = 0.1, \beta^{HVAC} = 2.0, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$.

The overarching goal was to ensure that the training process would consistently yield a functional model across different input datasets, geographic locations, and household device configurations. To improve training stability and generalization, several modifications were introduced:

- Feature scaling of input data,
- Parameter updates after batches of multiple epochs,
- Injection of stochastic noise directly into the action vector to encourage exploration.

Despite these improvements, the training process with DDPG remained sensitive to hyperparameter settings and initialization conditions. Consequently, and in line with recent advancements in the field, the training algorithm was transitioned to Proximal Policy Optimisation (PPO).

PPO, like DDPG, follows an actor-critic architecture. The actor module learns the optimal policy that maps states to actions in order to maximize expected returns, while the critic module estimates the value function of the current state. PPO improves stability and sample efficiency by introducing a surrogate loss function with clipping, which prevents large, destabilizing updates to the policy. This formulation inherently supports broader exploration of the action space without significantly increasing computational complexity.

With the adoption of PPO, training convergence was achieved consistently across different training instances, regardless of variations in input data, user configurations, or environmental conditions. Examples of learning curves using this method are presented in Figure 5.6.

The PPO-based training process continued to use recent months of historical and forecast data for:

- Energy consumption by non-controllable appliances,
- Renewable energy generation (primarily PV),
- Outdoor temperature conditions.

The reward function used to evaluate the agent's performance was based on a simulated household environment, incorporating detailed models of all end devices. These simulation models, developed in earlier project stages, allowed for accurate emulation of device behavior and energy consumption patterns, enabling realistic assessment of the agent’s decisions.

The training function was implemented in Python utilizing the following libraries:

- `PyTorch`[56] for defining the artificial intelligence (AI) model, which consists of a neural network structure incorporating activation functions and methods for determining optimal weight values.
- `NumPy`[57] for performing matrix operations.
- `Boto3`[58] to facilitate authentication and access to the S3 cloud storage service, where the trained model is stored due to limited local resources on the machine.

[56] https://pytorch.org/
[57] https://numpy.org/
[58] https://boto3.amazonaws.com/

The input to the function consists of the specifications of all endpoint devices on the farm that are subject to monitoring, as well as training parameters. The most critical of these parameters include:

- The number of episodes, which defines the duration of the training,
- The learning rates for both the actor and critic models,
- The coefficients associated with individual cost-to-reward components.

Training is conducted using real and predictive data from a specific time period (typically 90 days), which is retrieved through the besmart.energy API. During training, only the predictive values for a given time step are provided as input to the actor. However, the subsequent state and reward are derived from actual data. The reward function is designed to enable the agent to learn the relationship between predictive and actual data, ensuring that the agent's decision-making accounts for discrepancies between the two.

The function returns a boolean value indicating whether the training was successfully completed and whether the trained model was saved for future decision-making tasks.

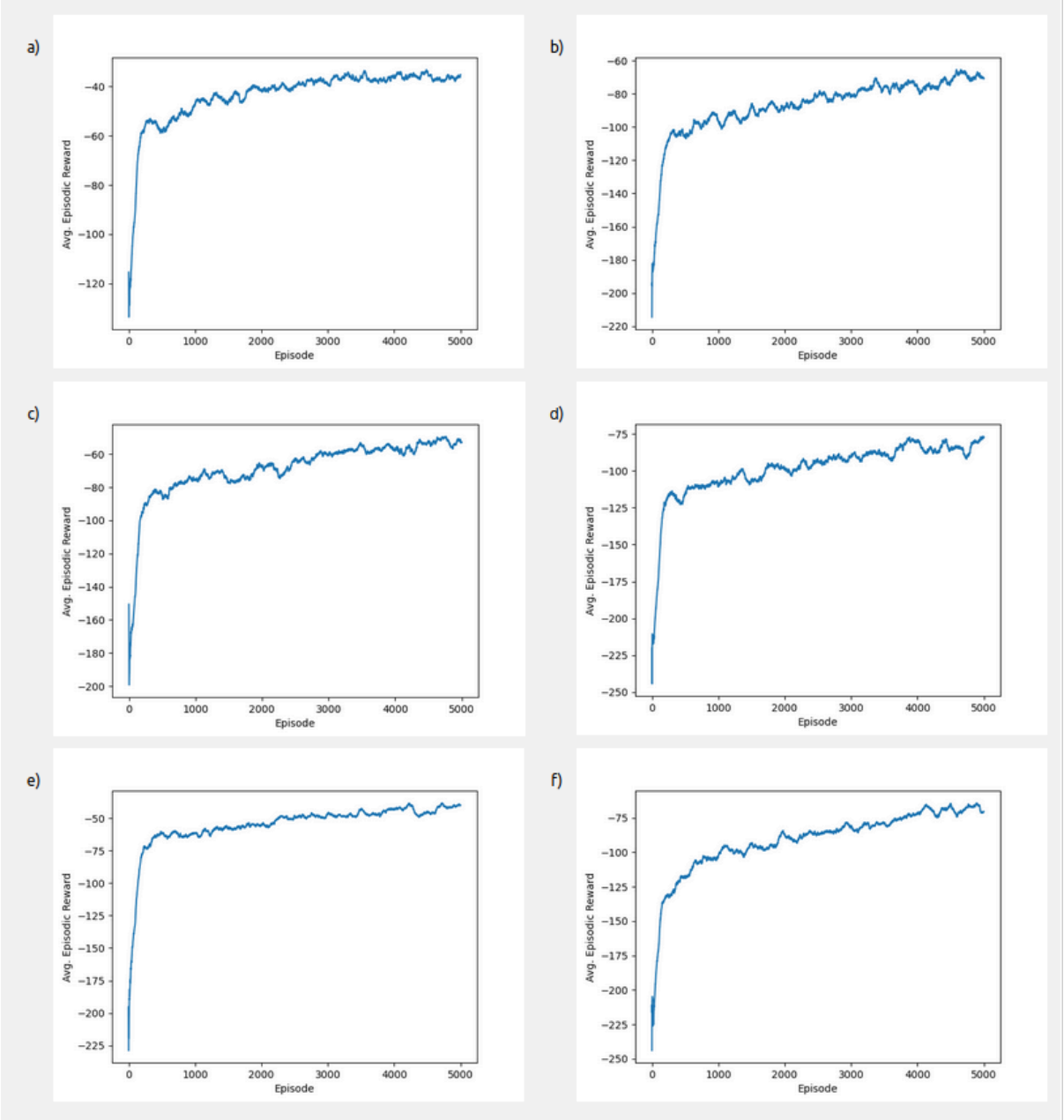


Figure 5.6. Learning curves of PPO-based HEMS algorithm for different coefficients of individual reward function factors: a) $\beta^{E} = 0.1, \beta^{HVAC} = 1.5, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; b) $\beta^{E} = 0.3, \beta^{HVAC} = 1.5, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; c) $\beta^{E} = 0.1, \beta^{HVAC} = 2.0, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; d) $\beta^{E} = 0.3, \beta^{HVAC} = 2.0, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; e) $\beta^{E} = 0.1, \beta^{HVAC} = 3.0, \beta^{ESS} = 0.8, \beta^{EV} = 0.8$; f) $\beta^{E} = 0.3, \beta^{HVAC} = 3.0, \beta^{ESS} = 0.8, \beta^{EV} = 0.8$.

Decision-making function with usage of AI model

At the core of the HEMS algorithm lies the decision-making function, which is responsible for determining the optimal operational settings for all controllable devices within the system. In the AI-based implementation, this function leverages a model individually trained for each SEM. This personalization ensures that the decision-making process reflects the specific energy consumption characteristics, generation profiles, and

behavioral preferences of the corresponding household. By capturing these unique features, the model can more accurately predict and optimize the household's energy flow in alignment with user comfort and system efficiency objectives.

The decision-making process begins by retrieving the pre-trained model from a secure S3 cloud storage service. The model retrieval procedure includes verification steps to ensure data integrity and compatibility with the current system configuration. Concurrently, the function obtains predictive data from the besmart.energy platform corresponding to the target time step. The predictive dataset typically includes short-term forecasts of PV energy generation, estimated consumption from uncontrolled appliances, and expected outdoor temperature—variables that exert a significant influence on the household's energy balance.

Additional variables necessary for constructing the full system state vector at time t, as defined in Formula (5.10), are collected directly from the SEM. These include real-time measurements such as current energy storage levels, internal temperature readings, and instantaneous power consumption of controllable devices. Once all relevant parameters are retrieved, the dataset is normalized according to the scaling factors used during model training to ensure input consistency and to maintain the stability of neural network inference.

The normalized state vector is then passed to the actor network—the decision-making component of the reinforcement learning model—which computes the optimal action vector. Each element of this output corresponds to a control signal for a specific device or system component. These actions may include, for example, setting the charging power of an EV, adjusting the heat pump's operating mode, or modifying the energy flow direction between the local storage unit and the external grid. The output from the actor network is then post-processed by the decision-making function into device-specific control parameters compatible with the physical controllers and communication protocols used within the household energy system.

Although the decision-making function involves advanced neural computations, it is designed to be computationally efficient. The inference process requires only a small fraction of the processing resources typically available in cloud environments. However, due to the limited memory and computational capacity of the SEM hardware, loading and executing the neural network model directly on the meter is not feasible. To overcome this hardware limitation, the decision-making function is deployed to a Serverless Runtime environment. This approach enables the execution of model inference on demand, without the need for continuous resource allocation, ensuring both cost efficiency and scalability.

The function is invoked at a user-defined frequency via the Device Client interface. This interface manages the communication between the SEM and the cloud-based runtime environment, handling tasks such as model invocation, data serialization, and response parsing. The execution time of the decision-making function is minimal—typically on the order of a few seconds—and the total round-trip latency, including data transfer, computation, and response delivery, should not exceed 45 seconds. This ensures that new control parameters are generated and transmitted with sufficient speed to maintain real-time or near-real-time responsiveness in system operation.

The overall design of the decision-making function emphasizes robustness, adaptability, and responsiveness. By continuously incorporating up-to-date predictions and real-time measurements, the function dynamically adapts its control decisions to changing environmental conditions, such as variations in solar irradiance, household demand, or outdoor temperature. This adaptive capability enables the HEMS to minimize energy costs, reduce dependency on external energy sources, and maximize the utilization of locally produced renewable energy—all while maintaining user comfort and respecting predefined operational constraints.

Furthermore, the modular structure of the decision-making function facilitates future extensions. For instance, additional predictors such as dynamic electricity prices, weather forecasts, or demand-response signals can be easily integrated to enhance decision quality. Similarly, the model can be periodically retrained or fine-tuned using updated data from the evaluation function, ensuring that the decision-making process remains consistent with the evolving behavior of the system and its users.

Evaluation function

The evaluation function constitutes the final, yet equally critical, element of the decision-making algorithm, designed to rigorously assess the performance of the trained model over time. Once the model has undergone its initial training phase, it is imperative to periodically evaluate and validate its ability to generate optimal actions based on the most recent data. This step is necessary to ensure the model remains aligned with real-world conditions and continues to provide high-quality decision-making outputs.

The training process equips the model with a set of parameters derived from the entire historical dataset, allowing it to adapt to the prevailing conditions, including the preferences and behaviors of residents at the time. However, as both energy consumption patterns and user behavior are subject to change over time, this adaptation may become less effective as the model’s underlying assumptions become outdated. For instance, changes in the preferred indoor temperature or the departure schedules of electric vehicles can significantly alter user behavior, necessitating ongoing adjustments to the model. These evolving factors underline the importance of regular evaluation to determine whether the model’s performance remains optimal or if retraining is required to accommodate new data.

The operation of the evaluation function is structurally similar to that of the decision function described in the preceding chapter. In this case, the function accepts as input the parameters of the monitored end devices, which are used to construct models that simulate their operation and energy flow within the local energy grid. This model-building process accounts for both the device characteristics and their interactions with the broader system. Additionally, historical data from both actual measurements and predictive models, covering the evaluation period, are retrieved using the besmart.energy API. The most recent version of the actor model is obtained from an S3 storage service. Access to both services is secured through password-based authentication mechanisms to ensure that sensitive data remains protected from unauthorized access.

Once retrieved, the model, having been converted from a data stream representation into a PyTorch-compatible format, is used to inform decisions regarding the operation of the individual device controllers. These decisions are based on the model's predictions about future conditions, as well as the current state of the system as dictated by the input data. At each simulation time step—determined by a user-specified frequency—the operation of the devices is simulated under the given input parameters. The status of each device is then updated using real-time data. The state of the system at time $t + 1$ is used to calculate the reward value, which quantifies the performance of the system based on the actions taken by the model. This procedure is performed iteratively across the entire evaluation dataset.

By default, the evaluation function returns a binary indication of whether the model is "valid." This is determined by comparing the average reward over the entire evaluation period to a user-defined threshold value. If the average reward exceeds the threshold, the model is considered to be valid; otherwise, further evaluation or retraining may be necessary. In addition to this basic mode, the function can be invoked in a more advanced mode, which outputs two key performance metrics:

Averaged reward: This metric reflects the average total reward (formula 5.11) accumulated over the entire evaluation period, providing a quantitative measure of the model's ability to achieve its objectives.

Averaged energy cost function: This metric calculates the mean of the absolute values of the energy exchanged with the external grid (formula 5.12) at each time step. Minimizing this cost is synonymous with maximizing the system's self-consumption, as it minimizes the need for external energy imports while maximizing the use of locally generated energy.

This dual-metric mode has been implemented to evaluate the performance of the AI-based algorithm, particularly in comparison to a baseline algorithm, within the context of the overarching objective of maximizing energy self-consumption while maintaining a balance with the comfort preferences of the residents. In this regard, a similar evaluation function was also developed for the baseline algorithm to ensure that comparisons between the two approaches are conducted on a level playing field.

Testing was performed using generated data from five distinct smart energy meters (SEMs), each representing different households with varying device configurations and resident activity patterns. For each SEM, ten distinct dates were randomly selected from a two-year period covered by the dataset. These dates were treated as the virtual "current" time within the simulation, with the training data representing the preceding 90 days, which the model used to learn the resident behavior patterns and system dynamics. The evaluation of both the AI-based and baseline algorithms was conducted over the subsequent 30-day period, using data the model had not previously encountered. This ensures a rigorous test of the model's ability to generalize and make accurate predictions on unseen data.

For each SEM, testing was repeated ten times with different random timestamps to increase the reliability of the results and to account for variability in the data. The results of these tests were then averaged for each SEM, providing a robust measure of the model's overall performance. The summarized results of this evaluation, averaged per SEM, are presented in Table 5.1, illustrating the comparative performance of the AI-based

and baseline algorithms in terms of both reward maximization and energy cost minimization. Across all SEMs, the AI-based approach consistently achieved improvements in both metrics, confirming that the developed decision-making algorithm effectively enhances self-consumption while respecting user comfort preferences. Although the magnitude of improvement varied with household characteristics and energy efficiency levels, performance gains were observed in all cases. Further enhancements could be realized by fine-tuning the coefficients within the reward function and increasing the number of training episodes. Additionally, the AI-based algorithm benefits from a 24-hour forecasting horizon—unlike the baseline model, which operates on a one-step-ahead prediction—allowing for more informed and anticipatory energy management decisions.

SEM	Averaged reward			Averaged energy cost function		
	baseline	AI-based	improvement	baseline	AI-based	improvement
1	-178.81	-93.91	47%	88.65	73.90	17%
2	-329.24	-147.20	55%	158.48	141.89	10%
3	-183.12	-119.34	35%	107.77	99.45	8%
4	-49.26	-41.90	15%	50.07	42.91	14%
5	-223.68	-161.58	28%	154.88	144.16	7%
mean	-195.82	-112.79	**42%**	111.97	100.46	**10%**

Table 5.1. Results of evaluation comparison between two versions of decision-making algorithms.

Application (COGNIT Device Runtime in C)

Our smart energy meters are devices with constrained resources, running on the Phoenix-RTOS operating system. Because of that, the demo application (apart from the Python one) has a C language based implementation that relies on the Device-Runtime-C. The application can be found in the GitHub repository use-case-3-phoenix-demo. For demonstrational and testing purposes, the application can be run in the QEMU environment for the ia32 target.

The activities related to the demo application in C involved:

- **Testing of the Device-Runtime-C**. Significant effort was put by relevant partners into the development of the Device Client in C. We provided support for this development by testing a couple of versions of the Client on our devices and by giving comments and feedback about the solutions that are most suitable for our smart energy meters and potentially other similar devices. This involved e.g. the suggestion to use the `mbedtls` library for secure communication instead of `openSSL`, since the former is much better suited for small, resource-constrained devices.
- **Integration with the devices' simulators using Modbus protocol.** The application aimed at resembling the real-life scenario in the way it communicates with

household devices. For that purpose, the communication between the application and the household devices simulators uses the Modbus protocol. When the application runs on the SEM device, it sends and receives Modbus frames over the RS485 serial port. In order to introduce these solutions the Modbus servers were added to devices' simulators with the help of `pymodbus` library. On the side of the C application a simple Modbus client library was developed as part of the R&D activities.

- **Putting it all together in the application logic.** The demo application structure has been established and implemented.
- **Integrating into Phoenix-RTOS and SEM.** The application was successfully launched on both the SEM (using iMX.RT1176 target processor) and on ia32 architecture running on QEMU.

