A Cognitive Serverless Framework for the Cloud-Edge Continuum

D5.4 Use Cases - Scientific Report - d

Version 1.0

12 November 2024

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 document provides an overall status of 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 provides an update on the Project's software integration process and infrastructure, on its testbed environment, and on the progress of the software requirement verification tasks during the Third Research & Innovation Cycle (M16-M21).

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:	1.0
Contractual Date of Delivery:	30/09/2024
Actual Date of Delivery:	12/11/2024
Lead Author:	Thomas Ohlson Timoudas (RISE)
Authors:	Monowar Bhuyan (UMU), Marek Białowąs (Phoenix), Dominik Bocheński (Atende), Malik Bouhou (CETIC), Aritz Brosa (Ikerlan), Idoia de la Iglesia (Ikerlan), Agnieszka Frąc (Atende), Grzegorz Gil (Atende), Torsten Hallmann (SUSE), Joel Höglund (RISE), Carlos Lopez (ACISA), Mateusz Kobak (Phoenix), Tomasz Korniluk (Phoenix), Johan Kristiansson (RISE), Antonio Lalaguna (ACISA), Fátima Fernández (Ikerlan), Carlos Lopez (ACISA), Marco Mancini (OpenNebula), Alberto P. Martí (OpenNebula), Philippe Massonet (CETIC), Nikolaos Matskanis (CETIC), Daniel Olsson (RISE), Mikel Irazola (Ikerlan), Álvaro Puente (Ikerlan), Holger Pfister (SUSE), Tomasz Piasecki (Atende), Francesco Renzi (Nature4.0), Bruno Rodríguez (OpenNebula), Juan José Ruiz (ACISA), Kaja Swat (Phoenix), Paul Townend (UMU), Iván Valdés (Ikerlan), Riccardo Valentini (Nature4.0), Constantino Vázquez (OpenNebula), Pavel Czerny (OpenNebula).
Status:	Submitted

Document History

Version	Issue Date	Status[1]	Content and changes
0.1	28/10/2024	Draft	Initial Draft
0.2	06/11/2024	Peer-Reviewed	Reviewed Draft
1.0	12/11/2024	Submitted	Final Version

Peer Review History

Version	Peer Review Date	Reviewed By
0.1	31/10/2024	Torsten Hallmann (SUSE)
0.1	11/11/2024	Antonio Álvarez (OpenNebula)

Summary of Changes from Previous Versions

First Version of Deliverable D5.4

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

Executive Summary

Deliverable D5.4, released at the end of the Third Research & Innovation Cycle (M21), is the fourth version of the Use Cases Scientific Report in WP5 ”Adaptive Serverless Framework Integration and Validation”. It offers a summary of the work done in this cycle and for the demonstration of the second version of the COGNIT Framework and its integration with the Use Cases to be demonstrated in the testbed environment for the project. Also, it provides information on the progress of the software requirements verification tasks per component

In connection with the main components of the COGNIT Architecture (i.e. Device Client,, COGNIT Frontend, Edge Cluster, Cloud-Edge Manager, and AI-Enabled Orchestrator), the Project has delivered progress across different Software Requirements linked to a set of functionalities for the Use Cases that enables their own research and development activities and helps in the integration of their devices with the COGNIT Framework.

Apart from this document (Deliverable D5.4) and the Project's global overview provided through Deliverable D2.4, specific research and development activities performed in WP3 “Distributed FaaS Model for Edge Application Development” (related to the Device Client, the COGNIT Frontend, the Edge Cluster, and the Secure and Trusted Execution of Computing Environments) are described in detail in reports D3.3 “COGNIT FaaS Model - Scientific Report - c” and D3.8 “COGNIT FaaS Model - Software Source - c”, whereas those performed in WP4 “AI-Enabled Distributed Serverless Platform and Workload Orchestration” (related to the Cloud-Edge Manager, the AI-Enabled Orchestrator, and the Energy Efficiency Optimization in the Multi-Provider Cloud-Edge Continuum) are described in reports D4.3 “COGNIT Serverless Platform - Scientific Report - c” and D4.8 “COGNIT Serverless Platform - Software Source - c”.

The information in this report will be updated with incremental releases at the end of each remaining research and innovation cycle (i.e. M27 and M33).

Table of Contents

Abbreviations and Acronyms

5G	Fifth Generation Mobile Network
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
CCAM	Cooperative, Connected and Automated Mobility
CC-CV	Current and Constant Voltage
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
EN	European Norms
ENS	Esquema Nacional de Seguridad (Spanish National Security Schema)
ESS	Energy Storage System
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
ID	Identifier
IP	Internet Protocol
IPv6	Internet Protocol version 6
ITS	Intelligent Transport System
kW	kiloWatt(s)
LTE	Long-Term Evolution
MAPEM	MAP (Topology) Extended Message
ML	Machine Learning
M-Hub	Mobility Hub (advanced TLC)

NB-IoT	NarrowBand - Internet of Things
OBU	On Board Unit
OCPP	Open Charge Point Protocol
OS	Operating System
PCI	Peripheral Component Interconnect
PV	Photovoltaic
QA	Quality Assurance
QoS	Quality of Service
RES	Renewable Energy Source
REST	Representational State Transfer
RL	Reinforcement Learning
RSU	Road Side Unit
RTOS	Real-Time Operating System
SAE	Society of Automotive Engineers
SEM	Smart Energy Meter
SPATEM	Signal Phase And Timing Extended Message
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
TEE	Trusted Execution Environment
TLC	Traffic Light Controller
TS	Technology Specifications
TSP	Traffic/Transit Signal Priority
TTC	TreeTalker Cyber
V2X	Vehicle to Everything communication technology
VM	Virtual Machine
VPN	Virtual Private Network

[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 is the fourth version of the Use Cases Scientific Report. The initial version of the Use Cases Scientific Report (Deliverable D5.1), released in M3, provided an initial collection of user requirements, a description of each of the Use Cases—including an initial architecture design and a plan for the demonstration and validation to take place in Tasks T5.3, T5.4, T5.5, T5.6—and an update of the COGNIT Testbed environment.

Both the second and the third versions of the report (Deliverables D5.2 and D5.3), released in M9 and M15, provided summaries of the overall status of the contribution of the Project's software requirements towards meeting the user requirements that guide the development of the COGNIT Framework at the end of the First and Second Research & Innovation Cycle (M4-M9 and M10-M15). They also provided additional information about the domains targeted by the Use Cases and the Partners involved in them, listing new user requirements identified during the cycle, and offering an update on the Project's software integration process and infrastructure, on its testbed environment, and on the progress of the software requirement verification tasks per component.

Since the previous version of the report (Deliverable D5.3), there have been changes to the COGNIT architecture, motivated by the requirements of the use cases. This second version of the COGNIT Framework is described in Deliverable D2.4. During this Research & Innovation Cycle (M16-M21), the project partners have worked together with the use cases to transition to this new architecture, which, from the perspectives of the use cases, have required some modifications in how the Device Client interacts with the COGNIT Framework.

D5.4 gives an incremental update on the progress. The document follows the same structure as in previous versions. D5.4 is composed of an introductory section and nine additional sections organised in two main blocks of content:

- **Part I** focuses on tracking the overall status (Section 2) and progress of the research and development work performed for each of the Use Cases, and their current status, with a dedicated section per Use Case (Sections 3 to 6).
- **Part II** focuses on the Project's software integration process and infrastructure (Section 7), on the evolution of the testbed environment (Section 8), and on the software requirements verification tasks carried out per component (Section 9).

The document ends with a conclusion section (Section 10).

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 started yet, for activities in progress, 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).																				

Common Requirements	UR0.1	Device applications should be able to offload any function written in C or Python languages.
	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.