The application has **three** main components:

- **Integration with the Device-Runtime-C and a mechanism for offloading functions**. The application uses the COGNIT Framework in the following manner:

```C/C++
static void *deviceRuntimeThread(void *args)
{
    offload_ctx_t *ctx = (offload_ctx_t *)args;

    if (device_runtime_init(
            &ctx->deviceRuntime,
            ctx->config->cognit,
            ctx->config->decisionReqs,
            &ctx->faas) < 0) {
        log_error("device_runtime_init failed");
        setState(ctx, offload_stateInitializationError);
        return NULL;
    }

    setState(ctx, offload_stateNone);

    while (1) {
        waitForReadyInput(ctx);

        const scheduling_t *reqs =
            ctx->currentFunction == offload_functionTraining ?
            &ctx->config->trainingReqs : &ctx->config->decisionReqs;

         e_status_code_t status = device_runtime_call(&ctx->deviceRuntime,
            &ctx->faas, *reqs, ctx->response);

         setState(ctx, status == E_ST_CODE_SUCCESS ?
            offload_stateResultReady : offload_stateError);
    }
```

```
    /* This function never returns */
    return NULL;
}
```

- **Sending data to besmart.energy.** The AI model uses energy-related data that should be aggregated and processed by `besmart.energy` beforehand. Thus, the application has to report that data to that service.
- **Communication with the devices using Modbus.** This includes both reading the devices' state and setting the configuration as ordered by the result of the AI decision model.

All of the above aspects of the application are indicated on the following screenshot that displays the log messages from the application.

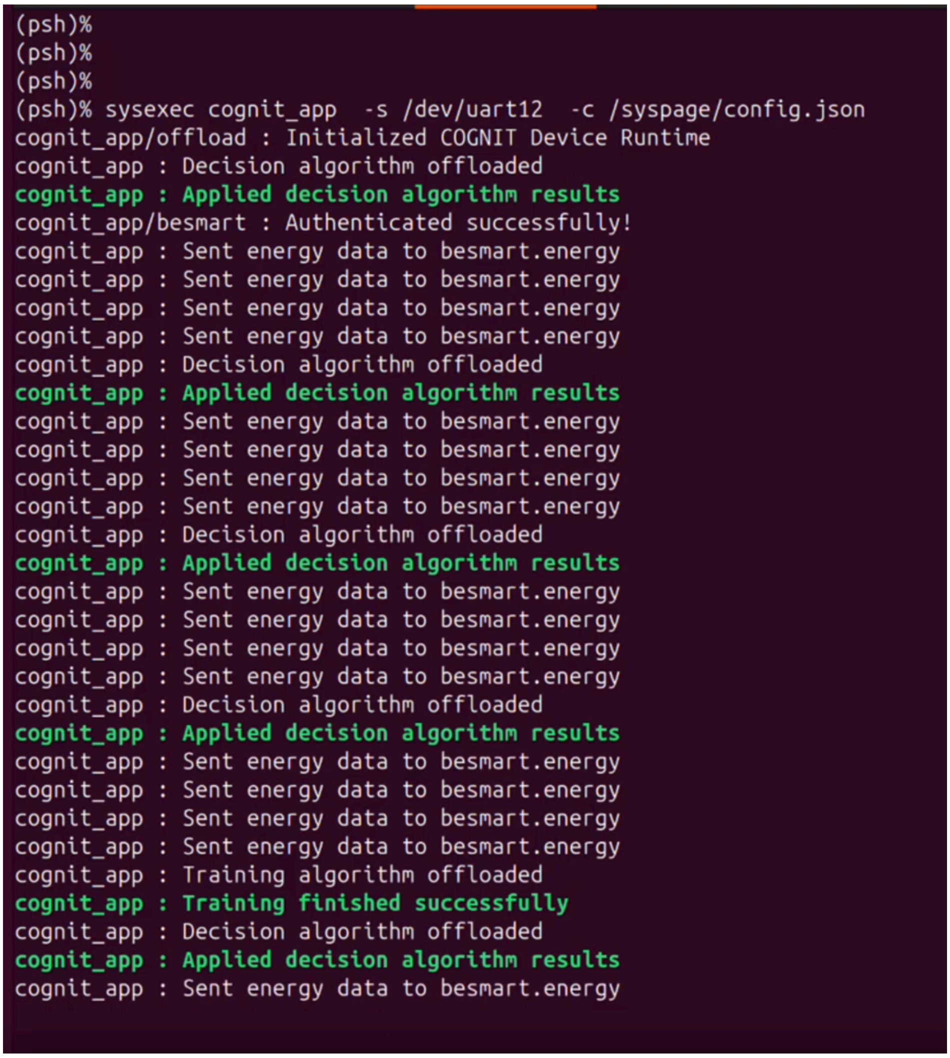


Figure 5.7. Screenshot presenting logs from the demo application running on a Smart Energy Meter

Application (COGNIT Device Runtime in Python)

The demo application in Python has been developed to support the validation of representative use-case scenarios and to evaluate the COGNIT Framework's overall performance, stability and responsiveness. The application enables the simulation of parallel operation among dozens to hundreds of households, each equipped with SEMs that dynamically compete for shared computational resources within the Cloud-Edge infrastructure when offloading functions. This large-scale configuration reproduces realistic system behavior that cannot be feasibly replicated within a conventional laboratory environment.

The demonstration is executed on a PC rather than the target device – SEM. The application integrates the COGNIT Device Runtime in Python to offload computational functions to the COGNIT Framework.

Just like the application in C, the Python one periodically performs the necessary actions:

- Acquiring data from the device simulators and SEM-simulator,
- Collecting user preferences data (e.g. temperature, EV plans),
- Offloading decision function,
- Offloading evaluation function,
- Offloading training function if necessary.

This is indicated in the main loop of the program, which looks like this:

```
Python
while not self.shutdown_flag:
	now = self._get_sem_time()

	new_model = False
	if self.use_model and now >= self.last_train_run +
self.cycle_train_time:
		if not self._offload_evaluate():
			new_model = self._offload_train()

	if new_model or now >= self.last_predict_run + self.cycle_time:
		self._offload_predict()
```

The application provides the possibility of dynamically changing the offloading frequency of both decision and evaluation functions.

Offloading is done in the following manner that enables convenient error tracking and preparing efficiency statistics:

```
Python
    def _run_algo(
```

```
            self,
            algo_function: Callable,
            algo_input: AlgoPredictParams | AlgoTrainEvalParams,
            function_type: str,
            timeout: int = 6000,
    ) -> Any:
        if not self.use_cognit:
            res = algo_function(*astuple(algo_input))
        else:
            try:
                ret = self.device_runtime.call(algo_function,
*astuple(algo_input), timeout=timeout)
                res = ret.res
                self.app_logger.info(f"\nRuntime result: {res}")
                if ret.err:
                    self.app_logger.error(f"\nError during offloading:
{ret.err}")
                    self.global_logger.info("Offload ERROR")
                    self.offload_stats[function_type].handle_error(ret.err)
                else:
                    self.global_logger.info("Offload OK")
                    self.offload_stats[function_type].handle_success()
            except:
                self.global_logger.error("Offload FATAL ERROR")
                self.offload_stats[function_type].handle_error("FATAL")
                return None
        return res
```

5.3.1 Lessons learned

The COGNIT Project was developed using an iterative approach. The project was organized in development cycles where every cycle ended with physical General Assembly meetings. That was a really useful approach because it allowed for coordinating all partners around common project goals. At the end of the day it seems to be one of the key aspects to manage such a consortium where there are multiple partners with different roles from all over the European Union.

Developing software for the furthest link (IoT devices) of Cloud Edge Continuum is different from developing high-level software. Rising awareness of that in different consortium partners as soon as possible was crucial to achieve project goals. Common (among partners) requirements defined regarding framework functionalities allowed for pointing out needs of every use case partner and found common ground for the COGNIT Framework to fulfill all defined needs.

5.3.2. Follow up on Risks and Mitigation Plan

Nº Risk	Potential risks		Contingency plan		Comments
	Level (1:low; 5:high)	*Impact*	*Description*	*Responsible*	
1	4	Insufficient connectivity	The location of the device may impact the connectivity as the coverage of GSM differs for different locations. Given that the goal is to feed the upper level system with energy consumption and production data within a one minute interval may the risk manifest it will be mitigated by developing means of device connectivity via end user LAN network instead of GSM.	Technical Lead	The risk of insufficient connectivity materialized. Connection using GSM was unable to fulfill criteria that data should be sent within one minute intervals. The mitigation plan was used and connectivity using the LAN network was developed. It allowed for successfully fulfilling criteria of sending data in one minute intervals.
2	2	Insufficient resources on the device (electricity meter)	For the first phase of the project a simulator will be developed to check the concept within the simulation environment. Once the concept is verified the COGNIT Client (or its essential for the Use Case functionality) will be ported and used on the device.	Project Lead	The risk of insufficient resources on the device materialized. However, the mitigation plan of using a simulation environment to develop UC3 solutions allowed for progress with works correctly until COGNIT Device Runtime in C (low memory footprint) was ready.
3	2	Not achieving interoperability and	This risk is mitigated through stating the requirements for the Use Case and close collaboration with other parties involved in the project. Additionally, to mitigate this risk, an iterative approach for project development was	Project Lead	Risk not materialized.

		orchestration	assumed, allowing for the parties to identify any interoperability and orchestration problems in time.		
4	3	Too high latency preventing correct performance of decision algorithm	In case the latency is too high it is foreseen to add an extra edge node in the test environment premises.	Technical Lead	Risk not materialized

5.4. Use Case Infrastructure, Demonstration, and Validation

On-premise Edge Cluster

The cluster consists of 4 machines with the following specification:

- Dell PowerEdge R640
- CPU: 2 x Intel Xeon Gold 6138, 20-core 2000/3700 MHz
- RAM: 8 x 32GB PC4-2666V, 2666 Mhz
- Storage: 2 x SSD SATA 480GB 6Gbps 2.5* + 4 x NVMe 960GB U.2 2.5*

out of which 3 are executive and 1 serves as backup.

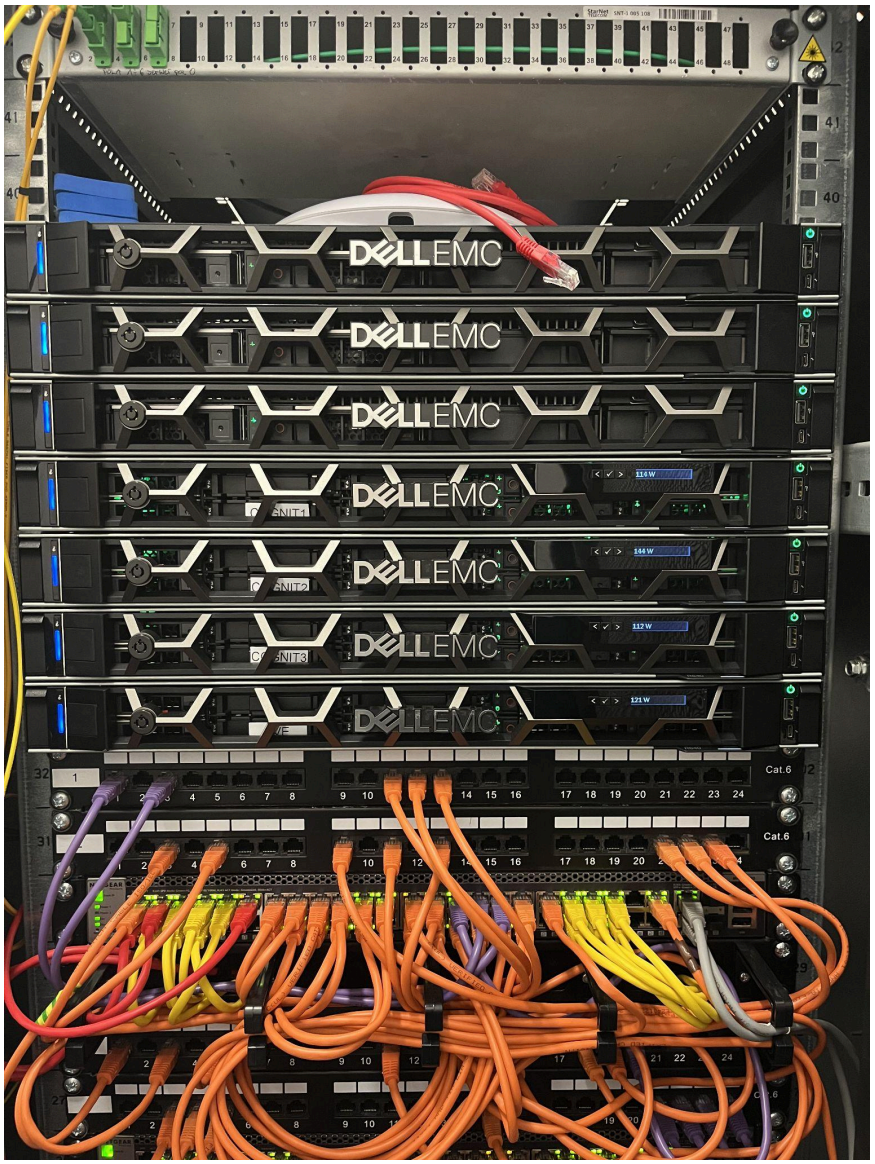


Figure 5.8. Machines of the Phoenix cluster (from 4th to 7th rack counting from top)

The machines were set up in the office of PHOENIX and integrated into the COGNIT testbed as `phoenix-cluster` Edge Cluster. This required some work in terms of:

- Installation of the Open Nebula hypervisor on each of the machines,

- Adjusting local network of the office for secure usage of the cluster,
- Providing appropriate virtual networks for connection with the Cognit Frontend in the testbed,
- Exposing the local Edge Cluster Frontend endpoints to be publically available.

Laboratory environment demo

The lab-demo consists of a few smart energy meters, each running the demo application in C (see picture below). All of them use GSM modems for Internet connection, which is necessary for offloading functions to the COGNIT Framework. SEMs are wired to a Raspberry Pi (in the middle of the picture) using the RS485 standard, hence the USB <–> RS485 converters are used. The Raspberry Pi (RPi) runs a household simulation for each of the meters.

Figure 5.9. Lab-demo setup

Simulation environment demo

The demo simulation environment is based on running multiple demo applications in Python. The script `multi_demo_runner.py` launches simulations and demo apps for all specified households scenarios:

```python
households = []
for id in sem_id_list:
    hsim = HouseholdSimulator(
        sem_id=id,
        config_dir=Path(cmd_args.scenario_dir),
        time_machine=time_machine,
        live=cmd_args.live,

training_state_changed_cb=training_controller.training_state_changed,
    )
    households.append(hsim)

for hsim in households:
    hsim.start()

print("Starting simulation")
time_machine.resume() # Start the simulation
```

The `time_machine module` ensures the synchronization of time progression across the households.

The number of simulated households scenarios can be scaled using the `--multiply N` option. This is particularly valuable for reproducing the validation scenario with a large number of smart energy meters performing offloading different functions at varying frequencies.

Additionally, a comprehensive statistics facility has been developed to provide the monitoring and analysis of offloading actions results. This functionality again proves especially useful when running test/validation scenarios involving many SEM devices.

```
>>> get_stats()
STATS:
H0 sem123456 train: 1/1, decision: 4/4
H1 sem235723 train: 1/1, decision: 4/4
H2 sem337656 train: 1/1, decision: 4/4
H3 sem683251 train: 1/1, decision: 4/4
H4 sem730853 train: 1/1, decision: 4/4
H5 sem832852 train: 1/1, decision: 4/4
H6 sem123456 train: 1/1, decision: 4/4
H7 sem235723 train: 1/1, decision: 4/4
H8 sem337656 train: 1/1, decision: 4/4
H9 sem683251 train: 1/1, decision: 4/4
H10 sem730853 train: 1/1, decision: 4/4
H11 sem832852 train: 1/1, decision: 4/4
H12 sem123456 train: 2/2, decision: 6/6
H13 sem235723 train: 1/1, decision: 4/4
H14 sem337656 train: 1/1, decision: 4/4
H15 sem683251 train: 1/1, decision: 4/4
H16 sem730853 train: 1/1, decision: 4/4
H17 sem832852 train: 1/1, decision: 4/4
H18 sem123456 train: 1/1, decision: 4/4
H19 sem235723 train: 1/1, decision: 4/4
H20 sem337656 train: 1/1, decision: 5/5
H21 sem683251 train: 1/1, decision: 4/4
H22 sem730853 train: 1/1, decision: 4/4
H23 sem832852 train: 1/1, decision: 4/4
H24 sem123456 train: 1/1, decision: 4/4
H25 sem235723 train: 1/1, decision: 4/4
H26 sem337656 train: 1/1, decision: 4/4
H27 sem683251 train: 1/1, decision: 4/4
H28 sem730853 train: 1/1, decision: 4/4
H29 sem832852 train: 1/1, decision: 4/4
```

Figure 5.10. Screenshot presenting statistics logs related to running multiple demo instances.

Test scenarios

The Energy Use Case helps to validate the project goals related to meeting end user needs (including expected quality of service) without manual intervention, and automatic scaling of the infrastructure. The two test environments, which correspond to deployments in real households, are:

- Laboratory environment demo (evaluation of application using real smart energy meters),
- Simulation environment demo (evaluation of infrastructure scalability).

Four test scenarios have been defined to evaluate the project goals. These are as follows:

1. **Vast number of concurrent requests**

Using simulation environment demo for running many (N=100) parallel simulations of households.

Frequency of offloading actions (only the AI-based decision-making function) by a single SEM simulator is constant (every 15 minutes).

Simulations run in regular simulation scenarios:

1) Periodic offloading of the AI-based decision-making function.