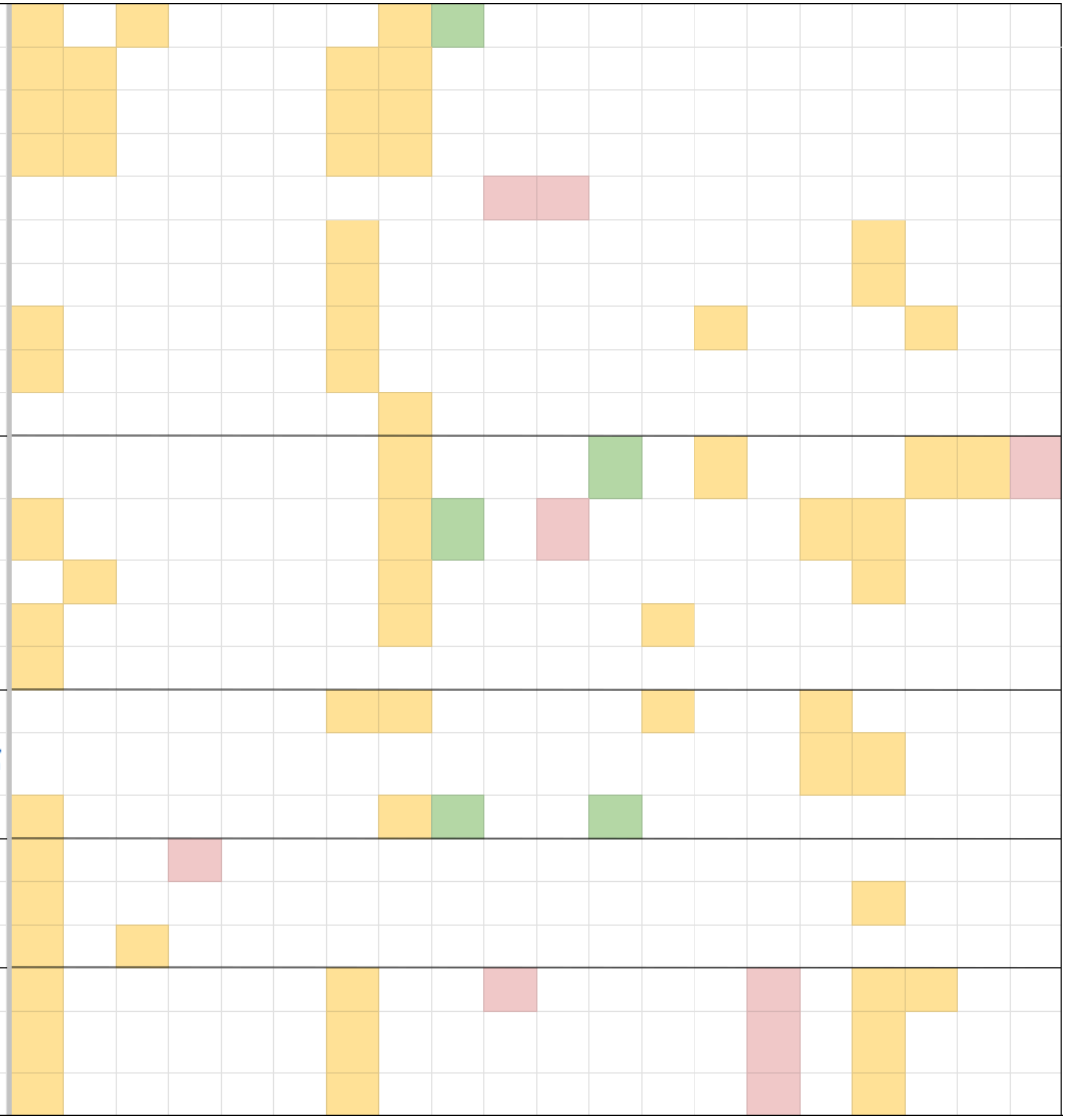

Table 2.1. Current status of each Software Requirement towards meeting its associated global/user requirements.

3. Use Case #1: Smart Cities

This use case addresses the use of the COGNIT Framework in critical infrastructures within the smart city. Mobility-Hub (henceforth M-Hub) is a new generation of traffic light controllers 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 services.

Considering that traffic infrastructures are owned by the city councils, enabling an open, standard-based far-edge platform will allow cities to deploy robust mobility and IoT services on top of existing traffic infrastructure. Therefore stakeholders such as Smart City, Mobility, Police, or emergency services departments inside city councils, could become beneficiaries and users of such infrastructure, without the need to deploy additional systems or equipment in the streets or vehicles. The objectives of the Smart City use case are:

- Implement a solution for a Traffic/Transit Signal Priority (TSP)[3] service for public transport and emergency vehicles relying on a continuum infrastructure.
- The new approach must optimise delays at intersections, and clear the junctions for emergency vehicles to reduce the probability of accidents, through direct interaction between the V2X On-Board Unit (OBU), the Road Side Unit (RSU) and the integration with the M-Hub.
- The serverless platform COGNIT enables at the edge, to design new digitalisation services optimising distributed on-prem resources.
- Seamlessly integrate far-edge clients (M-Hubs) with on-prem edge and cloud infrastructure, by means of the function-as-a-services (FaaS) paradigm.

3.1 Use Case Scenario and Architecture

The use case scenario and architecture are described in more detail in the previous deliverable “D5.3 Use Cases - Scientific Report, section 3.1: Current architecture and scenario”. A summary is provided below.

To demonstrate the capabilities of edge computing for improving the efficiency of public transportation and emergency services, this use case focuses on public bus and emergency vehicle prioritisation. A key element in this 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 Traffic Light Controller (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 decides if it should be approved or not, typically based on bus schedules. 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

[3] https://www.emtracsystems.com/wp-content/uploads/2021/01/TSPHandbook10-20-05.pdf

real-time, taking into account factors such as its position, speed, intersection layout, traffic status, etc.

The Smart City use case will explore the use of V2X technology for public bus and emergency vehicle 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.

3.2 Summary of developments during cycle M16-M21

During this cycle, we have mainly researched how to align the messages between the devices and the edge with standards, its content and the equipment configuration as described in Figure 3.4 and Section 3.3. Figure 3.7 in Section 3.3 illustrates how priority functionality is executed by serverless functions provisioned through the COGNIT Framework.

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:

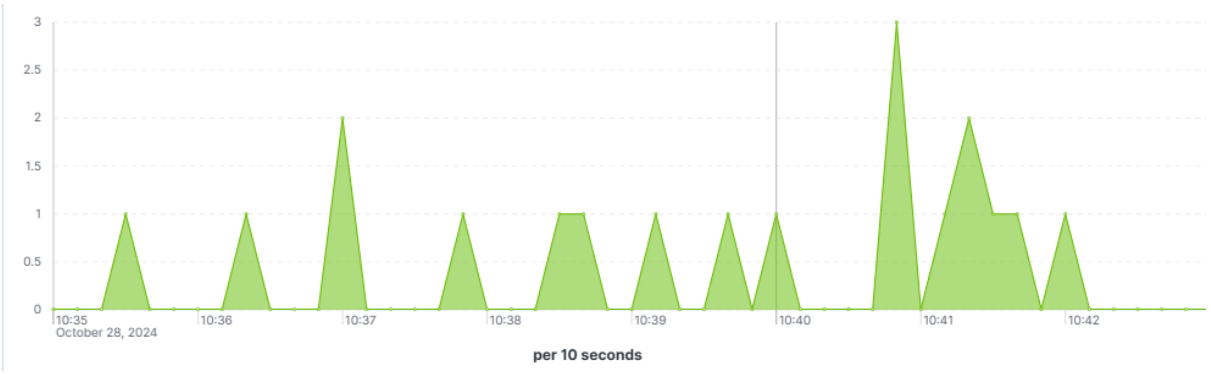

Figure 3.1. A snapshot of the priority request pattern obtained in the test bed on a typical day.

The data is accumulated 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 3.2 provides a few examples extracted from UNE/CEN

ISO 19091[4], but there will be additional unique scenarios. 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.

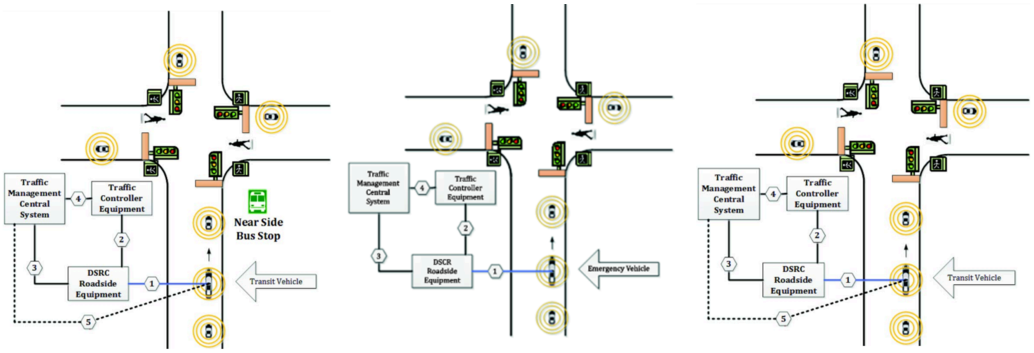

Figure 3.2. Examples of potential intersection scenarios in Transit Signal Priority (TSP). (Source UNE-CEN ISO/TS 19091)

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, but we haven't yet added its dynamic traffic model to be evaluated on a simulation engine. During this period we made a preliminary analysis of the intersections traffic flow in the test bed, and selected two of them to work on during the next phases, as shown in Figure 3.11 and Figure 3.12.