2) User requirements changes are predefined in daily schedules (without significant changes in user habits).

3) Simulations do not have to start at the same timestamp, as in real-life conditions the meter's time is not synced to the ms. In practice it is regarded that <30s deviation from the time source is acceptable.

To sum it up in this scenario there aren't any special changes in user requirements and also any additional AI retraining function triggering.

Parameters of scenario are:

$N = 100$ - number of parallel simulations,

$T_{A,off} = 15min$ - time period between regular offloading requests (the AI-based decision function) by a single device,

$T_{test} = 12h$ - overall time period when scenario runs,

$T_A = 45s$ - timeout for offloading AI-based decision-making function,

Validation criteria are:

$\eta_{A,res} = 90\%$ - resultant efficiency rate of offloading the AI-based decision-making function

2. Long test

Using the laboratory demo environment to offload functions at constant frequency.

Executions run in regular scenarios:

1) Periodic offloading of the AI-based decision-making function and the evaluation function.

2) If needed, trigger the training function (decided by evaluation result).

3) User requirements changes are predefined in daily schedules (without significant changes in user habits).

To sum it up in this scenario there aren't any special changes in user requirements and also any additional AI model training function triggering.

Parameters of scenario are:

$N = 4$ - number of devices using the framework,

$T_{A,off} \in \{1h,\ 2h,\ 3h,\ 4h\}$ - time period between regular offloading the AI-based decision-making function by a single device,

$T_{E,off} \in \{12h,\ 24h,\ 36h,\ 48h\}$ - time period between regular offloading the evaluation function by a single device,

$T_{test} = 7\ days$ - overall time period when scenario runs,

$T_T = 10min$ - timeout for offloading the training function,

$T_A = 45s$ - timeout for offloading the AI-based decision-making function,

$T_E = 90s$ - timeout for offloading the evaluation function.

Validation criteria are:

$\eta_{T,res} = 90\%$ - resultant efficiency rate of offloading the training function,

$\eta_{A,res} = 95\%$ - resultant efficiency rate of offloading the AI-based decision-making function,

$\eta_{E,res} = 90\%$ - resultant efficiency rate of offloading the evaluation function.

3. Model retraining

Using the simulation environment demo for running parallel simulations of households (N=10).

Frequency of offloading actions by a single SEM simulator is constant.

Simulations run in regular simulations scenarios:

1) Periodic offloading of the AI-based decision-making function and the evaluation function.

2) The training function is offloaded after every evaluation.

3) User requirements changes are predefined in daily schedules (without significant changes in user habits).

To sum it up, in this scenario there are special changes in evaluation of decision-making algorithm criteria to meet that every evaluation result triggers training function offloading.

Parameters of scenario are:

$N = 10$ - number of parallel simulations,

$T_{A,off} = 1h$ - time period between regular offloading the AI-based decision-making function by a single device,

$T_{E,off} = 4h$ - time period between regular offloading the evaluation and training functions by a single device,

$T_{test} = 48h$ - overall time period when scenario runs,

$T_T = 10min$ - timeout for offloading the training function,

$T_A = 45s$ - timeout for offloading the AI-based decision-making function,

$T_E = 90s$ - timeout for offloading the evaluation function.

Validation criteria are:

$\eta_{T,res} = 90\%$ - resultant efficiency rate of offloading the training function,

$\eta_{A,res} = 95\%$ - resultant efficiency rate of offloading the AI-based decision-making function,

$\eta_{E,res} = 90\%$ - resultant efficiency rate of offloading the evaluation function.

4. Varying user requirements

Using the laboratory demo environment to offload functions at varying frequency.

Executions run in regular scenarios:

1) Periodic offloading of the AI-based decision-making function and the evaluation function.

2) If needed, trigger the training function (decided by evaluation result).

3) User requirements changes are predefined in daily schedules (without significant changes in user habits).

To sum it up in this scenario there are special changes in user requirements which refers to varying frequency of offloading. During the test scenario there is at least one change in offloading frequency per device and irregular offload action triggered by the user at a random moment in the simulation.

Parameters of scenario are:

$N = 4$ - number of devices using the framework,

$T_{A,off} \in \{15min,\ 30min,\ 45min,\ ...,\ 4h\}$ - time period between regular offloading the AI-based decision-making function by a single device,

$T_{E,off} \in \{4h,\ 8h,\ 12h,\ ...,\ 24h\}$ - time period between regular offloading the evaluation function by a single device,

$T_{test} = 48h$ - overall time period when scenario runs,

$T_T = 10min$ - timeout for offloading the training function,

$T_A = 45s$ - timeout for offloading the AI-based decision-making function,

$T_E = 90s$ - timeout for offloading the evaluation function.

Validation criteria are:

$\eta_{T, res} = 90\%$ - resultant efficiency rate of offloading the training function,

$\eta_{A, res} = 95\%$ - resultant efficiency rate of offloading the AI-based decision-making function,

$\eta_{E, res} = 90\%$ - resultant efficiency rate of offloading the evaluation function.

In conclusion, if the four test scenarios are successfully validated, it means that COGNIT Framework works as expected and the Energy UC has done its contribution to the overall project evaluation.

Test results

In table 5.2 we present values of resultant efficiency rates obtained in individual test scenarios.

Test scenario ID	$\eta_{A, res}$	$\eta_{T, res}$	$\eta_{E, res}$
1	100.0%	-	-
2	99.8%	100.0%	100.0%
3	99.7%	100.0%	100.0%
4	98.7%	84.5%	98.0%

Table 5.2. Results of test scenarios.

As one can see, all but one result are satisfying the validation criteria. The $\eta_{T, res}$ in test scenario no. 4 is slightly too low (observed 84.5%, while at least 90.0% was expected). Some maintenance work was being done to the COGNIT testbed infrastructure just before this test scenario was performed. After these changes to the infrastructure were made we have observed some issues with the scalability. But since prior to that change all the other test scenarios resulted in almost perfect efficiency, we treat this misalignment as a minor mistake, most likely an accidental misconfiguration. The test scenario no. 4 will be rerun before the final review.

Hence, we confirm that the results of the tests indicate that the project has successfully met all established validation requirements. Comprehensive testing across various scenarios has allowed us to assess the functionality of the COGNIT Framework and can confirm the fulfillment of the following capabilities:

- The necessary packages for executing functions have been successfully installed within the images, and the flavor parameter correctly grants access to the appropriate images.
- The computational resources allocated to the Virtual Machines (VMs) are adequate to support the training of AI models.
- Both the C and Python Device Runtime Clients operate effectively with the COGNIT Frontend.
- The C-based Device Runtime Client satisfies the capacity limitation requirements, enabling it to be successfully installed on the smart energy meter.
- The results generated by individual functions are correctly transmitted to the Device Runtime Client, which offloads them within the expected timeframe.
- The connection between the local Device Runtime Client and the COGNIT Frontend remains stable throughout extended demonstrations and is maintained during long-running functions (lasting several minutes).
- There are no issues with communication between functions and external databases for retrieving or saving data and models.
- Test scenarios involving a large number of devices demonstrate that the scaling of VM instances is functioning as intended.

These outcomes validate the robustness and scalability of the system under diverse conditions.

5.5 Technology Readiness outlook and conclusion

Within the scope of UC3, our team successfully defined and verified the concept of enabling a resource-constrained IoT device (an energy meter) to utilize external cloud resources, specifically features of the COGNIT Framework. In this use case, the external resources were employed to run AI-based algorithms designed to determine optimal balancing strategies for household energy consumption and production.

As the framework was developed, the UC3 team built an environment enabling the use of the Function-as-a-Service (FaaS) paradigm on these resource-constrained devices. Initially, we verified Technology Readiness Level TRL 3 using a simulation environment (Python client) calling a simple function.

Following this milestone, we developed an environment allowing the IoT device (using a C-client running on Phoenix-RTOS) to access COGNIT capabilities and call three main functions:

1. Training function for the AI model

2. Decision-making function utilizing the AI model

3. Evaluation function

Laboratory tests confirmed that the solution achieved TRL 4 (Technology validated in a laboratory).

To validate the technology in a relevant environment and achieve TRL 5 (Technology validated in a relevant environment), the test scenarios described in Chapter 5.4 have been executed. This is particularly relevant as real-life solutions require support for multiple

devices, mirroring the scenario where energy meters are present in every flat and household. The results of the tests – though not entirely successful, as described in previous subsection – do confirm that TRL 5 has been reached.

Overall, the developed technology shows great potential for enhancing the abilities of small IoT devices by cognitively migrating workload. This approach not only extends device capabilities but also reduces the amount of data transferred across the network. The added value of this technology can be applied to other sectors where IoT devices lack the resources for complex tasks, such as Industrial IoT.

Future steps toward reaching higher TRLs should focus on meeting the requirements for enabling FaaS execution on millions of IoT devices while cognitively migrating workload to reduce carbon footprint, minimize cost, and improve latency.

6. Use Case #4: Cybersecurity

The smart mobility sector uses innovation and technology to optimize transportation systems, offering services, products, and technologies that improve effectiveness, sustainability, and convenience. This sector is crucial for the EU, driving economic growth, job creation, and environmental sustainability through electric vehicles and public transportation. Technologies like intelligent transportation systems (ITS) and autonomous vehicles enhance transport efficiency.

Key challenges include improving interoperability, developing smart infrastructure (e.g., sensors for autonomous driving), and establishing new regulatory frameworks for safety, liability, and data sharing. Cybersecurity is paramount, as cyberattacks on smart mobility systems can endanger passengers and pedestrians, cause economic loss, and disrupt operations by compromising vehicles and traffic signals. The sensitive data collected by these systems also requires robust protection.

Edge Computing introduces inherent challenges and risks. Located away from secure data centers, edge systems face new threats and require advanced security controls. Given critical latency requirements and unreliable cloud access, security detection and remediation must be autonomous and resilient. Edge systems, despite limited resources, must be capable of self-defense against attacks like denial of service.

Use case description

This Use Case aims to demonstrate an integrated cybersecurity solution for edge computing in smart mobility. The primary focus will be managing vehicle movement and switching between edge nodes while enforcing dynamic, geo-dependent security policies and controls reliably and autonomously.

Benefits of using the COGNIT Framework

In this Use Case, several core features of the COGNIT Framework are leveraged to address the challenges posed by the dynamic environment of a moving vehicle – in our lab simulation experiment: a moving rover – and the need for real-time anomaly detection.

Figure 6.1 presents the high-level integration with COGNIT. The rover (Device Client) collects two data feeds (authentication and GPS logs) to the platform for analysis. Based on the rover’s current location and latency/SLA constraints, the AI-enabled Orchestrator prepares a Serverless Runtime (SR) on the most suitable Edge Node so that the anomaly-detection function executes close to the data source. Results are returned to the rover for local handling and are also exposed on the UC4 dashboard for monitoring. If measured latency or predicted route conditions suggest an SLA (Service Level Agreement) breach at the current node (Edge Node 1), the orchestrator pre-stages a new SR on a more suitable node (Edge Node 2) and seamlessly redirects the next function invocation. This enables proactive placement, low-latency inference, and continuous monitoring without manual intervention.

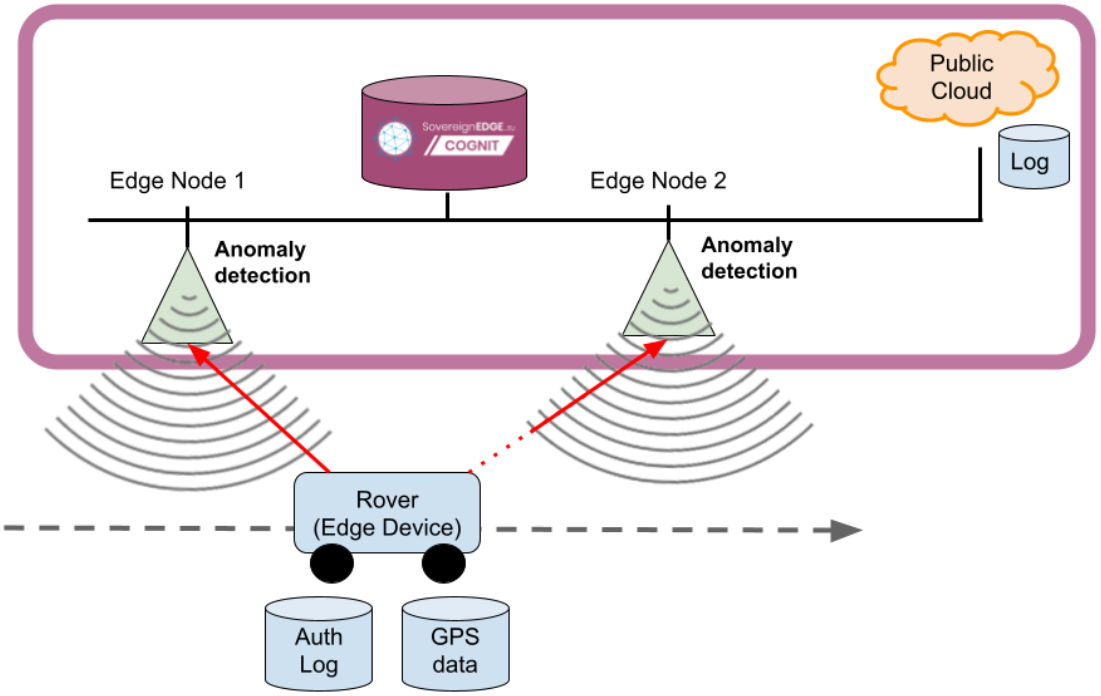


Figure 6.1. High-level architecture for the Cybersecurity Use Case.

One of the most critical features used in this use case is the AI-Enabled Orchestrator's ability to predict and plan the deployment of Serverless Runtimes based on the rover's route and the location of the Device Client (DC). This predictive capability allows for proactive resource allocation, ensuring that the SR is available and ready at the next optimal edge node before the rover reaches that point.

In this Use Case context, the term Service Level Agreement (SLA) is used as a conceptual performance threshold, rather than a contractual agreement, to represent the maximum acceptable latency required to ensure the anomaly-detection function operates effectively

The Device Client plays a crucial role in monitoring latency and ensuring SLA threshold compliance with the selected Edge Cluster (where the anomaly-detection function is executed). If a violation occurs, such as excessive latency, the Device Client can trigger a switch to a more suitable edge node defined by the AI-Enabled Orchestrator.

6.1. Reference Scenario

Use Case 4 of the project focuses on Cybersecurity and highlights the utilisation of the COGNIT Framework through a scenario of a rover (vehicle) using the following Cybersecurity mechanisms:

a. anomaly detection on authentication logs and gps data.
b. enabling Confidential Computing .

Use case Scenario – Anomaly Detection

The cybersecurity case study will use the COGNIT Platform to manage offloading anomaly detection functions to the edge. The Device Client communicates with the COGNIT Frontend to determine a suitable edge cluster for offloading the functions, and maintains a connection with the selected edge cluster, until the Device Client requests a switch to another one. The call to the COGNIT Frontend provides the necessary data for the AI-Enabled Orchestrator to identify the most appropriate Edge Cluster on which to deploy the Serverless Runtime and provide the necessary software stack to execute the anomaly detection functions. The Device Client then offloads the anomaly detection functions to the edge, by passing system logs as parameters to the functions.

As illustrated in Figure 6.2, the data analysed by the functions come from two log sources: authentication logs (the auth.log file) and location logs from the GPS sensor. A watcher observes new entries in these logs and, upon each detection, captures the entry as parameters and triggers the offload of the functions. The functions are executed in the Serverless Runtime via the Edge Cluster Frontend and include:

- **DT Anomaly Detection,** a **D**ecision-**T**ree-based function that compares log entries against patterns to flag potential anomalies;
- **EM Anomaly Detection**, a function based on an **E**mbedded **M**odel that uses vector comparison to confirm or reject the anomaly.

The results returned to the Device Client are then forwarded to the UC4 dashboard in the cloud, which provides real-time monitoring of the Device Client status and visualisation of the function results.

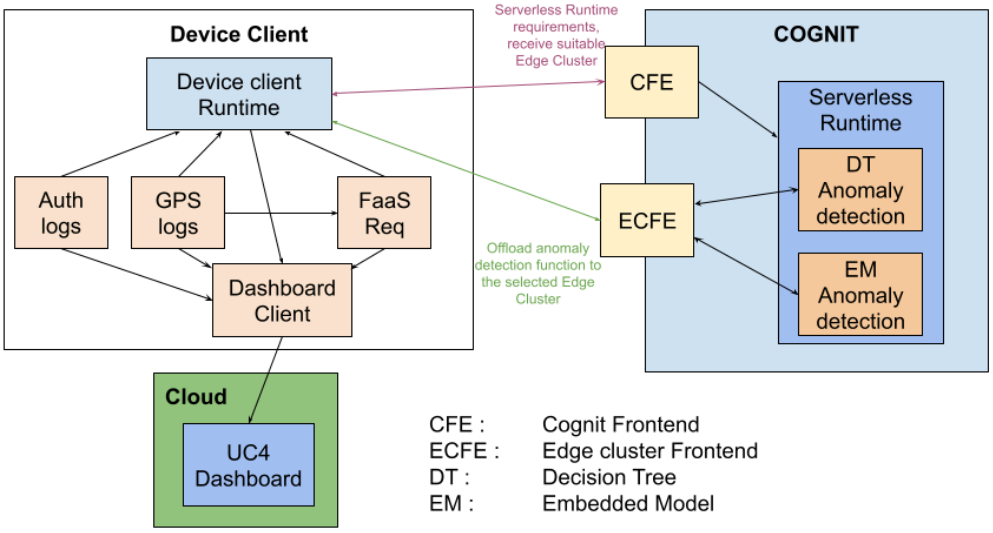


Figure 6.2. Device Client interaction with COGNIT Framework for anomaly detection.