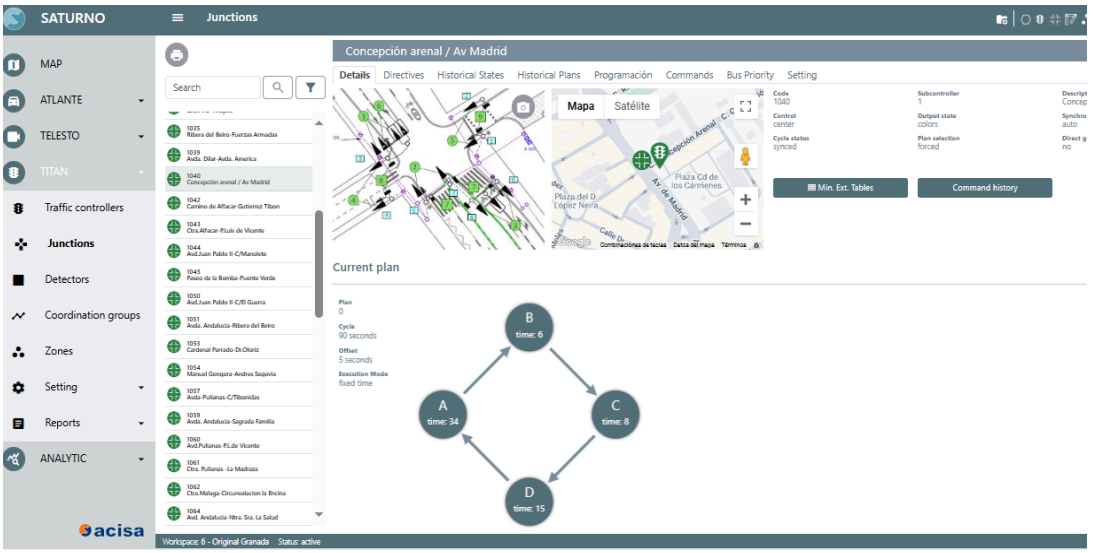

Figure 3.3. Example of the Digital Shadow of Granada's intersection in Saturno.

[4] https://www.iso.org/standard/69897.html

Integration of standards

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[5] 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.

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

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

The RSUs can send information about the traffic light groups and the topology of the intersection to the M-Hub system. This information will be used in the digital twin simulations that will 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 3.4 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.

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

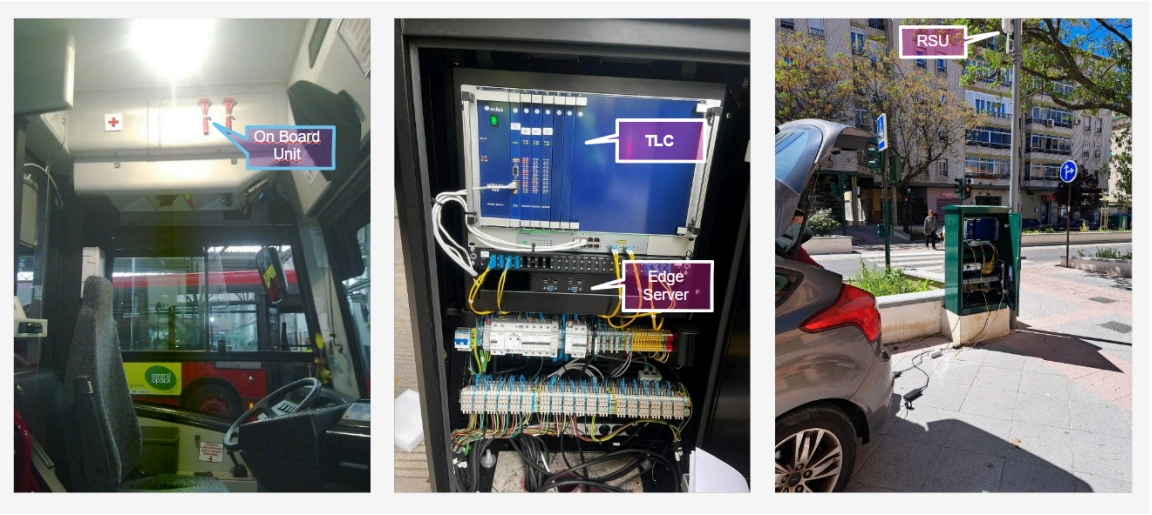

Figure 3.4. Installation of V2X On Board Units (left), M-Hub (center), and RSU (right) in the city of Granada.

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 Silver, as a client of the COGNIT Platform will make a remote FaaS request to M-Hub Gold installed on the far edge, 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 in 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” priority system enabled. In this project, the priority system will utilise the more advanced digital twin simulations enabled by COGNIT.

The graph shows that the average time crossing the bus is lower if the current priority system is enabled (green):

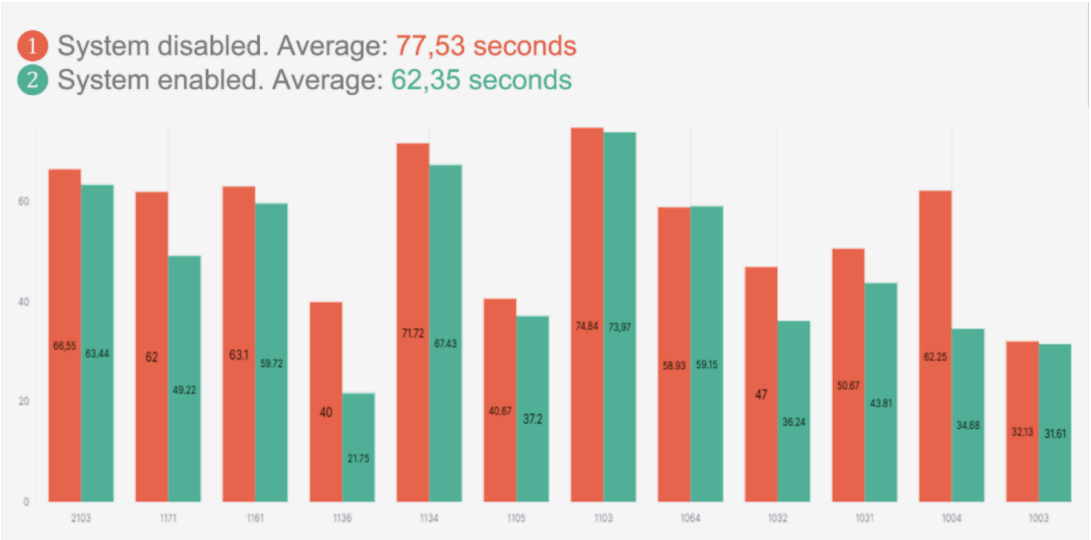

Figure 3.5. Comparison of average crossing times with and without priority system.

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 in achieving that, 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.

The following graph shows the average arrival time compared with the standard deviation of the different samples, per junction:

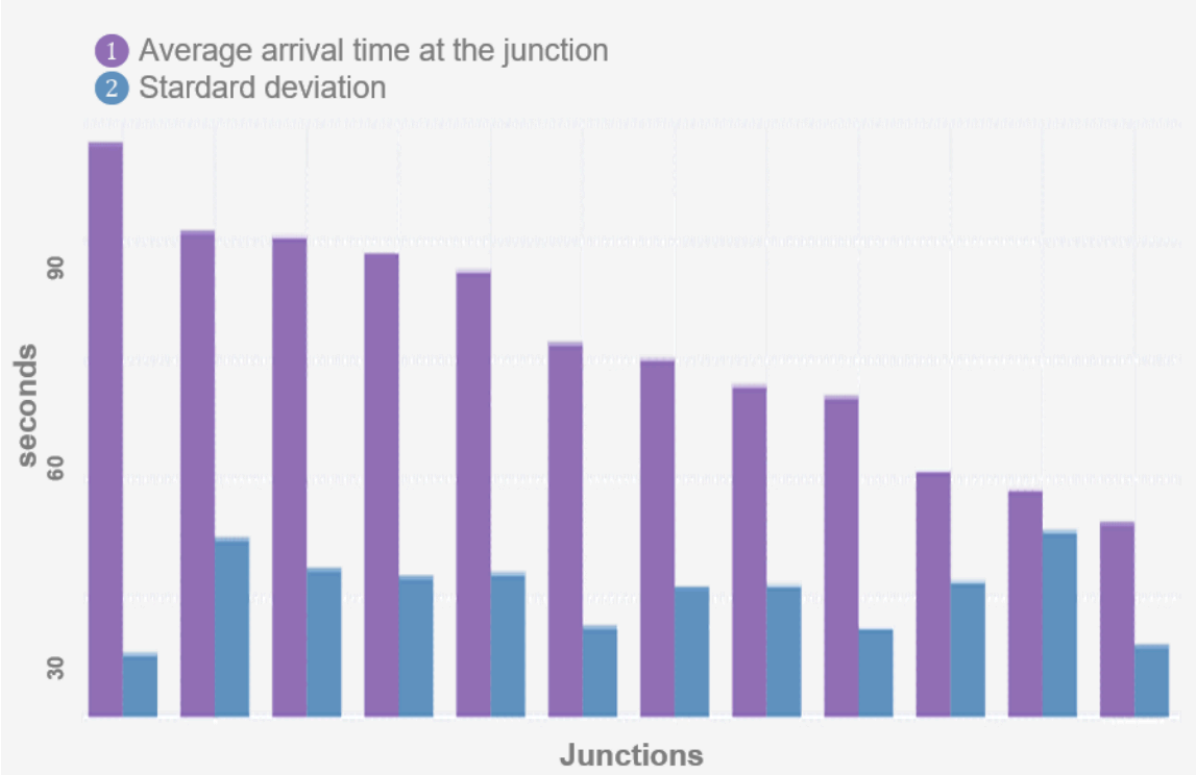

Figure 3.6. Average arrival time compared with the standard deviation

3.3 Integration with the COGNIT Framework

The following diagram in Figure 3.7 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.

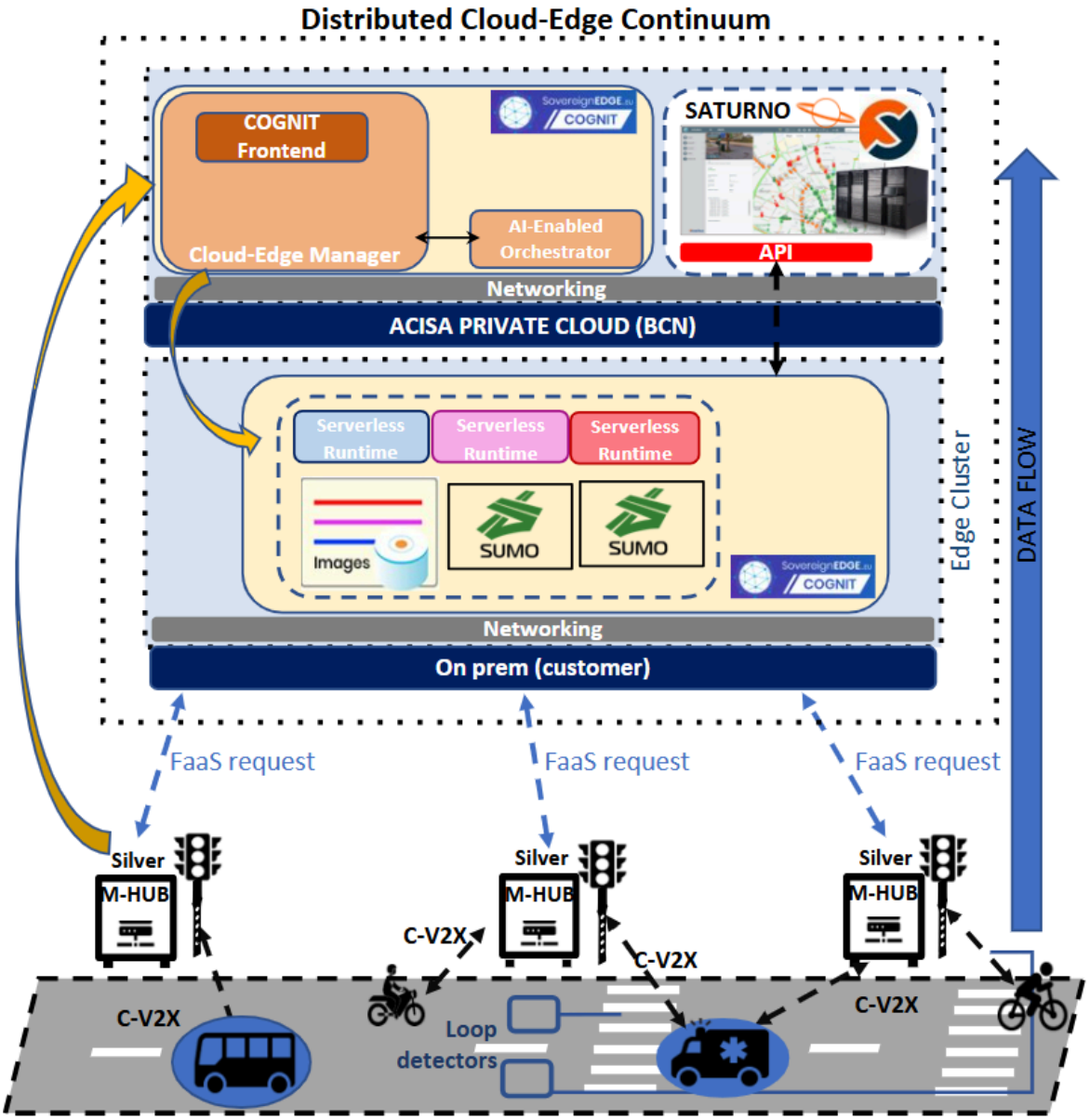

Figure 3.7. Overview of elements involved in the Smart City use case

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.[6] This is some heavyweight functionality that wouldn't make sense to deploy on the M-Hub Silver devices and will be leveraged using the remote FaaS to calculate some complex simulations and obtain KPIs to make decisions about priorities. It is a function to manage a priority and needs to be executed quickly. This requires setting up the Serverless Runtime image with the libraries and

[6] 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/

dependencies for the simulation, the models of the junctions (this is a temporary solution until the DaaS component is in place, expected in the forthcoming research and innovation cycle), 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 may be collected ideally by means of the DaaS service in the COGNIT Framework, although it currently resides as part of the intersection Digital Twin in Saturno. In future releases, the model might be a target for updates with recent traffic data stored in 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 limited time to make a decision. Our initial requirement for response time would be 1 second, although this will need to be confirmed through field testing.

With regard to response time for FaaS execution, another concern is to avoid as much as possible the “cold start” problem, since that would pose a risk to the response time our case needs. This is a clear requirement from this use case to avoid as far as possible any cold starts for our FaaS requests.

Being able to choose which infrastructure to run the serverless FaaS platform on could be a requirement for some cities. Smaller locations with small IT resources will probably prefer a deployment at managed service providers or public clouds. Bigger cities with more IT resources and knowledge and probably with more strict regulations, or compliance, would prefer or need a deployment on-premise. For the scope of this pilot, the preferred locations to deploy the edge nodes are Barcelona and Granada, Spain.

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 imply 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.

With regard to the data input for the SUMO simulations, currently it has been deployed in our Serverless Runtime image flavour, but we expect to manage it via the DaaS as soon as this service is available within the COGNIT Framework.

The hypothesis right now is that the application will require an average response time of less than 1 second, i.e., the time between the M-Hub performing a FaaS request, and an answer is received (does not include simulation time).

A typical medium city may have 200 intersections on average, we may consider a maximum rate of 10 priority requests per hour per intersection (see Figure 3.1 as reference), that is an upper limit of 2000 requests per hour in an average medium city. We may consider 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.

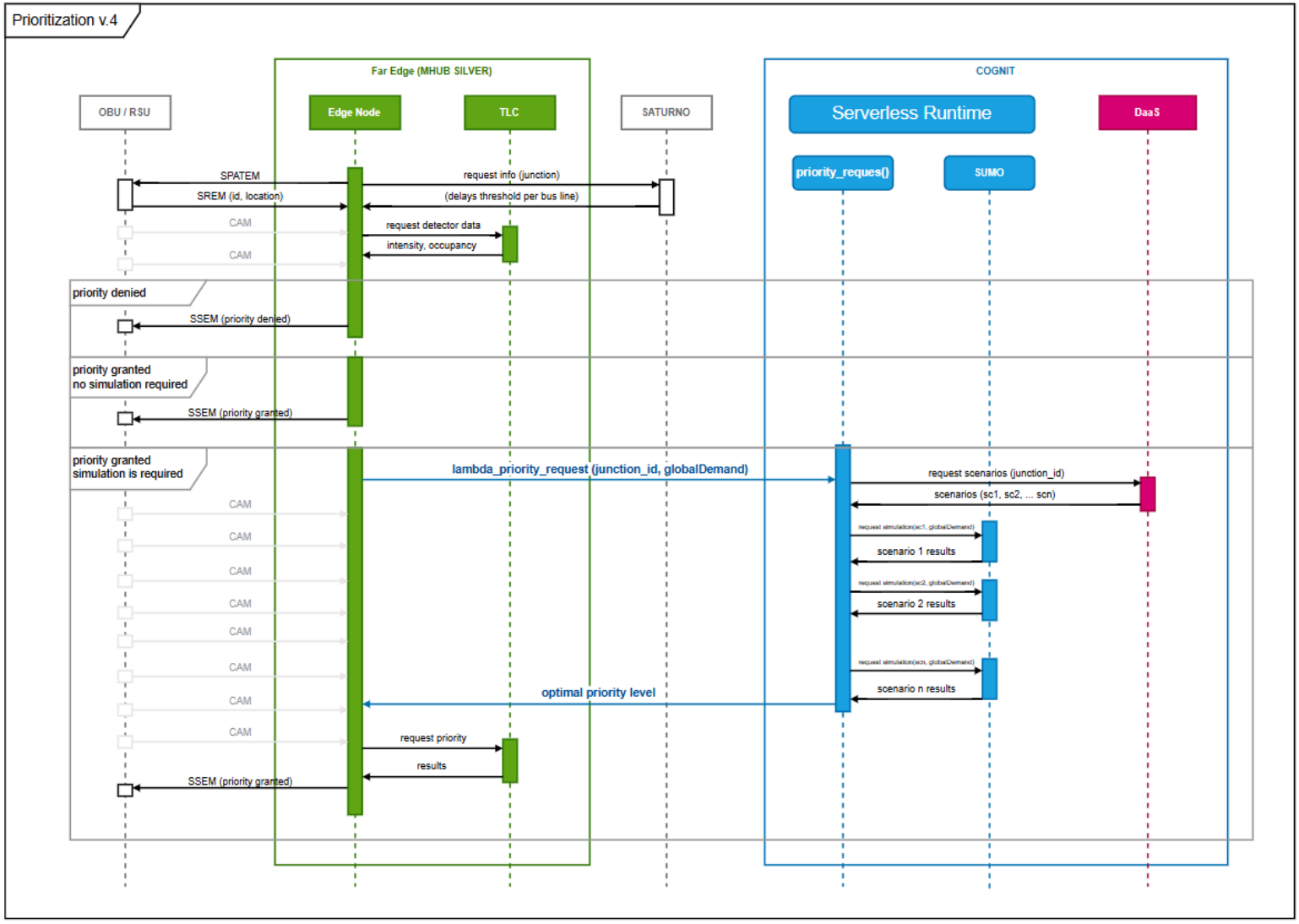

Figure 3.8. Overview of the prioritisation process, in which the M-Hubs makes FaaS requests to the Edge Cluster Frontend.