As the rover progresses along the road network, the response time (RT) of the anomaly detection function running on an Edge Cluster must be monitored against its SLA. If RT degrades beyond the SLA, the function should be redeployed to another Edge Cluster that can meet the RT SLA. Based on the latest version of the COGNIT Framework, and as

explained in document D3.5, §2.2, there are two methods for deciding when to connect to another Edge Cluster. The first uses the MAX_LATENCY requirement, which represents the RT SLA.

If MAX_LATENCY is provided, the Device Client's Latency Calculator class is invoked to obtain the latency of the available Edge Clusters. The cluster with the lowest measured latency is selected for offloading the functions.

If MAX_LATENCY is not provided, the mandatory GEOLOCATION requirement is used to select the nearest Edge Cluster Frontend for offloading.

For a device within the COGNIT Framework to transition seamlessly between edge platforms without compromising real-time SLAs, the redeployment time of the anomaly detection function to an alternative Edge Cluster must be known. This responsibility lies with the COGNIT Framework.

The deployment of a new serverless runtime and the anomaly detection functions on the target Edge Cluster must be initiated at the right moment to avoid any interruption in detection and any RT-SLA violation. This imposes the following requirements on the COGNIT Framework:

- It must be able to predict RT-SLA violations.
- It must be able to identify an Edge Cluster with the appropriate resources and their timely availability.
- It must account for the time needed to provision the serverless runtime on the Edge Cluster and start the anomaly detection function.

These three capabilities are covered by the project's requirements and are enforced by the COGNIT Framework's orchestration logic, using a generic, location-aware RT prediction to proactively prepare and, when needed, reconnect to a more suitable Edge Cluster.

Concretely, **UR4.1** (permissible edge nodes by policy) and **UR4.3** (execute as close as possible to the device) drive proactive placement to avoid RT-SLA breaches. **UR4.2** (live migration of data/runtime across edge locations) underpins selecting and preparing an alternative Edge Cluster in time, and the common requirements **UR0.5** (execution across tiers according to latency targets) and **UR0.7** (maximum runtime provisioning time) justify accounting for provisioning and start-up in the decision logic. This keeps the end-to-end handover "under control", predicts a potential violation, pre-places the Serverless Runtime, and switches on the next call without breaching the configured SLA.

The sequence diagram in Figure 6.3 below shows the mechanics behind the switching of the client device from one Edge Cluster to another:

1. On first initialisation, the Device Client authenticates with the COGNIT Frontend and receives a token.
2. The Device Client sends an offload request to the COGNIT Frontend , including the function to execute, its requirements, and the calculated latency.
3. The COGNIT Frontend forwards these details to the AI-Enabled Orchestrator.

4. Using the request details and a location-based RT prediction function, the AI-Enabled Orchestrator determines the most suitable Edge Cluster and prepares the Serverless Runtime for the next invocation.
5. The orchestrator verifies that the chosen Edge Cluster has the right resource types and sufficient capacity for the anomaly-detection function.
6. If resources are available, the orchestrator reserves them and deploys the Serverless Runtime along with the selected function flavour.
7. The orchestrator returns the Edge Cluster Frontend (EC FE) connection details to the CFE, which relays them to the Device Client.
8. The Device Client contacts the Edge Cluster Frontend to execute the function (by ID) and retrieves the function result.
9. Whenever the requirements change (e.g., updated location), the Device Client repeats Step 2 with the new values.
10. Switch on next call. If the orchestrator selects a different Edge Cluster, the Device Client switches to that cluster for the next function invocation.

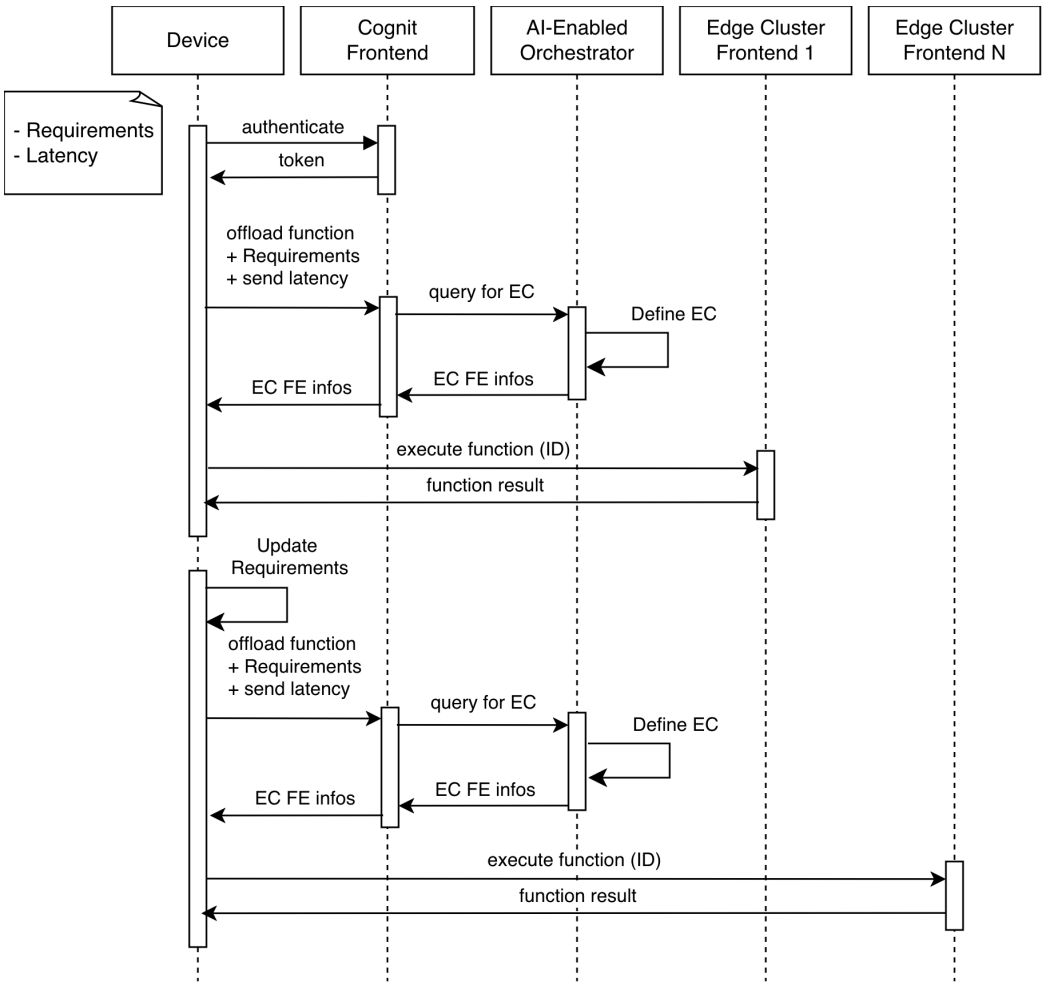


Figure 6.3. Interactions between the edge device and COGNIT Framework for switching Edge Clusters.

As described in the sequence diagram, the Device Client i.e. the rover in this case study, interacts only with the COGNIT Frontend component of the COGNIT Framework. The rest of the interactions required for switching Edge Clusters for the execution of the anomaly detection function are internal to the COGNIT Framework.

In summary the AI-Enabled Orchestrator receives the location of the Device Client and invokes an AI based generic location based RT prediction function to prepare a Serverless Runtime and deploy it on an Edge Cluster with a better RT. When the prediction function indicates that the RT will soon be violated, the AI-Enabled Orchestrator looks for another Edge Cluster and deploys a Serverless Runtime with the anomaly detection function.

The AI-Enabled Orchestrator provides the device via the COGNIT Frontend, which Edge Cluster Frontend it can contact for the next execution of the anomaly detection function. The device can then switch to the new Edge cluster ready for the next execution of this function.

Use case Scenario – Confidential Computing (Execute function in CC enclave)

This demonstration scenario of UC4 illustrates how confidential computing protects memory integrity for sensitive application environments such as autonomous vehicle services.

In order to test and demonstrate the Confidential Computing (CC) support and capabilities of the COGNIT Framework, UC4 has executed specific functions within a CC secured hardware enclave. This involved deploying a Confidential Computing-enabled testbed using the framework with a CC enabled configuration. The CC testing scenario has been included in the UC4 demonstration in order to study it, assess compatibility, performance, and potential benefits. For this scenario to be used, the sensitive function execution requirement must be enabled and passed to the COGNIT Framework. This requirement is a security strategy to enhance data protection and trust during sensitive processing tasks.

For this scenario we are conducting the same test twice: first without protection, and then with it enabled.

Test 1: Confidential Computing Disabled

- **Setup:** We have a terminal open on the host and another in the guest VM.
- **Action (VM):** A Python interpreter is started and a variable is set: `"CETIC_API_KEY=12345azert"`.
- **Action (Host):** A memory-scanning script is run against the VM's process ID, searching for the string "CETIC_API_KEY".
- **Result:** The host script successfully reads the VM's memory and outputs the plaintext data: `CETIC_API_KEY=12345azert”`.

Test 2: Confidential Computing Enabled

- **Setup:** The same two-terminal setup is used.
- **Action (VM):** We assign the same secret: `CETIC_API_KEY=12345azert`.
- **Action (Host):** We run the same memory-scanning script.

- **Result:** The script fails to find the string. The VM's memory is encrypted and inaccessible to the host, demonstrating that the content is protected.

6.2. Objectives and validation criteria

The validation scenario consists of reproducing the conditions during the rover movements, the transitions between edge nodes, and the processing of data via COGNIT. Here are the key elements of the validation process:

1. **Initialising the rover connection to COGNIT:**
   When the rover starts, it establishes a connection with the COGNIT Frontend. This step consists of authenticating the rover.
2. **Deploying the Serverless Runtime in the optimal Edge Cluster:**
   After initialisation, the rover requests the use of a Serverless Runtime where the anomaly detection function will be executed. COGNIT manages the orchestration and deployment of the Serverless Runtime in the most optimal Edge Cluster and using the appropriate Serverless Runtime flavour of UC4. The flavour of a Serverless Runtime refers to the virtual machine image used, which contains all the dependencies necessary for the anomaly detection function execution. In UC4, the flavour defined in the user-defined requirements is named "CyberSecurity".
3. **Executing the anomaly detection functions with collected data:**
   The anomaly detection functions are executed by processing the authentication and GPS data. The last block of lines written in the auth.log and GPS log file is captured and passed as a parameter to the function via the Device Client.
4. **Continuous data flow and function re-execution:**
   During the rover's movement, new data is continuously transmitted to the anomaly detection functions. Each new block of logs triggers the re-execution of the functions. COGNIT must ensure that these data flows are processed in near real-time, while maintaining minimal latency between data collection and function execution.
5. **Edge node switching:**
   As the rover moves, and the latency to reach the designated COGNIT edge node increases above the SLA, COGNIT should orchestrate a switch to another edge node to maintain low-latency processing and respect the SLA.

We have validated these requirements, by running simulations of the rover's operations and interactions, including its movement across different network areas to evaluate the planning and execution of edge node transitions and how they affect the speed of anomaly detection execution. Performance metrics, such as initialization time, execution time, response latency, and node switching efficiency, are measured to ensure that the COGNIT capabilities meet the requirements of the Use Case.

6.3 Summary of activities done during the project

This section summarises the activities performed for UC4 (Cybersecurity) and focuses on the final snapshot achieved by the end of the project. It outlines how we implemented

SLA-aware offloading triggers in the Device Client, packaged and deployed the Anomaly Detection functions as Serverless Runtimes, and validated the end-to-end workflow under mobility. Concretely, we observed changes in both data streams (authentication logs and GPS traces) and in configurable requirements to trigger function offloading, developed two complementary Anomaly Detection approaches (a lightweight rule-based baseline and an embedding-based model) executed in the Serverless Runtime, implemented local response mechanisms to enforce security actions on the rover, and delivered a dashboard for real-time monitoring of detections, connectivity, and Edge Node handovers. We additionally exercised Confidential Computing execution for sensitive functions as part of the validation activities. The following subsections detail each of these elements.

6.3.1 Observe changes (Data and Requirements) -> Trigger the offloading and the execution of the function

Two types of events influence the triggering of offloading and the execution of the anomaly detection function. On one hand, the update of the requirements, and on the other hand, the addition of a new block of lines in the log file by the rover system.

To monitor these changes, we developed and implemented two classes to watch and handle file modifications:

LogHandler:

- Monitors and reads changes in the log file (e.g., auth.log).
- Passes the detected events as parameters to the function.
- Triggers the offloading and execution of the function.

RequirementsHandler:

- Monitors changes in the requirements file (e.g., FLAVOUR, MAX_LATENCY, GEOLOCATION, etc.).
- Dynamically reloads the requirements and applies them so they are taken into account during the next triggering of offloading and function execution.

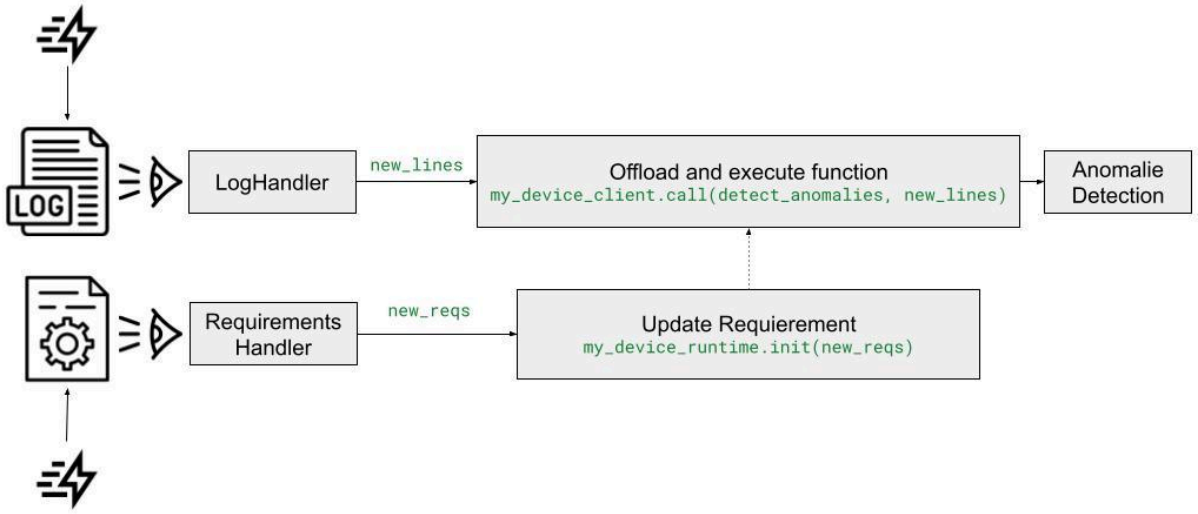


Figure 6.4. Observe changes (Data and Requirements) -> Trigger the offloading and the execution of the function

6.3.2 (Logs, GPS) Data Using M-LLM: From Rule-Based to Embedding-Based Approaches

As soon as the Device Client detects a change in the logs, an anomaly detection request is offloaded to the COGNIT Framework. After discarding Opni, we surveyed lightweight options and tested an AI path (a LanoBERT based PoC, then a local LLM) to validate anomalies from context. Both were too resource intensive for our latency constraint. We therefore adopted two complementary approaches: a fast, deterministic comparison method for first-pass Anomaly Detection and a compact embedded model that provides semantic verification while meeting Edge constraints.

Decision-tree-based anomaly detection

A lightweight, rule-driven baseline that evaluates each event against configurable **Rules** and assigns a **severity** score (for auth logs) or a binary anomaly flag (for GPS) known for its:

- Deterministic, auditable decisions.
- Low compute footprint, easy local tuning.
- Complements the embedding model with explicit policy checks.

Methodology

- **Auth logs.** Each entry is matched against Rules (known users, usual time windows, allowed source IPs) and textual cues (e.g., *Failed password*, *Invalid user*, *NOT in sudoers*).
- **GPS logs.** Each GPS point is compared to the *map-matched expected route*; we flag anomalies when *off-route distance* or *time–position residuals* exceed thresholds. Optional *TOA/TDOA* fusion (Time of Arrival / Time Difference of Arrival) provides an independent position; *GPS vs. TOA/TDOA* discrepancies beyond tolerance are anomalous.

Scoring & outputs

- **Auth logs (severity 0–3):**
  - 0 (normal): routine lifecycle (legitimate login, clean logout, authorized sudo).
    1 (low): minor deviation (unrecognized IP).
  - 2 (medium): contextual deviation (known user outside usual hours).
    3 (high): strong abuse signals (invalid/locked user, *NOT in sudoers*, failures on unknown user).
- **GPS logs (binary):** Anomalous or Normal (no severity gradation).

Each flagged item includes the original line/fix, extracted entities (user/IP or coordinates), timestamp, and either **severity** (auth) or **anomalous**: true/false (GPS), plus a short reason.

Embedding-based anomaly detection

The new model introduces a semantic, embedding-based anomaly detection **system leveraging** "sentence-transformers/paraphrase-MiniLM-L3-v2"**, a compact variant of BERT (MiniLM) known for its:**

- Low computational cost **and** high inference speed on CPU.
- Ability to understand semantic similarity **in both textual logs and behavioral sequences (GPS data).**
- Suitability for embedded or serverless environments.

Embedding-Based Approaches Performance validation

On a standard CPU system, the Transformers library loads in approximately 3.5 s, the model initializes in about 0.43 s, and each subsequent inference takes around 0.01 s. These figures indicate that near real-time anomaly detection is feasible on CPU-only deployments without GPU acceleration. In contrast, preliminary tests with Lanobert and a generic LLM exceeded 10 s per execution, and were therefore not retained for this use case. We focused performance validation on the embedded model, as the rule-based decision-tree is effectively instantaneous and did not warrant a comparative benchmark.

For GPS data, the system learns the embedding "signature" of *normal vehicle behavior* (route regularity, speed range, pauses). It can thus detect *contextual deviations* such as a speed that is reasonable on a highway but unusual in an urban area, or a detour inconsistent with the vehicle's usual delivery pattern.

In summary, this compact LLM-based module merges *semantic intelligence from NLP* with edge computing efficiency, enabling unified anomaly detection across *system logs, GPS trajectories, and operational metrics*, while maintaining minimal computational overhead.

Methodology

- Each log entry or GPS sequence is transformed into a *feature embedding vector* using the Hugging Face feature-extraction pipeline.
- These embeddings are compared to a local repository of labeled reference samples using *cosine similarity*.
- If no repository exists, it is automatically initialized from predefined baseline data (normal authentication events or valid driving zone around an Edge Node).

Scoring & outputs

- **Auth logs (binary):** Anomaly confirmed or refuted.
- **GPS logs (binary):** Anomaly confirmed or refuted.

The architecture is fully CPU-based, lightweight, and has no external dependencies.

6.3.3 Development of a remediation solution based on the results of anomaly detection in authentication Logs.

In this Use Case, an attack can occur from the moment the vehicle starts up and throughout its entire journey. An attacker who manages to connect to the Device Client could inject malicious data to deceive the vehicle's system, potentially causing an accident or rendering the vehicle unusable. To mitigate these risks and maintain strict control over the users and systems authorised to interact with the device client, we have implemented a remediation mechanism that limits unauthorised connection attempts and reacts automatically based on the outputs of the anomaly-detection process.

Remediation refers to all actions taken to correct, neutralise or mitigate a threat following a security event. The remediation solution implementation consists of two primary components operating within the Rover environment:

1. **Device Client**: Monitors authentication logs, sends them to a serverless runtime for analysis, and processes the returned results.
2. **Response Daemon**: Consumes detection events and takes appropriate security actions.

The architecture also includes the Serverless Runtime environment that hosts the Anomaly Detection functionality.

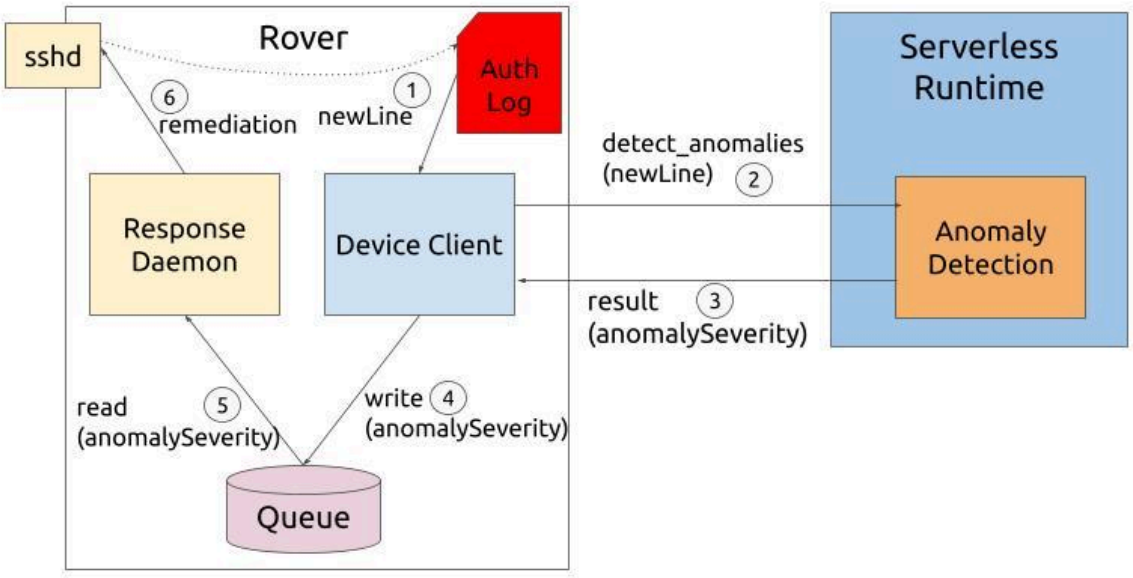


Figure 6.5. Remediation workflow

Workflow

The system operates according to the following workflow:

1. When a new authentication log entry appears, it is detected by the Device Client.
2. The Device Client sends the new log line to the Anomaly Detection function in the Serverless Runtime.
3. After analysis, the Anomaly Detection function returns results to the Device Client, including severity levels for any detected anomalies.

4. If anomalies are detected, the Device Client writes appropriate events with severity information to a Queue.
5. The Response Daemon reads the Queue to obtain anomaly information and severity levels.
6. Based on the severity level, the Response Daemon implements remediation actions to the SSH daemon (sshd).

Response Actions

The Response Daemon implements three types of blocks based on severity levels:

1. User-Specific blocks: Deny access to the user attempting to log in for 10 minutes.
2. IP-specific blocks: Deny access to the user attempting to log in only from specific IP addresses (lower severity) for 10 minutes.
3. Global blocks: Deny access to the user attempting to log in permanently from all IP addresses (higher severity).

Blocks can be temporary (automatically expiring after a set duration) or permanent (requiring explicit unblock), depending on the severity level.

For example, if the following line is analysed, anomaly detection will classify it as a severity 2 anomaly :

```
Shell
Feb 20 01:05:23 server sshd[22346]: Failed password for user1 from 203.0.113.45 port 44445 ssh2
```

Because the user user1 is not allowed to log in outside business hours (08:00 - 16:00, as defined in the rules file), the daemon issues a temporary global block for 10 minutes, denying login for user1 from all IP addresses.

6.3.4 Dashboard Implementation

To enable real-time monitoring and visualisation of the cybersecurity use case, a dedicated web-based dashboard was developed and deployed in the RISE infrastructure. The dashboard serves as the central interface for observing the Device Client status, tracking anomaly detection results, and monitoring the interaction between the vehicle and the COGNIT edge infrastructure.

The dashboard is implemented as a Flask-based web application with Socket.IO for real-time bidirectional communication. The frontend utilises HTML5, CSS3, and vanilla JavaScript, with Leaflet.js for interactive map visualisation. The application follows a modular architecture where the backend receives telemetry data from the Device Client via WebSocket connections and broadcasts updates to connected dashboard clients in real-time.

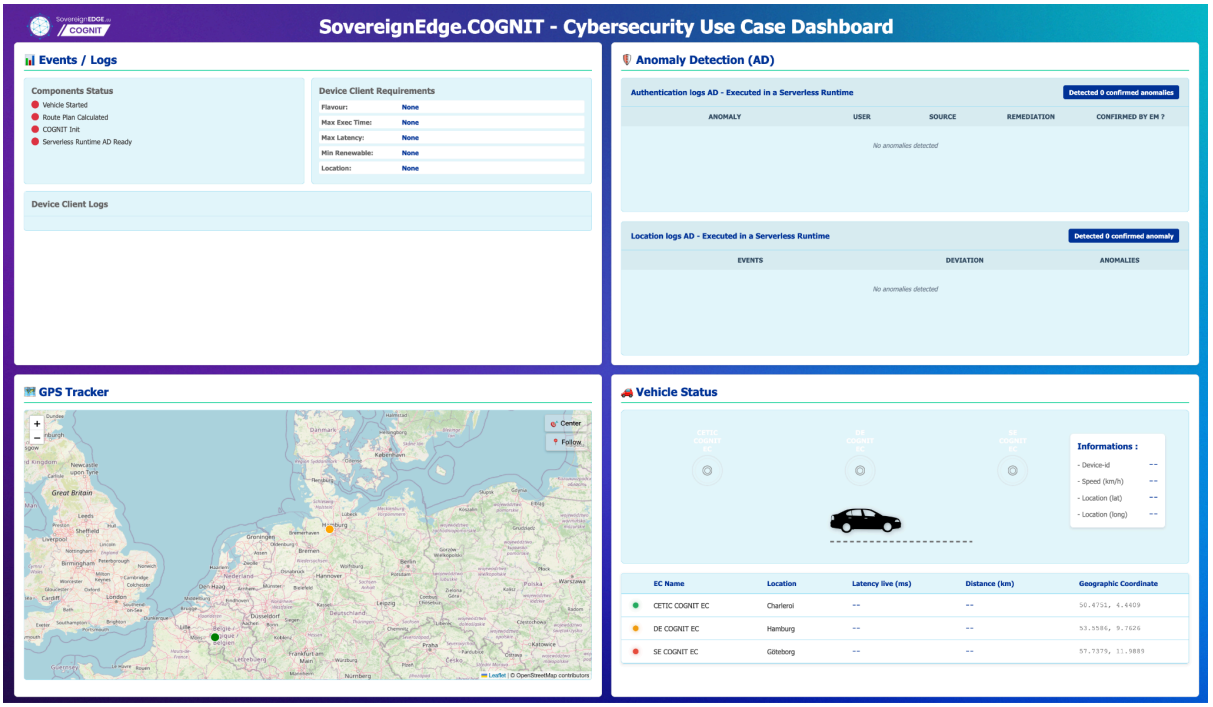


Figure 6.6. UC4 Dashboard overview

As shown in Figure 6.6, the dashboard is organised into four main panels, each addressing a specific monitoring requirement.

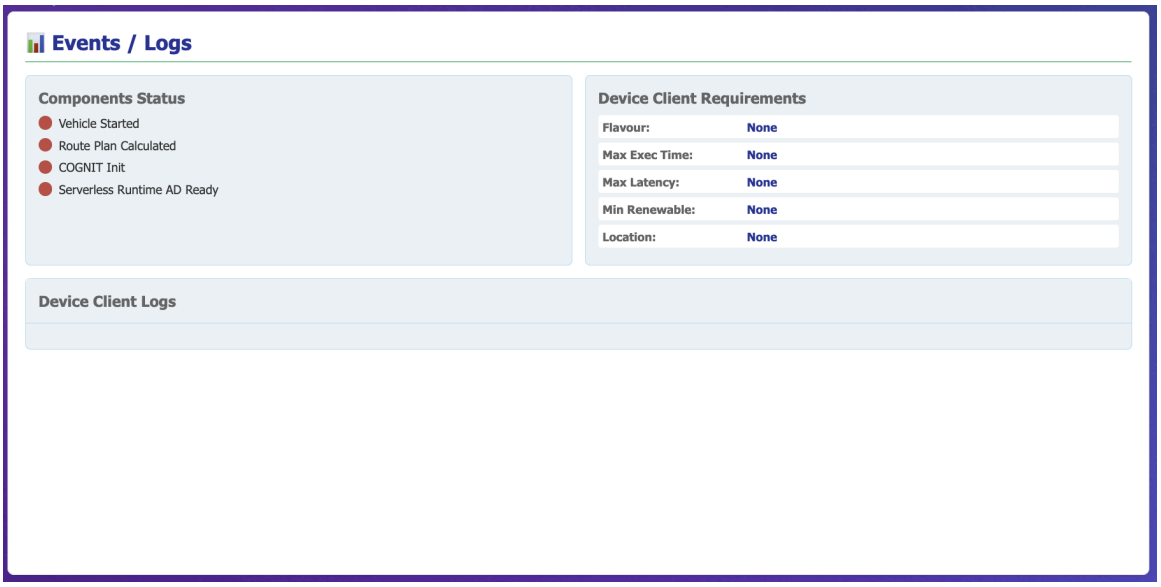


Figure 6.7. UC4 Dashboard Events/Logs Panel (top-left)

Figure 6.7 displays the operational status of system components (Vehicle Started, Route Plan Calculated, COGNIT Init, Serverless Runtime AD Ready) through color indicators. It also presents the Device Client requirements (FLAVOUR,

MAX_FUNCTION_EXECUTION_TIME, MAX_LATENCY, MIN_ENERGY_RENEW, GEOLOCATION) and streams real-time device logs.

Figure 6.8. UC4 Dashboard Anomaly Detection Panel (top-right)

Figure 6.8 shows two distinct sections for visualising anomaly detection results. The Authentication Logs AD section displays a table with detected anomalies, including severity badges (Medium, High, Critical), user information, source IP addresses, remediation actions, and confirmation status from the Embedded Model function. The Location Logs AD section shows GPS deviation events and their anomaly classification status.

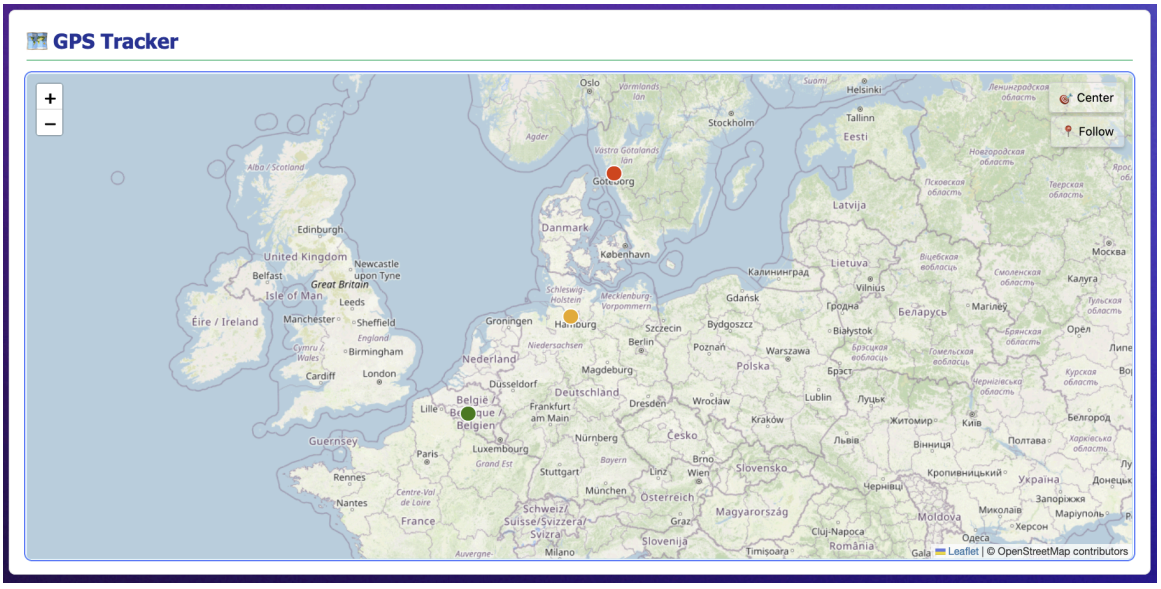


Figure 6.9. UC4 Dashboard GPS Tracker Panel (bottom-left)

Figure 6.9 presents an interactive map, displaying the vehicle's real-time position, the calculated route and the Edge Cluster locations with markers.

Figure 6.10. UC4 Dashboard Vehicle Status Panel (bottom-right)

Figure 6.10 visualises the vehicle's connectivity to edge clusters, based on real-time distance and latency calculations. The panel displays key vehicle information (Device ID, speed, coordinates) and maintains a comprehensive table of Edge Clusters showing their latency measurements, distances, and geographic coordinates.

6.3.5 Lessons learned

During the COGNIT Project, several lessons were learned during the development and integration cycles with the Framework.

A lightweight, rule-driven decision-tree remains effective for first-pass sorting of authentication and GPS events to ensure timely responses, while an embedded confirmation model provides a second look, although the confirmation was initially designed with an LLM, an embedded approach proved more efficient for edge execution and sufficiently accurate for our needs.

Closed-loop remediation can be made both feasible and scalable. By reconfiguring the sshd policy based on anomaly detection outcomes, it is possible to set IP-specific or global blocks (temporary or permanent) and deliver severity-aware automated responses on the rover, without operator intervention, as a first line of defense.

The security requirements at the network edge are compatible with serverless offloading. Despite constrained hardware and mobile connectivity, the UC4 design preserves autonomy through local monitoring and remediation while using Serverless Runtimes for compute-intensive analysis and validating an edge-centric cybersecurity posture for mobility.

To improve reproducibility across trials, SLA policies (e.g., MAX_LATENCY and switching) should be formalised in a configuration, making the DC hand-over behavior deterministic and traceable.

6.3.6. Follow up on Risks and Mitigation Plan

Nº Risk	Potential risks		Contingency plan		Comments
	Level (1:low; 5:high)	*Impact*	*Description*	*Responsible*	
1	3	SLA/latency breach during mobility	Use the Device Client's SLA-aware handover and the AI-Enabled Orchestrator to pre-place SRs and switch on violation, define MAX_LATENCY thresholds and switching hysteresis in configuration	Technical Lead	The architectural mechanism used to alleviate the problem is described in §6 (DC + Orchestrator + SR pre-placement).
2	4	Insufficient connectivity with local network providers	Subscribe and use the QoS feature on the provider's 5G connections to guarantee minimum connectivity.	Project Lead	Risk not materialised
3	3	Device resource limits	Keep lightweight actions on-device (local watchers + remediation), offload heavy anomaly detection to SR, prefer a low-footprint Device Client.	Technical Lead	The risks are mitigated by the UC4 design: local watchers/remediation + SR offload.