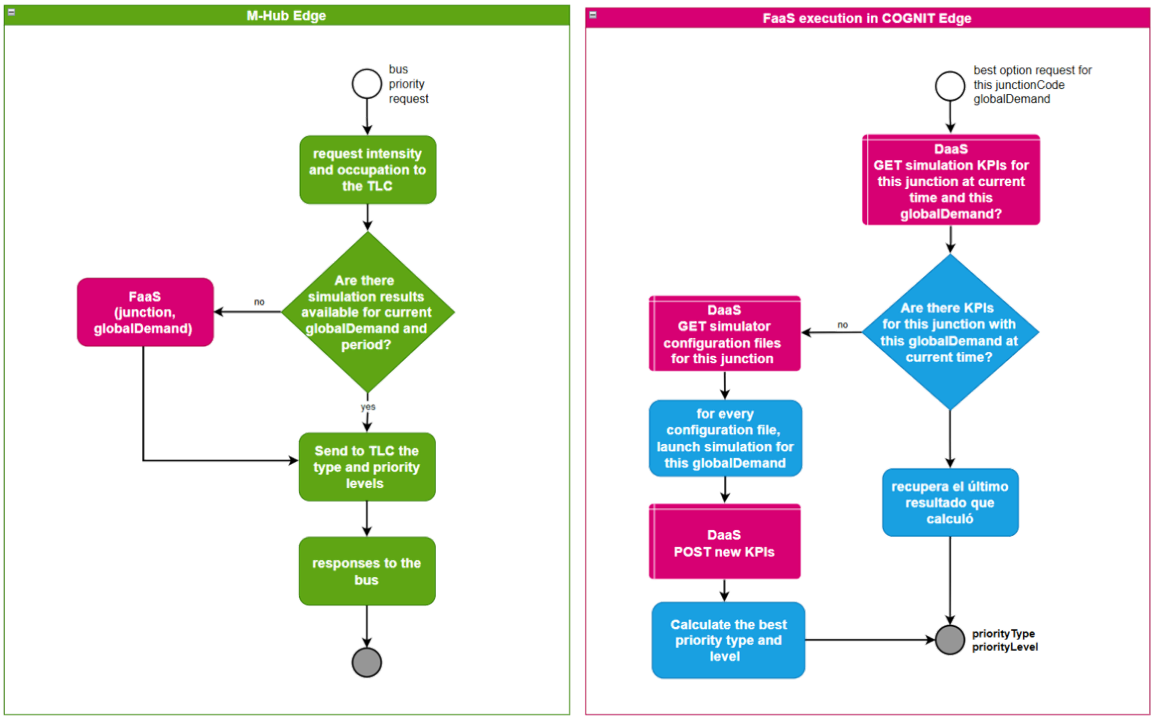

Figure 3.9. Detail of the process in M-Hub and FaaS request.

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 Silver.

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 DaaS on-premises and specific nodes.
- Locality 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.

Validation Scenario

ACISA is the current traffic management contractor in the city of Granada, and is currently equipping 38 intersections with its new M-hub and V2X systems to provide public bus

prioritisation for the bus line 4 of the city, consisting of 18 buses, as can be seen in Figure 3.10. The current status is that:

- Every intersection is configured in SATURNO, following the V2X standard.
- The basic configuration includes all incoming and outcoming 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.

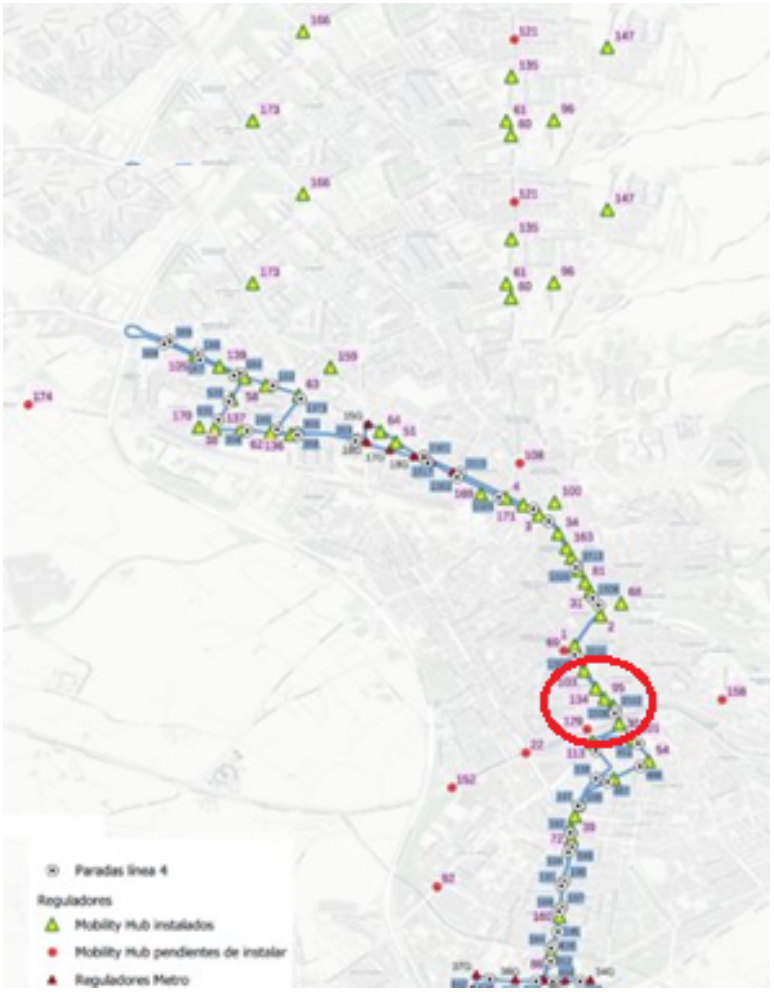

Figure 3.10. Granada's line 4 map shows the intersections covered with M-Hub and V2X equipment. In red, the selected intersections are to be used as testbed on use case 1.

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

- Line 4, consisting of a maximum of 18 vehicles, all with an OBU installed.
- Long route, with a total of 38 traffic light controllers including an RSU and an EDGE internal CPU.
- Bus delay data of each bus are updated in SATURNO every 5 minutes.
- Two of those intersections in line 4, have been selected to be used as testbed for the COGNIT Project (marked in red in Figure 3.10, and in more detail in Figure 3.11 and Figure 3.12)

- The current priority algorithm will be compared against the more sophisticated one to be developed leveraging the FaaS functionality provided by the COGNIT Framework, to trigger traffic simulations based on the dynamic model provided by its digital twin.