4	4	Execution time of the embedded-model AD function	Keep the model lightweight so function execution < MAX_FUNCTION_EXECUTION_TIME and overall response < MAX_LATENCY on target hardware.	Technical Lead	The initially planned approach was replaced with an embedded model to meet performance targets. (EM/DT functions are defined in §6.3.1.)
5	3	Scalability of SR deployments/handovers	Predict maximum offload from expected vehicles, oversize lab resources, use virtualisation to test at scale, and iterate on bottlenecks. (Approach validated in other UCs.)	Technical Lead	Risk not materialised
6	4	Dashboard exposed to the Internet	Secure the dashboard by following cybersecurity best practices. Enforce authenticated access, and keep real-time streaming confined to required channels.	Technical Lead	The OWASP Top 10 guidelines were followed to ensure the security of the dashboard located in the cloud.
7	3	Delays in activating confidential computing and processing encrypted data.	Stage delivery (non-CC path first, then CC), and measure execution/performance of CC deployments versus baseline to inform prediction/placement. (CC scenario is defined in §6.1.) Design the resources and optimise the prediction.	Project Lead	Risk not materialised

6.4. Use Case Infrastructure, Demonstration, and Validation

6.4.1 Use case demonstration and validation

The two demonstration scenarios are handled separately. On the one hand, there is anomaly detection, and on the other hand, confidential computing. Anomaly detection requires multiple edge clusters to demonstrate the framework's ability to manage the switch from one edge cluster to another. Confidential computing requires specific hardware compatible with AMD SEV to encrypt the data within the serverless runtime and make it invisible from the outside.

6.4.1.1 Anomaly detection

We simulate a rover travelling from CETIC (Charleroi, Belgium) towards one of the RISE sites (Gothenburg, Sweden). The Device Client (DC) authenticates to the COGNIT Frontend and begins offloading anomaly-detection (AD) requests. While the rover “moves,” the framework evaluates SLA/latency and distance to available Edge Clusters and instructs the DC to switch on the next invocation when a better target exists. During the trip, an external actor attempts to connect, generating authentication and GPS events that feed the AD pipeline. The dashboard provides continuous situational awareness (initialisation, logs, device requirements, anomalies, and EC status).

Demonstration scenario

- **Given** the rover is authenticated to the COGNIT Frontend and the required SR for AD is deployed on the selected Edge Cluster,
- **When** new auth.log are produced and an attack attempt occurs during movement,
- **Then** authentication and location anomalies are detected and reported on the dashboard, appropriate remediation is applied on the rover, and the DC switches Edge Cluster on the next invocation when SLA would be violated (as per configured MAX_LATENCY and switching hysteresis).

Validation Criteria

- Anomalies are detected, reported, and remediated (IP-specific or global, temporary or permanent).
- The COGNIT Framework manages EC switching across the path (e.g., CETIC -> Hamburg -> Gothenburg) based on SLA and geolocation.
- Dashboard shows Initialisation, Logs, Requirements, Anomalies, and Edge Cluster views :
    1. Events/Logs with component state and device-provided requirements;
    2. Authentication and GPS anomaly tables (severity and remediation status);
    3. Live map with route and EC markers;
    4. Vehicle/EC status panel listing measured latency and distance to each EC.

6.4.1.2 Confidential computing

This scenario illustrates memory-protection guarantees with and without AMD SEV enabled. We run the same workload twice inside an SR on a CC-capable host.

Demonstration scenario

This scenario evidences memory-protection guarantees when executing a function inside a Serverless Runtime (SR) on an AMD SEV–capable Edge Cluster. We run the same workload twice, first with Confidential Computing (CC) disabled, then with CC enabled while a host-side memory script scans to read a known secret from the guest VM that hosts the SR.

Phase 1 - CC disabled :

- On the guest VM (inside the SR), start a Python interpreter.
- Offload and execute a function that sets : secret="azert12345".
- On the host (Edge cluster), run the memory-scanning script against the VM's process ID searching for secret=.
- The script outputs the plaintext secret="azert12345".

Phase 2 - CC enabled :

- Repeat the same steps with CC enabled on the VM / host.
- The script fails to find the plaintext; guest memory is encrypted and unreadable from the host.

Validation Criteria

- A function runs in an SR on a local Edge Cluster without CC; host-side read of guest memory succeeds.
- The same function runs in an SR on a CC-capable Edge Cluster, host-side read of guest memory fails.
- Artifacts capture:
    - Host scanner output (success vs failure),
    - SR launch parameters indicating CC state,
    - Guest-side console showing the secret assignment.

6.4.2 Use case internal testbed

The UC4 internal testbed connects a rover-simulator device to the COGNIT Frontend and to two independent Edge Cluster nodes. One node, hosted at CETIC, is used for day-to-day integration and functional checks; a second, bare-metal node at OVH is reserved for confidential-computing (CC) experiments and end-to-end tests.

From an integration standpoint, the testbeds are wired into the project's COGNIT continuum: the Edge Cluster Frontends are reachable from the device over the same secure networking used elsewhere, and they are registered so that the COGNIT Framework can expose Serverless Runtimes to the Device Client when required.

Networking is kept simple but realistic. The device authenticates to the COGNIT Frontend and communicates with the selected Edge Cluster Frontend over routable addresses, while the cluster side provides the usual virtual networks and inbound access for SR deployment and invocation.

To preserve the mobility and latency characteristics of the demonstration without access to a 5G slice in this phase, we deliberately abstracted the radio layer and operated over wired IP links. The goal was to keep the semantics of mobility (geolocation, SLA targets, and handover triggers) while ensuring that end-to-end latencies stayed within a credible range. We therefore used geographically proximate clusters so that baseline RTT and throughput were comparable to what a moving device would experience under good 4G/5G coverage. The DC announces its geolocation and SLA constraints to the orchestrator, which makes placement decisions accordingly. This preserves the behaviour we needed to validate (pre-placement of SRs and switch on next call when the configured latency threshold would be exceeded) without depending on a specific Radio Access Network (RAN) technology.

The two local Edge nodes are used in tandem to exercise the switching logic and the CC path within the same bench. Both nodes (CETIC and OVH) are connected to the project's main Edge Cluster at RISE, yielding a three-node topology. Along the simulated route, the orchestrator first indicated a better target and the DC switch from CETIC to OVH on the next invocation, later in the run, as conditions evolved, a second decision promoted RISE, and a new switch was performed from OVH to RISE. This two-step progression lets us observe the full loop (decision, switch on the next call to the new EC, and dashboard confirmation) under controlled load and connectivity, while keeping the application workload unchanged.

The OVH node serves as the CC target. When the confidential-computing flag is enabled for a function, the orchestrator restricts candidate clusters to CC-capable infrastructure and prepares the appropriate SR flavour on OVH; when the flag is disabled, the same function can be scheduled on CETIC, OVH, or RISE. Using an identical workload and client path across CC and non-CC runs isolates the effect of memory protection on observability (host-side reads) and on execution overhead, while keeping the validation path consistent with the demonstration described above.

Hardware configuration and specification

CETIC host (virtualised)

- 2x Intel(R) Xeon(R) Gold 5317 CPUs @ 3.00GHz (12 cores, 24 threads per CPU).
- 256GB RAM.
- Approximately 40TB of network-attached storage configured on TrueNAS.
- Networking resources with two 1Gbps NICs and four 10Gbps NICs.

COGNIT Edge Cluster node (VM on CETIC host) - cetic-node1

The COGNIT edge cluster node operate with the following specifications:

- 8 CPU cores
- 16GB RAM
- 100GB disk space

OVH host for confidential computing (bare-metal) - cetic-node2-cc

- 48-core AMD EPYC "Genoa"
- 128 GB RAM

- 2× 1 TB SSD (RAID-1)

Rover simulator (Device Client VM)

The Device Client VM is used as a rover simulator. It hosts the response daemon and the log simulator. The rover simulator has been virtualized and deployed on a separate hypervisor, with resources aligned to mimic the specifications of a Raspberry Pi 4, with:

- 4 CPU cores
- 16 GB RAM
- 32 GB disk

The COGNIT client framework is deployed inside a Docker container on this VM, and includes the various components previously described.

6.5 Technology Readiness outlook and conclusion

UC4 progressed from TRL 2 - 3 to TRL 5 within the COGNIT project. We moved from concept and early prototypes to an integrated system validated in a relevant environment: the components of the COGNIT Framework operated together on a multi-edge topology. We demonstrated anomaly detection with closed-loop remediation, SLA-driven handover with “switch on next call,” and the same serverless workload executed with and without confidential computing to demonstrate data-in-use protection. The internal testbed, dashboard evidence, and reproducibility parameters substantiate the TRL-5 claim.

COGNIT’s contribution has been decisive: edge-aware orchestration and serverless offload kept device code lightweight while preserving local autonomy; deterministic handover maintained latency targets during mobility; confidential computing added a credible protection layer for sensitive workloads on shared edge infrastructure. Together these capabilities made the use case’s cybersecurity pipeline feasible for mobile scenarios without sacrificing control or security.

To lift readiness further (towards TRL 6–7), we will extend validation from controlled benches to operational pilots. First, we will run tests on a real vehicle, followed by several vehicles simultaneously, to characterise end-to-end behaviour and fleet-level observability under authentic motion and coverage dynamics. Second, we will replace wired connection with managed 4G/5G. Third, we will address scalability by driving higher request rates, more concurrent SRs, and Edge Clusters, establishing envelopes for Orchestrator decision time, SR start time, and DC behaviour under load. In parallel, we will evolve the dashboard from a single-device focus to fleet views, and the dashboard endpoint will be refactored from a simple Internet-exposed service to a production-grade ingress (e.g., WAF - Web Application Firewall / CDN - Content Delivery Network, rate-limits, fine-grained authorisation) sized for sustained load.

The Embedded Model (EM) confirmation path will mature from a simple database to durable an S3-compatible store (e.g., MinIO) with versioned artefacts, promotion/rollback, and audit trails across edges; we will also consider federated learning so models improve while sensitive data remains local. These steps would close the gap between demo-grade analytics and operations-grade learning pipelines.

In conclusion, UC4 now stands as an integrated, relevantly validated solution (TRL 5). The path to a higher level of maturity will prioritize pilot testing on real vehicles, fleet-scale operation, radio-realistic deployments, production-grade observability, and an industrialised EM pipeline positioning COGNIT for operational impact and cross-sector value.

PART II. Software Integration and Verification

7. Software Integration Process and Infrastructure

This section describes the software integration process and infrastructure deployment for the COGNIT Framework. COGNIT OpsForge is the deployment automation tool used for this process. It sets up the Cognitive Serverless Framework for the Cloud-Edge Continuum, by using Terraform and Ansible automation built on top of the OpenNebula one-deploy[59] project. OpsForge can target either a supported cloud provider or on-premises hosts.

Developed throughout the COGNIT project lifecycle, it addresses the challenge of deploying the framework components across public cloud or on-premises infrastructure, covering infrastructure provisioning, software installation, and service configuration so that organizations can deploy the COGNIT Framework in a few minutes.

OpsForge aligns with the final COGNIT architecture (as defined in D2.6), which consists of five main components organized in two logical groups: the Distributed Serverless Model (Device Client, COGNIT Frontend, Edge Cluster) and the Cognitive Cloud-Edge Module (Cloud-Edge Manager, AI-Enabled Orchestrator).

OpsForge automates

1. the deployment of the control plane components—COGNIT Frontend, Cloud-Edge Manager, and AI-Enabled Orchestrator—ensuring proper integration, configuration, and operational readiness (see Section 7.1);
2. the deployment of Edge Clusters and configuration of their services (Edge Cluster Frontend and Serverless Runtimes) so that devices can offload functions (see Section 7.2).

OpsForge can be used also to create custom images for Serverless Runtimes using KIWI (see Section 7.3).

7.1 Deployment of COGNIT Control Plane

7.1.1 Architecture and Design

OpsForge is implemented as a Ruby command-line interface (CLI) application that orchestrates multiple infrastructure-as-code and configuration management tools. The tool follows a three-phase deployment model:

1. **Infrastructure Provisioning**: Uses Terraform to provision compute resources, networking, and storage on target infrastructure (AWS EC2 or on-premises hosts).

2. **Software Installation and Configuration:** Uses Ansible playbooks to install, configure, and integrate the control plane components: Cloud-Edge Manager, COGNIT Frontend, AI-Enabled Orchestrator, and supporting services.

[59] https://github.com/OpenNebula/one-deploy

3. **Content Population**: Populates the Cloud-Edge Manager with predefined resources in the default datastore: images for Edge Cluster Frontend and Serverless Runtimes, VM templates for Edge Clusters and Serverless Runtimes, and OneFlow Service Templates for Edge Clusters (containing both Edge Cluster Frontend and Serverless Runtime components).

The deployment architecture creates a control plane subnet hosting four core components:

- **Ingress Controller**: On AWS deployments, OpsForge provides a dedicated ingress host that serves as an Nginx reverse proxy with SSL termination, a NAT gateway, and an SSH jump host for the private control plane subnet. On on-premises deployments, no ingress host is provisioned; components are accessed directly on their respective hosts.
- **Cloud-Edge Manager:** A customized OpenNebula 7.1.80 that manages the distributed cloud-edge infrastructure. This customized version includes COGNIT-specific features (Biscuit authentication driver, Scaphandre energy monitoring integration) that are not available in the upstream OpenNebula distribution.
- **COGNIT Frontend:** The main entry point for Device Clients, deployed as a FastAPI application. OpsForge configures the COGNIT Frontend with a SQLite database for device-to-cluster assignment caching, Biscuit token generation, and integration with the Cloud-Edge Manager.
- **AI-Enabled Orchestrator:** The intelligent orchestration component that uses AI/ML models for workload prediction and resource optimization. OpsForge deploys the orchestrator with its multi-cluster and cluster optimizers, device load prediction configuring them to interact with the Cloud-Edge Manager and COGNIT Frontend.

7.1.2 Deployment

OpsForge supports **two deployment modes**, each optimized for different infrastructure scenarios:

- **AWS Cloud Deployment:** When deploying to Amazon Web Services, OpsForge automatically provisions a Virtual Private Cloud (VPC) with Internet Gateway, a public subnet, security groups, and an EC2 instance for the COGNIT Frontend host. The Terraform module creates a single EC2 instance with a public IP, appropriate instance type, storage volume, and network configuration. The deployment can target any AWS region where the specified instance type is available.
- **On-Premises Deployment:** When deploying to private datacenters or managed infrastructure, OpsForge uses a pre-existing host specified in the deployment template. This host must have Ubuntu 24.04 installed, root SSH access with authorized keys, and network connectivity to any edge hosts that will be provisioned.

OpsForge uses **YAML template files** that define deployment parameters. The template schema enforces validation and provides clear documentation of all configurable options.

The deployment process is initiated with a single command:

```shell
./opsforge deploy <opsforge_template.yaml>
```

OpsForge performs the following operations:

1. **Template Validation**: Validates the YAML template against the JSON schema, ensuring all required fields are present and data types are correct.

2. **Dependency Verification:** Checks that all required tools are installed (Ruby, Terraform, Ansible, AWS CLI if deploying to AWS) and that the `one-deploy` git submodule is present.

3. **Infrastructure Provisioning** (AWS only): Executes Terraform to create VPC, subnets, security groups, and EC2 instances. Terraform state is maintained locally for subsequent operations (updates, deprovisioning).

4. **Ansible Playbook Execution**: Runs the main playbook for the control plane deployment which:
    - Configures the COGNIT package repository on the frontend host
    - Installs the Cloud-Edge Manager (OpenNebula 7.1.80)
    - Creates a Python 3.12 virtual environment with locked dependencies for COGNIT services
    - Installs the COGNIT Debian packages and enables their systemd services
    - Replaces the default OpenNebula marketplaces with the Cognit Marketplace, which contains the OneFlow service templates for each use case
    - Registers the geo-carbon hook for geolocation-aware energy monitoring
    - Configures the Cognit Marketplace that will contain all the OneFlow service for each use case

5. **Output Generation:** Provides connection information including:
    - Frontend host IP address or hostname
    - Cloud-Edge Manager credentials (oneadmin and the configured password)
    - URLs for web interfaces and APIs (Sunstone, FireEdge, COGNIT Frontend)
    - SSH access instructions

In the following two examples of deployment templates.

For AWS deployment:

```
None
:infra:
  :aws:
    :region: "eu-central-1"
    :ssh_key: "cognit-deploy-key"
    :ec2_instance_type: "t3.large"
    :volume_size: 125
:cognit:
  :one_pass: "mypassword"
```

For on-premises deployment:

```
None
:infra:
  :hosts:
    :frontend: "172.20.0.4"
:cognit:
  :one_pass: "mypassword"
```

OpsForge deploys components according to the final COGNIT architecture (D2.6), ensuring proper integration:

- **Device Client:** Not deployed by OpsForge (installed on end-user devices), but OpsForge ensures the COGNIT Frontend is properly configured to accept Device Client connections.

- **COGNIT Frontend:** Installed as Debian package with SQLite database for device-to-cluster assignment caching, Biscuit token generation, and integration with the Cloud-Edge Manager for application requirements and function storage.

- **Cloud-Edge Manager:** OpenNebula 7.1 with COGNIT-specific extensions including the geo-carbon hook for geolocation-aware energy monitoring and the Cognit Marketplace containing OneFlow service templates for all use cases.

- **AI-Enabled Orchestrator:** Installed as Debian package on the same host as the Cloud-Edge Manager. Reads device assignments from the COGNIT Frontend database and updates them asynchronously based on optimization algorithms for multi-cluster and in-cluster scheduling. It will scale the amount of VMs within each cluster if needed for workload optimization.