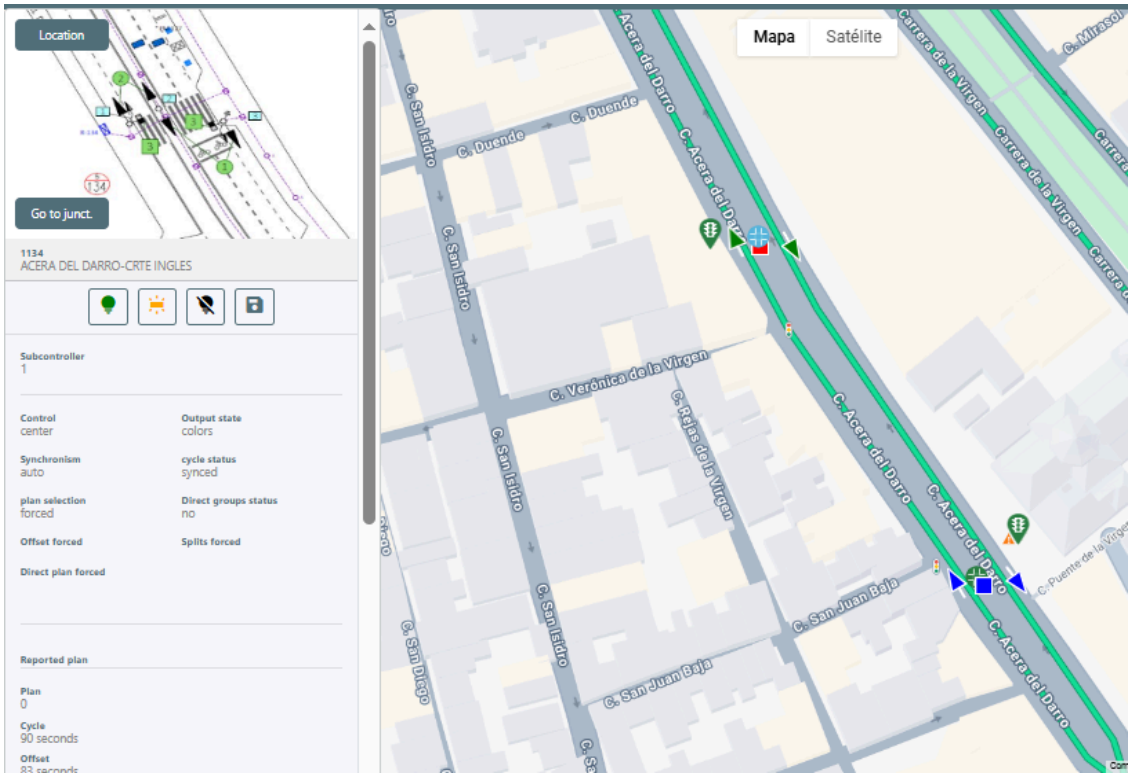

Figure 3.11. Detail of intersections used as use case 1 test bed.

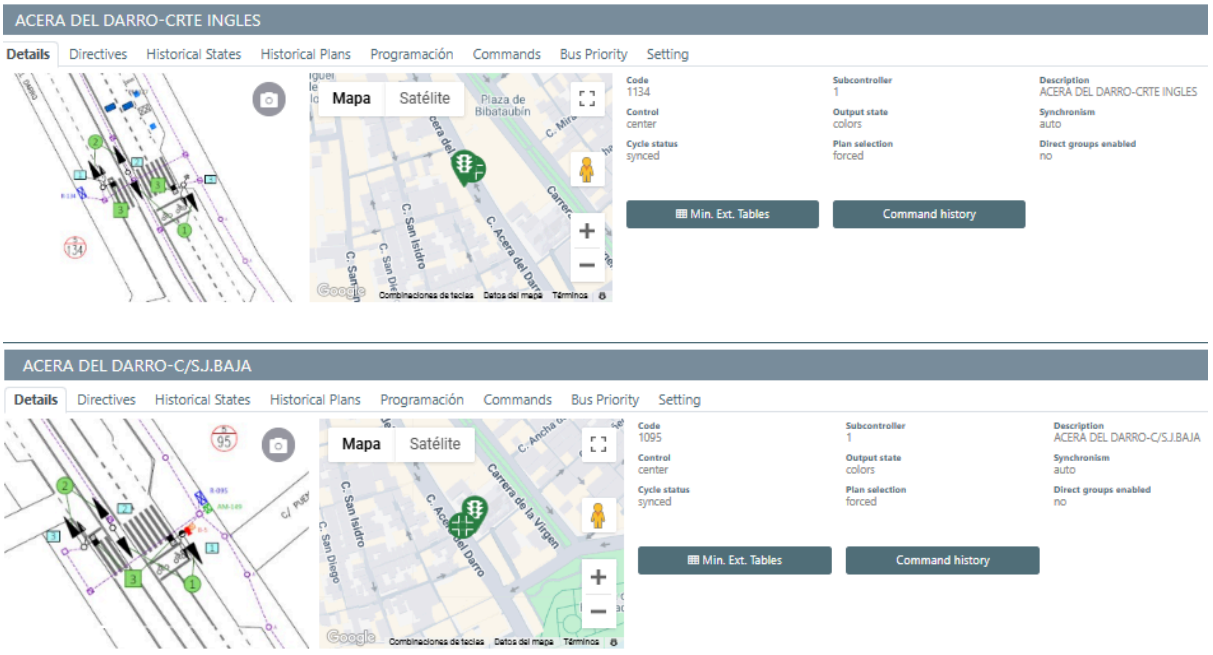

Figure 3.12. Granada's intersection 1134 and 1095 as represented in Saturno.

Use Case Testbed

The use case has set up a testbed in Granada for testing the integration of the real device on 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:

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.

Figure 3.13 shows the elements that have been deployed in Granada for the pilot project.

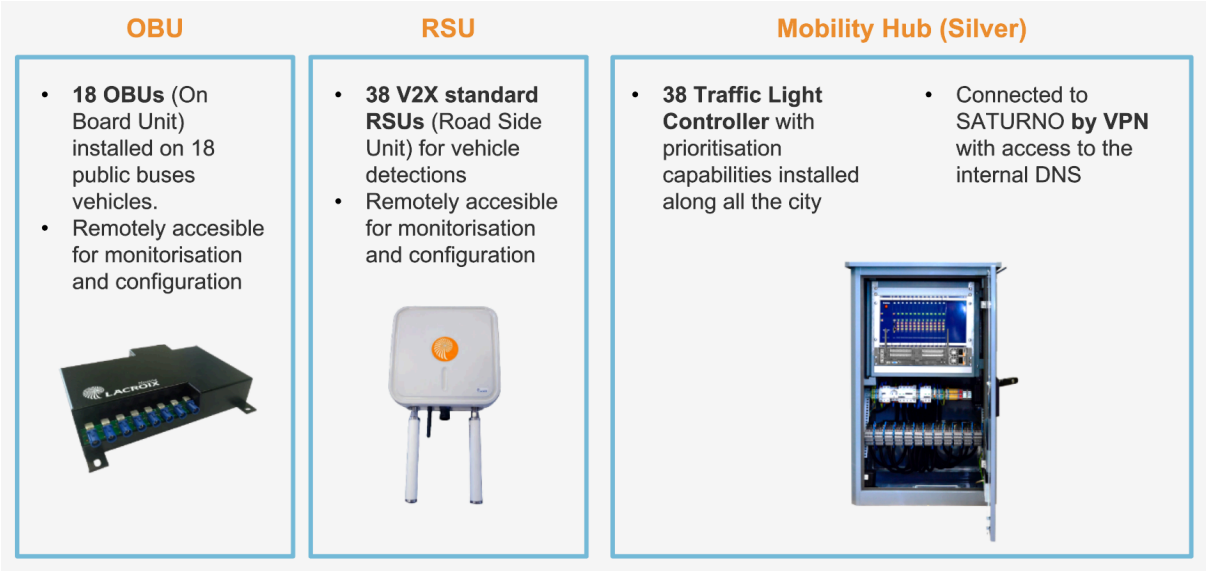

Figure 3.13. Devices deployed on the pilot on Granada.

4. Use Case #2: Wildfire Detection

The wildfire detection use case explores the use of IoT technologies for fire detection in forests. The early detection of wildfires is of utmost importance to obtain 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 because of 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 unnecessary intervention leads to an increase in costs.

The use of the COGNIT Framework can help offload power-consuming tasks and efficiently implement reliable algorithms for the verification of flame presence, thereby reducing the overall device power consumption. Moreover, it can improve the integration and management of edge and cloud resources, leading to a more reliable network even during challenging events such as fires.

4.1 Use Case Scenario and Architecture

The use case scenario and architecture are described in more detail in the previous deliverable “D5.3 Use Cases - Scientific Report - c”, within section 4.1 “Current architecture and scenario”. For convenience a summary is provided below.

The devices designed for early detection of wildfires are equipped with multiple sensors related to fire presence and emissions. The relevant parameters 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 (%).
- UV radiation to detect UV emissions from flames at a distance up to 200 m.
- Camera images.
- Geographic coordinates hardcoded in the device.

Sensors can operate in the default mode or high-alert mode. In the default mode, data from all sensors except the camera are provided every hour and saved in a database after a simple data processing step. The collected data can be utilised for various applications, including fire risk prediction and monitoring of the overall status of the forest. In high-alert mode, the sensor turns on the camera and transmits data more frequently. This high-alert mode is explained in more detail in Section 4.3 below.

Figure 4.1 below shows an example of a possible wildfire prevention network. In forests, Internet connections can be unavailable or unreliable. If the area is well covered by a public LTE network, an NB-IoT connection can be used by the devices; otherwise, one possible solution for providing connectivity to the device 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 the 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.