After initial deployment, administrators must set up the Provider Catalog in the Cloud-Edge Manager to define cloud provider credentials and infrastructure templates for Edge Cluster provisioning.

7.2 Provisioning of Edge Clusters

After the control plane is deployed, OpsForge provisions Edge Clusters on cloud providers or on-premises infrastructure. Edge hosts are pre-configured and added as KVM compute nodes through the Cloud-Edge Manager's provisioning workflow.

OpsForge automatically registers the COGNIT Marketplace in the Cloud-Edge Manager during the control plane deployment. For each Edge Cluster, OpsForge exports a OneFlow service from this marketplace based on the selected flavour. Each flavour corresponds to a specific use case (SmartCity, WildFire, Energy, Cybersecurity) and contains both an Edge Cluster Frontend and a Serverless Runtime (FaaS) role. If the service template and its associated VM templates and images are not already present in the Cloud-Edge Manager, they are automatically downloaded from the marketplace during the export. OpsForge then instantiates the service and waits for all VMs to reach RUNNING state.

Once the service is running, OpsForge configures an Nginx reverse proxy on the edge host to expose the Edge Cluster Frontend API and updates the cluster template with discovery metadata (endpoint URL, flavour, provider) so the control plane can manage the edge cluster. At this point, the Edge Cluster is fully operational and devices can begin offloading functions without any additional configuration.

Edge cluster provisioning is performed after the control plane is deployed, using:

```shell
./opsforge deploy-edge <edge_template.yaml>
```

In the following an example for deploying an Cluster on on-premise hosts:

```
:infra:
  :hosts:
    :frontend: "172.20.0.4"
:cognit:
  :flavour: "Energy"
  :provider: "MyProvider"
  :edge_host_ips:
    - "172.20.0.79"
```

The `:flavour:` field selects the use case, `:provider:` is stored as cluster metadata and used by the COGNIT Frontend to select eligible clusters that meet device offloading requirements (e.g., matching the requested use case flavour, provider), and `:edge_host_ips:` lists the IP addresses of the on-premises hosts that will be added as

KVM compute nodes. These hosts must have root SSH access and be reachable from the frontend host.

7.3 OpsForge KIWI Integration

Serverless Runtime appliances are specialized virtual machine images containing pre-configured operating systems, Serverless Runtime software, and use-case specific libraries. Creating these appliances manually is time-consuming and error-prone, requiring expertise in operating system installation, package management, service configuration, and image optimization. OpsForge integrates with openSUSE KIWI, an appliance building framework, to automate the complete Serverless Runtime image creation process.

KIWI (Linux System Appliance Builder) is an open-source project designed for building ready-to-use operating system images that include pre-configured applications for specific use cases. KIWI uses XML-based description files to define the appliance structure: base operating system, installed packages, configuration files, and post-installation scripts. KIWI builds complete, bootable images that can be deployed directly to virtualization platforms without additional configuration.

7.3.1 Architecture

OpsForge's KIWI integration consists of:

- **KIWI Description Files:** XML files defining the Serverless Runtime appliance structure, stored in `ansible/roles/kiwi/files/serverless_runtime/`. These files specify:
  - Base operating system (openSUSE Leap)
  - Required packages (Python runtime, FastAPI, RabbitMQ client libraries, Prometheus exporter)
  - Serverless Runtime software installation
  - Network configuration
  - Systemd service definitions
  - Post-installation scripts
- **Ansible Playbook:** The `kiwi.yaml` playbook orchestrates the KIWI build process on a remote SUSE host. The playbook:
  - Verifies the host is running openSUSE
  - Installs KIWI and required dependencies
  - Copies KIWI description files to the build host
  - Executes KIWI to build the appliance
  - Converts the raw image to qcow2 format (compatible with OpenNebula/KVM)
  - Reports the output image path
- **Build Command:** The `build_sr` command in the OpsForge CLI triggers the KIWI build process:

```
Shell
./opsforge build_sr <host> [sr_version] [jumphost] [flavour]
```

Parameters:

- `host`: Hostname or IP address of the SUSE machine that will build the appliance (must have ~10 GB free disk space)
- `sr_version` (optional): Version tag for the Serverless Runtime software
- `jumphost` (optional): SSH jump host if the build host is not directly accessible
- `flavour`: Use case flavour identifier for flavour-specific builds

7.3.2 Build Process

The KIWI build process executes the following steps:

1. **Host Preparation**: Ansible verifies the build host is running openSUSE, installs KIWI and dependencies (qemu-img, libvirt), and creates output directories.
2. **Image Description Deployment:** KIWI description files are copied to the build host, including:
    - `appliance.kiwi`: Main KIWI XML description defining the appliance structure
    - `config.sh`: Configuration script executed during image build
    - `disk.sh`: Disk partitioning and filesystem setup script
    - Network configuration templates
    - Systemd service unit files
3. **KIWI Build Execution:** KIWI processes the description files to:
    - Bootstrap a minimal openSUSE installation in a chroot environment
    - Install all specified packages and dependencies
    - Execute configuration scripts
    - Create the root filesystem
    - Generate a bootable disk image
4. **Image Conversion:** The raw disk image is converted to qcow2 format using `qemu-img`, creating a compressed, copy-on-write compatible image suitable for OpenNebula.
5. **Output Delivery:** The final qcow2 image is available on the build host at `/root/kiwi-image/output/cognit-sr.x86_64-<version>.qcow2`. The image can be uploaded to OpenNebula image datastores or distributed to Edge Clusters.

The final COGNIT architecture supports multiple Serverless Runtime flavours, each tailored to specific use cases with pre-installed libraries and tools. OpsForge's KIWI integration can build flavour-specific appliances by:

- Specifying the `flavour` parameter in the `build_sr` command
- Using flavour-specific KIWI description files that include use-case-specific packages
- Configuring the Serverless Runtime software with flavour identifiers during build

This enables organizations to create optimized Serverless Runtime images for each use case, reducing VM instantiation time and ensuring consistent runtime environments.

7.3.3 Integration with Deployment Process

KIWI-built Serverless Runtime images are integrated into the OpsForge deployment process:

1. **Image Building:** Administrators use OpsForge's KIWI integration to build Serverless Runtime images for different use case flavours.

2. **Service Template Creation:** Serverless Runtime images are packaged together with Edge Cluster Frontend into OneFlow Service Templates, with separate templates for each use case (SmartCity, WildFire, Energy, Cybersecurity).

3. **Automatic Population:** During the OpsForge deployment content population phase, these service templates, along with their associated images and VM templates, are automatically populated into the Cloud-Edge Manager's default datastore, making them immediately available for instantiation.

4. **Instantiation:** After Edge Clusters are provisioned, administrators can directly instantiate the pre-populated service templates through the Cloud-Edge Manager, as all required resources are already available and ready for use.

7.3.4 Operational Benefits

The KIWI integration provides several operational advantages:

- **Reproducibility:** KIWI description files are version-controlled, enabling exact reproduction of Serverless Runtime images across different build hosts and time periods.

- **Consistency:** All Serverless Runtime instances use identical base images, ensuring consistent behavior and reducing configuration drift.

- **Automation:** The complete build process is automated, eliminating manual steps and reducing human error.

- **Optimization:** KIWI builds minimal, optimized images containing only required packages, reducing image size and boot time.

- **Maintainability:** Updates to Serverless Runtime software or dependencies require only modifying KIWI description files and rebuilding.

7.4 Fourth Version of the COGNIT Software Stack

The final version (4.0) of the COGNIT Software Stack (aligned with the architecture defined in D2.6) represents the culmination of six research and innovation cycles (M1-M36), incorporating all enhancements, optimizations, and architectural refinements developed throughout the project. This version fully implements the distributed serverless model for the cloud-edge continuum, with intelligent orchestration, comprehensive monitoring, and automated deployment capabilities.

The software stack consists of seven core components, each fulfilling specific roles in the COGNIT Framework architecture:

Distributed Serverless Model Components:

- **Device Client:** Lightweight SDKs (Python and C) that enable devices to offload function executions to the cloud-edge continuum. The Device Client handles authentication with the COGNIT Frontend, manages application requirements, uploads functions and data, and executes functions through synchronous requests to Edge Cluster Frontends.

- **COGNIT Frontend:** The main entry point for the COGNIT service, implemented as a FastAPI microservice. It authenticates devices using Biscuit token-based cryptography, stores application requirements and functions in the Cloud-Edge Manager document pool, and provides intelligent Edge Cluster assignment through a database-mediated caching mechanism that enables low-latency responses while the AI-Enabled Orchestrator asynchronously optimizes assignments.

- **Edge Cluster Frontend:** Deployed per Edge Cluster, this FastAPI service acts as the interface between devices and Serverless Runtimes. It implements a queue-based architecture using RabbitMQ for internal request dispatching, while presenting a synchronous API to devices. The Edge Cluster Frontend provides intelligent scaling capabilities, enabling dynamic adjustment of Serverless Runtime cardinality based on workload demands.

Cognitive Cloud-Edge Module Components:

- **Cloud-Edge Manager:** A customized OpenNebula 7.1.80 that includes COGNIT-specific capabilities: Biscuit authentication driver, Scaphandre energy monitoring integration, Prometheus exporters for VM metrics, geolocation metadata injection, and OneForm for multi-provider Edge Cluster provisioning. The Cloud-Edge Manager orchestrates Serverless Runtime lifecycle management using OneFlow services.

- **AI-Enabled Orchestrator:** An intelligent orchestration component that uses AI/ML models for workload prediction and multi-objective optimization algorithms for resource placement. The orchestrator asynchronously updates device-to-cluster assignments in the COGNIT Frontend database, enabling proactive optimization while maintaining low-latency device responses.

Execution Environment:

- **Serverless Runtime:** Specialized virtual machine instances that execute offloaded functions. Serverless Runtimes are deployed as openSUSE-based appliances built with KIWI, containing pre-installed Serverless Runtime software, Python runtime, RabbitMQ consumers, and Prometheus exporters. Multiple flavours exist for different use cases (SmartCity, WildFire, Energy, Cybersecurity), each with use-case-specific libraries and tools.

Deployment Automation:

- **COGNIT OpsForge:** Deployment automation tool that provisions infrastructure and installs and configures the control plane components (COGNIT Frontend, Cloud-Edge Manager, AI-Enabled Orchestrator). OpsForge supports deployment to AWS EC2 and on-premises infrastructure, with integrated KIWI support for building Serverless Runtime images that are automatically populated into the Cloud-Edge Manager during the content population phase.

The final software stack implements several key architectural innovations:

- **Database-Mediated Assignment Caching**: The COGNIT Frontend uses an SQLite database with composite primary keys (device_id, flavour) to cache. device-to-cluster assignments, enabling fast response times while the AI-Enabled Orchestrator asynchronously optimizes assignments in the background.

- **Queue-Based Function Dispatching:** Edge Cluster Frontends use RabbitMQ message queues for internal request distribution, providing natural load balancing, fault tolerance, and scalability while maintaining synchronous APIs for devices.

- **Multi-Provider Edge Cluster Provisioning:** OneForm integration enables automated provisioning of Edge Clusters across multiple cloud providers (AWS, Azure, Google Cloud, Equinix) and on-premises infrastructure, with support for cross-site live migration through VXLAN/EVPN overlays.

- **Intelligent Scaling:** Edge Cluster Frontends expose scaling endpoints that enable the AI-Enabled Orchestrator to dynamically adjust Serverless Runtime cardinality based on queue depth metrics and workload predictions, with graceful drain algorithms that prevent interrupting in-flight executions.

- **Comprehensive Monitoring:** Integration of Scaphandre for energy consumption metrics, Prometheus for infrastructure and application metrics, and geolocation metadata enables real-time operational decisions and ML model training for proactive optimization.

- **Biscuit Token Authentication:** Cryptographic token-based authentication using Biscuit tokens provides fine-grained, capability-based access control across the

entire COGNIT Framework, enabling secure multi-tenant operations without password management overhead.

Deployment and Integration

OpsForge automates the deployment of the control plane components:

- **Control Plane Components:** OpenNebula Cloud-Edge Manager, COGNIT Frontend, and AI-Enabled Orchestrator are automatically deployed and configured on the control plane subnet.
- **COGNIT Features:** The customized Cloud-Edge Manager includes all COGNIT-specific features (Biscuit auth driver, Scaphandre integration) that are built into the customized OpenNebula 7.1.80 distribution.
- **Default Resources:** When OpenNebula Cloud-Edge Manager is installed, default datastores (image and system datastores) are automatically created as part of the standard installation process. Virtual networks are defined on Edge Clusters when they are provisioned, not in the control plane.
- **Edge Cluster Components:** Edge Cluster Frontend and Serverless Runtime are packaged together in OneFlow Service Templates that are pre-populated in the Cloud-Edge Manager during OpsForge deployment. Different service templates exist for each use case (SmartCity, WildFire, Energy, Cybersecurity). Edge Clusters can be provisioned by administrators using OpsForge.

8. COGNIT Testbed Infrastructure

8.1 Summary

The COGNIT testbed infrastructure has undergone a significant transformation from July 2023 (M7) to September 2025 (M34), evolving from a basic proof-of-concept deployment to a production-ready, distributed edge-cloud platform. The testbed now supports the complete COGNIT v2 architecture with centralized management, distributed edge clusters across Europe, and comprehensive data sharing capabilities.

- Timeline: July 2023 - September 2025 (27 months)
- Primary Location: RISE ICE Datacenter, Luleå, Sweden
- Scale: 600+ vCores, 1.7TB RAM, distributed across 7+ clusters in multiple European countries

8.2 Infrastructure Evolution Timeline

Phase 1: Foundation and Initial Deployment (Q3 2023 - Q1 2024, M7 - M18)

- July 2023 - Initial Project Setup
    - Project initiated with basic repository and documentation framework
    - Initial commit establishing project structure
- September 2023 - Core Infrastructure Deployment
    - Deployment migration to new infrastructure platform
    - Implementation of basic OpenNebula frontend
    - Setup of core service endpoints
        - Sunstone UI for VM management
        - API endpoints for programmatic access
        - Grafana monitoring integration
        - OneFlow API for service orchestration
- October-November 2023 - Service Expansion
    - Addition of Provisioning Engine (COGNIT v1 architecture component)
    - Integration of first compute nodes at ICE cluster
        - 2 GPU-enabled nodes (48 vCores, 512GB RAM)
    - Implementation of SSL termination via Nginx Proxy Manager
- December 2023 - Q1 2024 - Network Foundation
    - Development of WireGuard VPN management network
    - IPv6 gateway implementation for developer access
    - Internal network architecture establishment (10.10.10.0/24, fd67::/64)

Phase 2: Architecture Transition and Scaling (Q1 2024 - Q4 2024, M18 - M24)

- Q1 2024 - Network and Host Expansion
    - Addition of scheduler-extras VM for enhanced processing
    - Implementation of internal IPv6 network (fd67::/64)
    - Documentation restructuring for better organization

    - Individual host documentation creation for p02r11srv01 and p02r11srv15
- Q2-Q4 2024 - Edge Cluster Integration
    - Integration of multiple edge clusters:
        - DYN cluster (4 nodes, 64 total vCores)
        - Barcelona UC1 edge cluster (64 vCores, 128GB RAM)
        - CETIC edge cluster (2 nodes, 16 vCores each)
    - Addition of ICE internal IPv4 network (192.168.120.0/22)
    - Prometheus monitoring implementation across hosts

Phase 3: COGNIT v2 Implementation (Q1 2025 - Q3 2025, M25 - M34)

- January 2025 - Data Layer Implementation
    - Major milestone: MinIO S3 storage deployment
    - Public DaaS implementation (single-node, scalable architecture)
    - Edge private DaaS instances at each cluster
    - Architecture v2 transition: Deprecation of Provisioning Engine
- February-April 2025 - Architecture v2 Completion
    - COGNIT Frontend deployment (replacing Provisioning Engine)
    - Edge Cluster Frontend implementation at ICE
    - Phoenix cluster integration (3 nodes, 240 total vCores, 768GB RAM)
    - Francesco cluster addition (64 vCores, 128GB RAM)
- September-October 2025 - Upgrade and finalization
    - Final documentation updates and system optimization
    - OpenNebula upgrade from 6.8 to 7.0 (frontend + hypervisors)
    - Integration of additional COGNIT-specific features compatible with OpenNebula 7.0

8.3 Available Resources

By the end of the project, this were the compute resources usable for the partners:

- Total Capacity: ~600 vCores, ~1.7TB RAM
- GPU Resources: 2 GPU-enabled nodes (Nvidia GTX 2080ti, GTX 1080ti)
- Geographic Distribution: 7 clusters across multiple European sites
    - ICE Cluster: 2 GPU-enabled nodes (48 vCores, 512GB RAM)
    - DYN Cluster: 4 nodes (64 vCores, 48GB RAM)
    - Barcelona UC1: 1 node (64 vCores, 128GB RAM)
    - CETIC: 2 nodes (16 vCores, 32GB RAM)
    - Phoenix: 3 nodes (240 vCores, 768GB RAM)
    - Francesco: 1 node (64 vCores, 128GB RAM)

For the duration of the project, internal github repositories were used to share the full testbed deployment details, usage instructions and documentation.