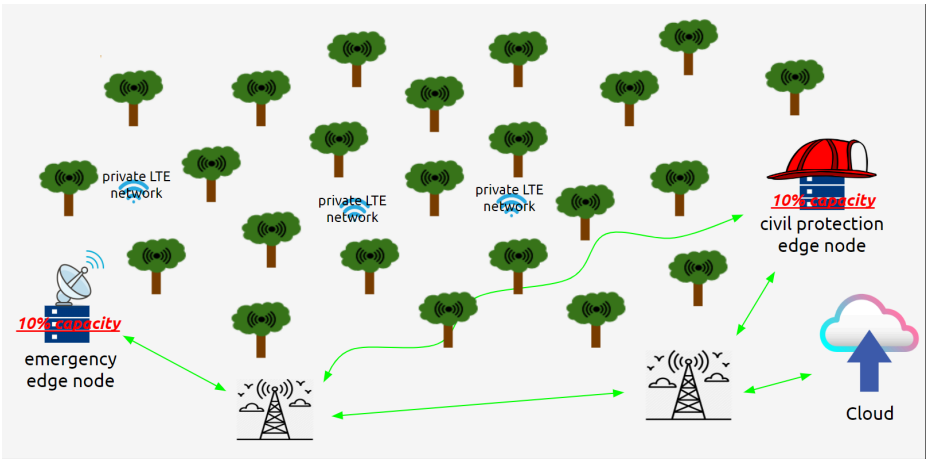

Figure 4.1. Example of a possible implementation of the wildfire early detection network

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 interferences 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 surface and 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.

The reduction in the amount of information to transfer is especially beneficial in the case of using satellite connections, resulting in a reduction in transfer time and costs. The amount of edge nodes' resources to deploy depends on the network size, but deploying 10% of the resources that are required when all the devices are in high-alert mode is suggested. 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 will help manage 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.

4.2 Summary of developments during cycle M16-M21

During the M10-M15 cycle, a suitable image recognition algorithm was found and offloaded to the COGNIT Framework using a Python program.

During the M16-M21 cycle, efforts have been focused on developing virtual simulations of the network to test COGNIT behaviour. One statistical simulation and one spatial simulation were developed as expected; the two models will be further improved and will be used for the validation of the wildfire use case.

Moreover, during this cycle, a COGNIT edge node was set up in Nature 4.0 headquarters and connected to the main testbed.

Finally, tests have been performed on the TreeTalker Fire platform. Integration with *sim7022*, an upgraded version of the previous NB-IoT module, was successfully tested by sending POST requests to a remote server. The acquisition of the image from the *OV2640* camera by *esp32* has been tested, as well as the possibility of importing and analysing the acquired images using Python. The C version of the COGNIT client has been selected as the preferred solution for microcontrollers because it requires fewer resources and is faster than MicroPython. The client will be tested in the following cycles.

In the upcoming cycle, the shared *use-case-2* GitHub folder will be updated with the complete versions of the two simulation software applications developed during the M16-M21 cycle. The image recognition function will be further improved, including the adjustments needed to work with the new COGNIT Frontend.

Further details of the improvements done during the cycle are described in the following sections.

4.3 Integration with the COGNIT Framework

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 be in default mode most of the time, characterised by low offloading frequency 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 costs and helps avoid wasting computational resources. The priorities of the two modes are different, while the default mode can use low-cost, low-carbon footprint machines to elaborate and store the data remotely, the high-alert mode requires more computational resources and a certain latency control to guarantee a certain level of service during emergencies. COGNIT can help manage these two modes and seamlessly switch between them.

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 in 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 development of improved versions of the sensor devices and the deployment of the COGNIT client on them will be used to test the interaction between real devices and the framework and to demonstrate the use case feasibility. Depending on the progress of the development, a real network might be deployed in the field by the end of the project.

A server has been bought to test and demonstrate the integration of an on-premise resource to the testbed and their interaction, as well as for future tests requiring on-premise resources.

Application structure

The UC2 reference scenario consists of the following elements:

- The sensor devices.
- One or more edge nodes.
- One or more remote servers.
- The UC2 Serverless Runtime image.
- The image recognition function and the default mode function.

The devices scan their surroundings every fifteen minutes to assess the presence of flames. If an indication of a flame is detected, the device enters high-alert mode and sends a wake-up signal to nearby devices. In high-alert mode, the device turns on the camera and offloads an image recognition function to the COGNIT Platform every minute to monitor the situation. If no fire is detected by the image recognition algorithm after fifteen attempts, the device returns to default mode. A device that receives a wake-up signal enters high-alert mode; however, the wake-up signal is only propagated further by the device if the presence of a flame is confirmed by the image recognition algorithms to avoid wasting resources in the case of false alarms. This is illustrated in Figure 4.2 below.

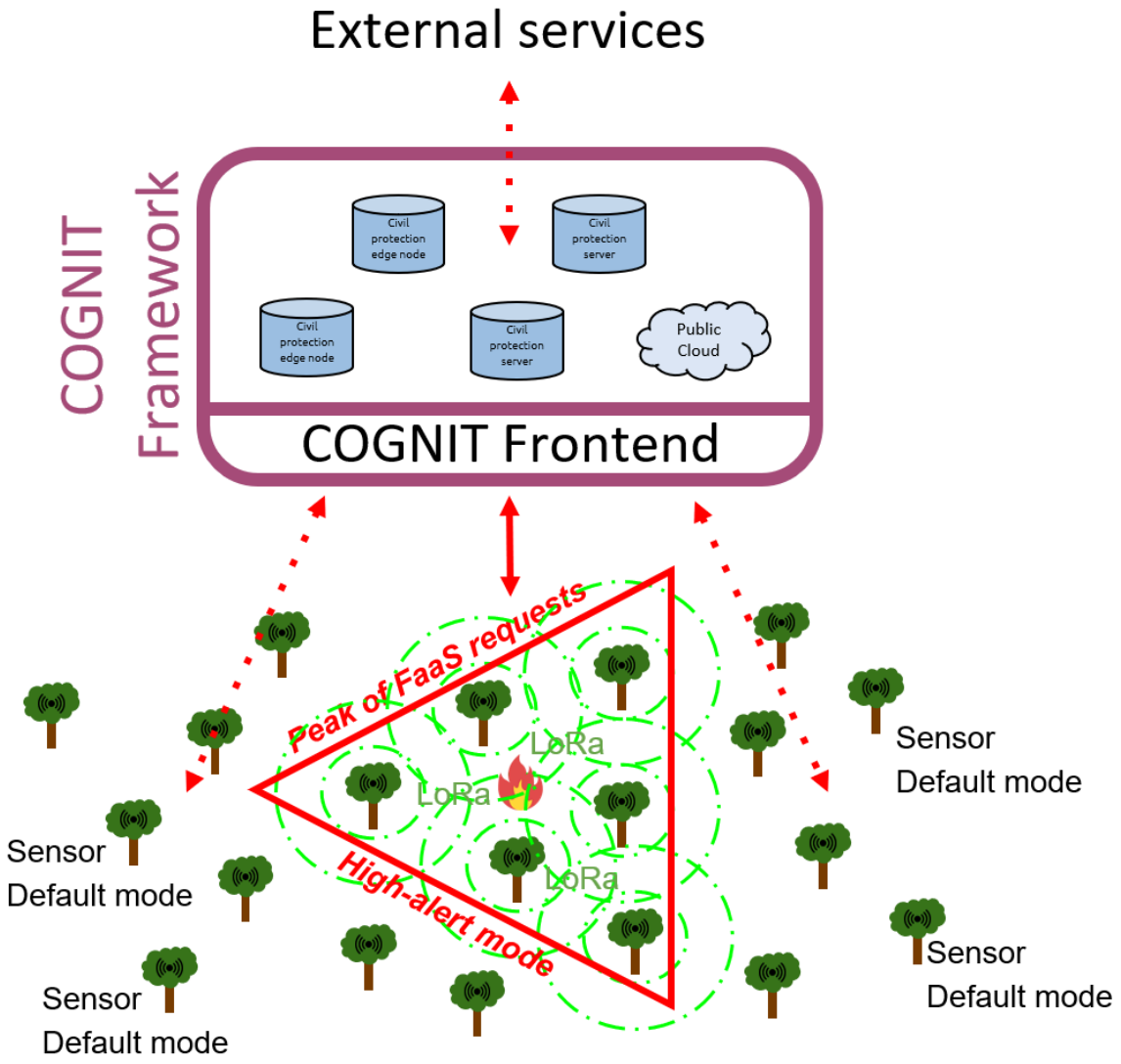

Figure 4.2. Schematisation of wildfire use case scenario

In the final release, the image recognition function and default mode function will be cached using the DaaS functionality of COGNIT currently under implementation. The functions will be retrieved by the UC2 Serverless Runtime when they must be executed on the data sent by the devices. The functions are written in Python while the device will provide its arguments using the C device client. COGNIT will run on private networks of civil protection organisations due to the sensitivity of the data and for security reasons and will manage their private servers and edge node resources. The requirements from a device will depend on which mode it is operating in. Data sent in default mode do not have priority or response time constraints; therefore, in the default mode the requirement for the COGNIT AI-Enabled Orchestrator is to use green or low-carbon footprint resources for the task. However, in the high-alert mode, there is a maximum latency requirement that depends on the distance from the device to areas with a confirmed fire. The AI-Enabled Orchestrator should prioritise using edge computing resources on premises, and prioritise processing at the edge of requests from the devices nearer to the fire. In case the

compute resources on premises are insufficient, external servers (e.g., public cloud resources) could be used to to process lower-priority requests. The purpose is to decrease the latency for the devices with the most relevant information and to use the edge resources to avoid overloading the public network during an already critical situation. The use of edge nodes allows the maintenance of the partial functionality of the system, even if the connection with the Internet is cut off.