9. Software Requirements Verification

Possible status are: NOT STARTED | COMPLETED

9.1 Device Client

SR1.1 Interface with COGNIT Frontend
Status: COMPLETED
Description: Implementation of the communication of the Device Client with the COGNIT Frontend. Following the instructions[60] of the Project's GitHub repository, a user can authenticate, update application requirements and get an Edge Cluster Frontend IP address to offload the execution of a Python function. All the library functionalities can be tested standalone by executing the unit tests provided on the GitHub repository.[61]
Completed Verification Scenarios: • [VS1.1.1] The Device Client is able to get authorisation from COGNIT and is able to send App valid requirements to the COGNIT Frontend. • [VS1.1.2] The Device Client is able to receive a valid Edge Cluster (effectively a valid Edge Cluster Frontend IP address). • [VS1.1.3] The Device Client is able to update the App requirements at any moment. • [VS1.1.4] The Device Client is able to receive a changed Edge Cluster seamlessly from a COGNIT's proactive decision-making action. • [VS1.1.5] The Device Client is able to handle (upload/read) data on the COGNIT global layer.

SR1.2 Interface with Edge Cluster
Status: COMPLETED
Description: Implementation of the communication of the Device Client with the Edge Cluster.
Completed Verification Scenarios: • [VS1.2.1] The Device Client is able to execute functions (either preloaded or

[60] https://github.com/SovereignEdgeEU-COGNIT/device-runtime-py/blob/main/README.md
[61] https://github.com/SovereignEdgeEU-COGNIT/device-runtime-py/tree/main/cognit/test

uploading it at the moment of execution) on the assigned Edge Cluster.

- [VS1.2.2] The Device Client is able to handle (upload/read) data privately in the assigned Edge Cluster.

SR1.3 Programming languages

Status: COMPLETED

Description: Support for different programming languages.

Completed Verification Scenarios:

- VS[1.3.2] Test VS1.1.1 to VS1.1.4 and VS1.2.1 validation scenarios both in C and Python versions of the Device Client.
- VS[1.3.1] Test VS1.1.5 and VS1.2.2 validation scenarios both in C and Python version of the Device Client.

SR1.4 Low memory footprint for constrained devices

Status: COMPLETED

Description: Low memory footprint for constrained devices.

Completed Verification Scenarios:

- VS[1.3.2] Test VS1.1.1 to VS1.1.4 and VS1.2.1 validation scenarios in the C version of the Device Client.
- [VS1.4.1] Test VS1.1.5 and VS1.2.2 validation scenarios on a device with less than 500kB of RAM, making use of the Device Client in C.

SR1.5 Security

Status: COMPLETED

Completed Verification Scenarios:

- [VS1.5.1] The Device Client is able to perform secure communications against the COGNIT Frontend and the assigned Edge Cluster Frontend with the

acceptance of the authorization mechanism.

- [VS1.5.2] The Device Client is not permitted any unauthorised action towards the COGNIT Frontend or the assigned Edge Cluster.

SR1.6 Collecting Latency Measurements
Status: COMPLETED
Completed Verification Scenarios: • [VS1.6.1] The Device Client is able to measure latencies to different Edge Clusters concurrently with other activities of the Client..

9.2 COGNIT Frontend

SR2.1 COGNIT Frontend
Status: COMPLETED
Description: Provides an entry point for devices to communicate with the COGNIT Framework for offloading the execution of functions and uploading global data.
Completed Verification Scenarios: • [VS2.1.1] Authenticate a Device against the COGNIT Frontend and verify that an authorization token is returned. • [VS2.1.2] Upload application requirements to the COGNIT Frontend and verify that a unique ID is returned for the application requirements. • [VS2.1.3] Upload application requirements and query the COGNIT Frontend for an Edge Cluster and verify that it meets the application requirements. • [VS2.1.4] Upload a function to the COGNIT Frontend and verify that a unique ID is returned for that function. • [VS2.1.5] Test uploading and downloading data by the device to and from the COGNIT Frontend.

9.3 Edge Cluster

SR3.1 Edge Cluster Frontend

Status: COMPLETED

Description: The Edge Cluster must provide an interface (Edge Cluster Frontend) for the Device Client to offload the execution of functions and to upload local data that is needed to execute the function.

Completed Verification Scenarios:

- [VS3.1.1] Instantiate a Serverless Runtime and verify that a device can request the execution of a function to the Edge Cluster Frontend and assert the result of the function.
- [VS3.1.2] Test uploading and downloading data by the device to and from the Edge Cluster using a secure communication channel.

SR3.2 Secure and Trusted Serverless Runtimes

Status: COMPLETED

Description: The Serverless Runtime is the minimal execution unit for the execution of functions offloaded by Device Clients.

Completed Verification Scenarios:

- [VS3.2.1] Build a Serverless Runtime image, customised for each Use Case, in an automated way.

9.4 Cloud-Edge Manager

SR4.1 Provider Catalog

Status: PARTIALLY COMPLETED

Description: Implement a backend to persist information about the available providers that can be used to extend the capacity of the COGNIT infrastructure.

Completed Verification Scenarios:

- [VS4.1.1] Listing the providers belonging to the Provider Catalog.

Pending Verification Scenarios:

- [VS4.1.2] Filtering the providers according to a desired latency threshold on a geographic area.
- [VS4.1.3] Filtering the providers according to energy consumption per hour threshold.
- [VS4.1.4] Filtering the providers according to some specific hardware characteristics (e.g. Trusted Execution Environments).

SR4.2 Edge Cluster Provisioning

Status: COMPLETED

Description: The Cloud-Edge Manager must be able to provision Edge Clusters as a set of software-defined compute, network, storage on any cloud/edge location available in the Provider Catalogue.

Completed Verification Scenarios:

- [VS4.2.1] A YAML file containing the information about the provision is provided to the Cloud-Edge Manager that creates a new Edge Cluster.
- [VS4.2.2] Query the Cloud-Edge Manager to return the status of an Edge Cluster identified by its ID.
- [VS4.2.3] Query the Cloud-Edge Manager to scale up/down the number of hosts of an Edge Cluster identified by its ID.
- [VS4.2.4] Query the Cloud-Edge Manager to delete an Edge Cluster identified by its ID.

SR4.3 Serverless Runtime Deployment

Status: COMPLETED

Description: The Cloud-Edge Manager must be able to deploy Serverless Runtimes as Virtualized Workloads within an Edge Cluster.

Completed Verification Scenarios:

- [VS4.3.1] A YAML file containing the information about the deployment is provided to the Cloud-Edge Manager that creates a new Serverless Runtime.

- [VS4.3.2] Query the Cloud-Edge Manager to return the status of a Serverless Runtime identified by its ID.
- [VS4.3.3] Query the Cloud-Edge Manager to scale up/down the resources (CPU, memory and disks) of a Serverless Runtime identified by its ID.
- [VS4.3.4] Query the Cloud-Edge Manager to update the deployment of the Serverless Runtime identified by its ID.
- [VS4.3.5] Query the Cloud-Edge Manager to delete a Serverless Runtime identified by its ID.

SR4.4 Metrics, Monitoring, Auditing

Status: COMPLETED

Description: Edge-Clusters monitoring, Serverless Runtimes metrics collection and continuous security assessment.

Completed Verification Scenarios:

- [VS4.4.1] Create an Edge Cluster and deploy a Serverless Runtime and check the metrics collected for a certain period of time.

SR4.5 Authentication & Authorization

Status: COMPLETED

Description: Authentication and authorization mechanisms for accessing cloud-edge infrastructure resources by the devices for offloading workloads.

Completed Verification Scenarios:

- [VS4.5.1] Test the creation of new users and groups
- [VS4.5.3] Communicate with the COGNIT Frontend and the Edge Cluster Frontend using tokens.
- [VS4.5.2] Assign ACLs to designated users and test the creation of new Edge Clusters and Serverless Runtimes.

SR4.6 Plan Executor

Status: COMPLETED

Description: The Plan Executor is responsible for converting plans provided by the AI-Enabled Orchestrator in Cloud-Edge Manager actions for the life cycle management of Edge Clusters and Serverless Runtimes.

Completed Verification Scenarios:

- [VS4.6.1] Submit a plan to the Plan Executor for creating new Serverless Runtimes and verify the deployment of the Serverless Runtimes.
- [VS4.6.2] Submit a plan to the Plan Executor for migrating a Serverless Runtime.

9.5 AI-Enabled Orchestrator

SR5.1 Building Learning Models

Status: COMPLETED

Description: Provide AI/ML models trained with input from collected metrics from the Cloud-Edge Manager monitoring service related to Edge Clusters and Serverless Runtimes deployed across the distributed cloud-edge continuum.

Completed Verification Scenarios:

- [VS5.1.1] List instances from Devices to Applications to System for metrics to be collected.
- [VS5.1.2] Correlate and represent features that are ready to take as input to the Model.
- [V1S5.1.3] Feedback-aware performance check when training the model on represented features.
- [VS5.1.4] Assess the ability in terms of AUROC score for each task (e.g. scheduling).

SR5.2 Smart Management of Cloud-Edge Resources

Status: COMPLETED

Description: The AI-Enabled Orchestrator is responsible for the automated

management of cloud-edge continuum resources in order to optimize the performance of the applications that are offloading functions to the COGNIT Framework.

Completed Verification Scenarios:

- [VS5.2.1] Assess the ability of workload and resource optimization in terms of energy and performance trade-off.

9.6 Secure and Trusted Execution of Computing Environments

SR6.1 Advanced Access Control

Status: COMPLETED

Description: Implement policy-based access control to support security policies on geographic zones that define a security level for specific areas.

Pending Verification Scenarios:

- [VS6.1.1] Define a security policy that is based on geographic zone attributes.
- [VS6.1.2] Check enforcement of new security policy when edge device moves closer from one edge node than another.

SR6.2 Confidential Computing

Status: COMPLETED

Description: Enable privacy protection for the application workloads at the hardware level using Confidential Computing (CC) techniques.

Completed Verification Scenarios:

- [VS6.2.1] Deploy a function on a host that provides confidential computing capability.
- [VS 6.2.2] Check that the function is executed inside the host trusted execution environment (TEE).

SR6.3 Federated Learning

Status: NOT STARTED

Description: Enhance privacy of AI workloads that have confidentiality requirements preventing the exchange of information for training. Federated Learning techniques enable confidential or private data processing under the control of the data owner or controller, with only learned models shared.

Pending Verification Scenarios:

- [VS6.3.1] Perform training of the ML algorithm without exchanging local data.
- [VS6.3.2] Check that the redistributed models for inference do not contain private data.

10. Use Case Requirements Verification

This section contains a brief follow-up on the use case requirements initially stated in the early month three D5.1 deliverable, to ensure that they have been handled during the project.

Id	Description	Source	Follow-up
UR0.1	Device applications should be able to offload any function written in C or Python languages.	All	This has been successfully implemented and tested by the relevant UCs
UR0.2	Device applications should be able to upload data from the device ensuring data locality with respect to where the offloaded function is executed.	All	This requirement is satisfied by the DaaS component, and tested by the relevant UCs using a MinIO-based implementation.
UR0.3	Device applications should be able to upload data from external backend storages ensuring data locality with respect to where the offloaded function is executed.	All	The DaaS component allows uploading from external services where needed.
UR0.4	Execution of functions such as ML inference engines should be able to load machine learning models stored ensuring data locality with respect to where the function is executed.	All	This requirement is satisfied by the DaaS component, and tested by the relevant UCs using a MinIO-based implementation.
UR0.5	Function execution can be executed in different tiers of the Cloud-Edge continuum according to network latency requirements.	All	Network latency requirements can be specified by the UCs, and are handled transparently by the COGNIT backend.
UR0.6	Device application shall have the ability to define maximum execution time of the	All	Maximum execution time requirements can be specified by the UCs, and are handled

	offloaded function upon offloading.		transparently by the COGNIT backend.
UR0.7	Device application shall have the ability to specify and enforce runtime maximum provisioning time and runtime shall be provisioned within the previously specified time.	All	Maximum response time requirements can be specified by the UCs, and are handled transparently by the COGNIT backend.
UR0.8	Device applications must be able to request and obtain an authorization prior to establishing any further interaction with COGNIT.	All	This is handled through the COGNIT biscuit authorisation system.
UR0.9	IAM system integration for high granularity authentication and user management for device clients and the COGNIT Framework.	All	Cloud-Edge Manager IAM system provides authentication. Fine-grained access control implemented.
UR0.10	Push mechanism to inform about status/events from the COGNIT Framework back to the requestor device client.	All	Push mechanism not implemented, as it was not needed. The Device client uses only synchronous request-response patterns to obtain execution results.

Table 10.1. Common user requirements.

Id	Description	Source	Follow-up
UR 1.1	Function execution shall be supported in shared, multi-provider environments (with different access and authorization procedures), and the execution must be isolated from other processes on the host system.	UC1	Multi-provider isolation via VM-based Serverless Runtimes. Strong isolation between processes. Multi-provider support through Edge Cluster abstraction.
UR 1.2	Device application shall have the ability to dynamically scale resources for offloading function execution to maximise exploitation of resources in shared environments, while avoid saturation or resources kidnapping.	UC1	Elastic scaling of Serverless Runtimes. Horizontal scaling based on workload. Resource utilization optimization. Avoids saturation through proactive scaling.

UR 1.3	Function execution should exploit data locality and prioritise edge nodes where the required data is already stored.	UC1	Achieved through the Local DaaS deployed in each Edge Cluster
UR 1.4	The whole life cycle of either function execution or code offloading should be auditable and non repudiable.	UC1	Function executions are comprehensively monitored and logged throughout their lifecycle.
UR 1.5	Device applications should be able to request execution over GPUs.	UC1	GPU support was not implemented as the Smart Cities use case did not require GPU acceleration. CPU-based execution was sufficient.

Table 10.2. User requirements for UC1 (Smart Cities).

Id	Description	Source	Follow-up
UR 2.1	It shall be possible to obtain both a-priori estimates of expected, and actual measurements of, energy consumption of the execution of function.	UC2	Energy metrics available through the monitoring system. Actual energy measurements via Scaphandre.
UR 2.2	COGNIT Framework should be able to adapt to rare events with sudden peaks of FaaS requests, in which the offloaded function requires much heavier computations and more frequent execution than usual.	UC2	AI-Enabled Orchestrator adapts to sudden peaks. Automatic scaling of Serverless Runtimes. Workload prediction enables proactive scaling.
UR 2.3	Possibility for devices to request access to GPUs, when available, during high-alert mode.	UC2	GPU support was not implemented as the Wildfire Detection use case did not require GPU acceleration. CPU-based execution with elastic scaling was sufficient for handling high-demand modes.

Table 10.3. User requirements for UC2 (Wildfire Detection).

Id	Description	Source	Follow-up

UR 3.1	Device Client and user applications shall share a maximum of 500 kB of available RAM in total.	UC3	C Device Client meets memory footprint constraint.
UR 3.2	It shall be possible for the user application to dynamically scale up/down resources for function execution due to changes in the user preferences.	UC3	Elastic scaling for changing demand. Dynamic resource allocation. Serverless Runtime scaling based on application demand.
UR 3.3	The SDK for the Device Client shall have support for the C programming language.	UC3	C language support provided and tested

Table 10.4. User requirements for UC3 (Energy).

Id	Description	Source	Follow-up
UR 4.1	The Device Client should have the ability to dynamically set the permissible edge nodes for executing the function based on policy (e.g. geographic security zones, distance to edge node).	UC4	The relevant policy permissions for edge nodes have been successfully tested
UR 4.2	The COGNIT Framework should have the ability to live migrate of data/runtime to different edge locations based on policy and location of function execution (e.g. geographic security zones, distance to edge node).	UC4	Data availability via Global DaaS, accessible from all Edge Clusters, providing transparent data availability across clusters. Dynamic device reassignment through Multi-Cluster Optimizer enables devices to be reassigned to different Edge Clusters.
UR 4.3	The Device should be able to request the execution of a function as close as possible (in terms of latency) to the Device's location.	UC4	Device location considered in Edge Cluster selection. Function execution as close as possible to device location. Latency estimation informs placement decisions.

Table 10.5. User requirements for UC4 (Cybersecurity).

11. Conclusions

Through this final WP5 deliverable, the project has documented how it has advanced the vision of a cloud-edge continuum and validated its benefits through its four diverse use cases: Smart Cities, Wildfire Detection, Energy, and Cybersecurity. These scenarios have demonstrated that combining cloud scalability with edge proximity can enable low-latency, secure, and context-aware services, while maintaining many of the benefits, such as flexibility and computational power, of centralised infrastructures.

Throughout the project the use cases have transitioned from initial requirement, design and iterated development into preliminary testing and finally during the last reporting period to large-scale validation on integrated infrastructures, demonstrating interoperability between local heterogeneous edge and cloud environments. The COGNIT software stack has enabled streamlined deployment and orchestration across distributed resources. Furthermore the use case partners have outlined the continued development and testing on a wider set of IoT and other hardware platforms needed to further increase the technology readiness of the developed solutions.

Key achievements include:

- End-to-end validation of use case applications on real hardware and shared testbeds.
- Improved technology readiness for both embedded applications and critical components, paving the way for scalable and trustworthy deployments across several application areas.
- Deployment of advanced orchestration mechanisms, including AI-driven resource management and secure execution environments.
- Integration and verification of software requirements across diverse infrastructures, ensuring the initially identified demands have been met, while meeting user requirements.

In conclusion, the project has confirmed that the cloud-edge continuum is technically feasible and strategically relevant for next-generation digital services. By bridging centralized and distributed computing paradigms, this approach can deliver greater responsiveness, increased resilience, and improved sustainability. The validated use cases and integrated software stack set the stage for continued evolution toward an open, easy to integrate, intelligent, and secure computing ecosystem.