Relevant Key Features of COGNIT

COGNIT provides multiple features for adapting to user needs and requirements. From the UC2 perspective described in the previous paragraphs, the most interesting COGNIT features are:

- **Scalability**: The computational resources required by the network during the default mode are very low, as are the number of FaaS requests, 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 should guarantee service when resources are required.
- **Forecasting**: Even if the occurrence of fires is unpredictable, the pattern of offloading requests during a fire event is progressive. A specific feature of COGNIT that the UC2 can benefit from is the possibility to run wildfire simulations to train its AI-Enabled Orchestrator to improve its performance during these events.
- **Integration**: In contrast to other FaaS solutions, the possibility of installing the COGNIT Framework on-premise and using owned resources or resources under the control of the civil protection organisations is very important since real-time data on wildfires are considered sensitive information.
- **C client**: IoT devices are implemented for real-time monitoring applications in all fields. The main programming language for microcontrollers is C++ because of its efficiency and performance even in devices with tight memory constraints. COGNIT makes available its FaaS solution to microcontrollers through its C client offering new possibilities to IoT devices, TreeTalker Cyber Fire included, in particular when combined with its caching function feature.
- **Caching functions:** The option to cache functions as part of DaaS features is also of primary interest for UC2. COGNIT Serverless Runtimes are expected to run Python code, which can be cached and used with the function arguments sent by the C client to provide the response. This feature makes Python functions available to C-programmed microcontrollers that would not be able to run the function itself due to limited resources and energy constraints. TreeTalker Cyber fire will fully take advantage of this feature running the image recognition algorithm in TensorFlow.
- **Efficiency**: The possibility of choosing options such as minimum power consumption or minimum carbon footprint is a valuable feature during the default mode, and it is a great feature in general for all applications that do not have strict latency constraints.
- **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.

Validation Scenario

This use case will demonstrate offloading of the image recognition algorithms from the TreeTalker Cyber (TTC) Fire and how COGNIT manages sudden peaks of requests.

The C function and its deployment on the TTC-fire will be tested on actual devices to demonstrate the possibility of using COGNIT on relevant target devices, using the C client implementation and the caching functions feature of the DaaS service.

The test performed on real devices will be scaled up using virtual simulations of wildfires to test the scalability, latency, forecasting features and integration with on-premise resources of the COGNIT Framework. The utilisation of virtual simulations enables the evaluation of system behaviour across multiple scenarios with minimal cost. The simulation could also be useful for potential customers and for studying sensor network configuration performance in advance.

Currently, two simulations are in the final stage of development: the statistical simulation and the spatial simulation. Both allow the user to select the number of devices involved in the simulations and modify the FaaS request frequency for the default and high-alert modes. The standard simulation settings are as follows:

- 1000 TreeTalker Cyber Fire.
- 1 hour FaaS request frequency during the default mode.
- 15 minutes scanning frequency for wildfire detection.
- 1 minute FaaS request frequency during the high alert mode.

The sensor distribution and trigger differ between the two simulations.

The function offloaded by the simulated devices will be the image recognition function for fire detection, which was tested on a COGNIT Serverless Runtime during the previous cycle.

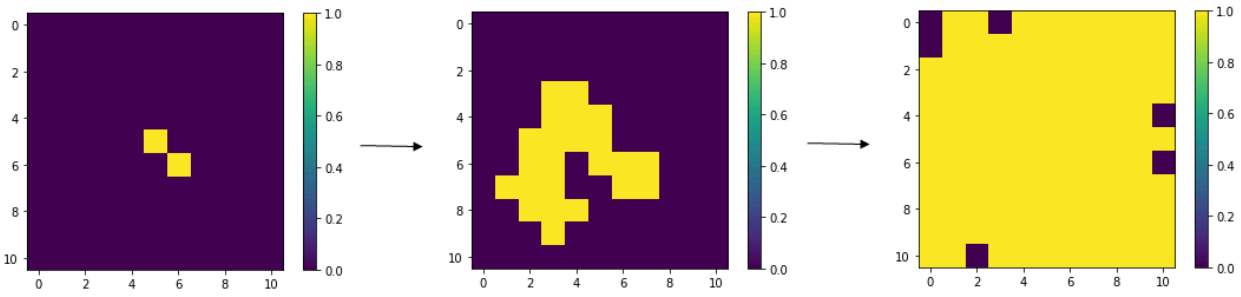

Figure 4.3. Graphical example of the statistical model. Each area represented by a square contains one sensor. The square becomes yellow when all the sensors in the area are activated. The three images show the progression of the model until all the 94 deployed sensors are active.

The statistical simulation consists of a grid that represents a uniform distribution of devices in the forest. The number of devices can be selected, and the grid will subsequently be adapted. Each square corresponds to the maximum distance that the wake-up signal can reach. The simulation starts from the central square with the detection of a flame by a single device. A wake-up signal is sent to devices in the neighbouring squares that subsequently enter the high-alert mode. The woke-up devices have a certain probability of offloading an image in which a fire is present; otherwise, an image portraying a forest without a fire is sent. If the response of the fire detection algorithm is positive, the device wakes up nearby devices. The probability of detecting a flame decreases with the increase of the distance from the central square. Moreover, the probability increases with every high-alert-mode cycle (every minute by default). The probabilities can be adapted to simulate different fire spread speeds. The combination of the statistical model with a grid structure takes into account the possibility that other devices in the area could have already been activated by other TTC Fires. The process stops when all the devices deployed are in high alert mode.

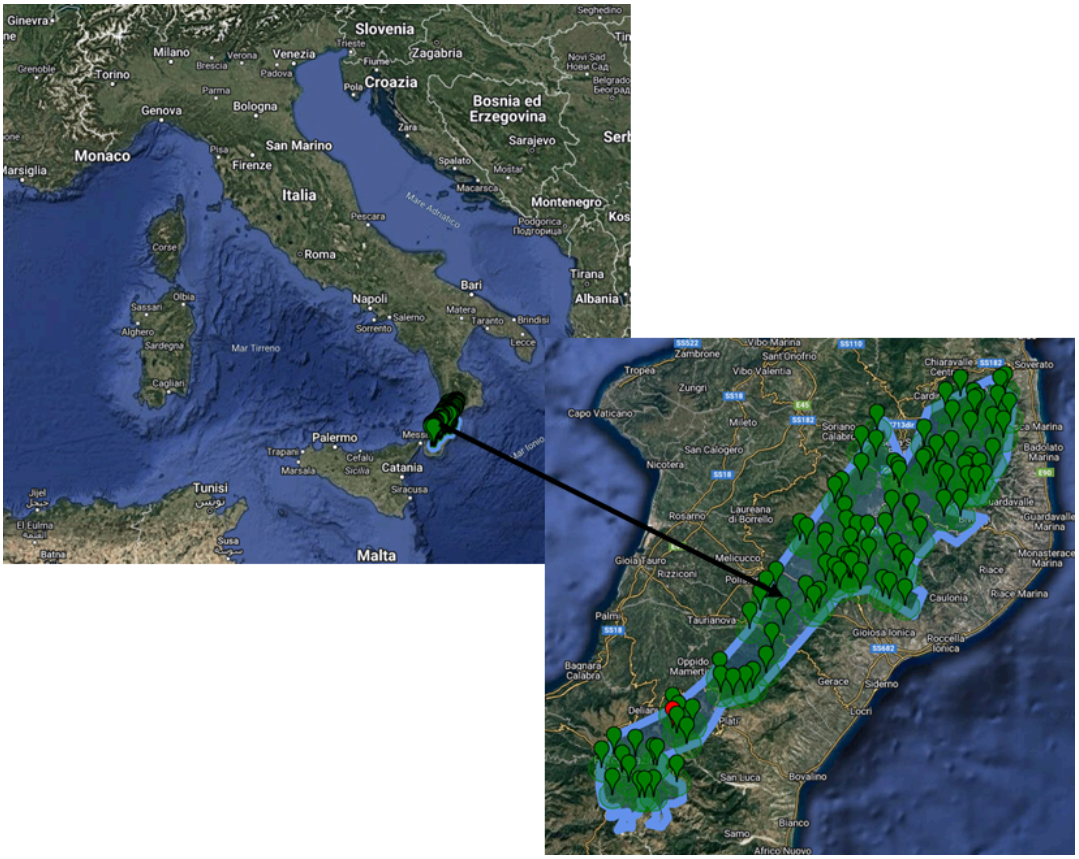

Figure 4.3. Graphical example of the spatial simulation, the 100 sensors are indicated as green marks and their field of view is represented by a green area centred on the sensor. The red mark is the fire starting point.

The spatial simulation has been improved compared with the original idea. In this simulation, an area of interest where the sensors are deployed can be selected. The

coordinates of the sensors can be inserted, as well as the starting point of the fire (which can also be randomised) and its average speed. The average field of view of the sensors (in green in the map), as well as the average distance reached by the wake-up signals, are inserted inside the program. When a sensor's field of view is crossed by the spreading flame, the device enters the high-alert mode and wakes up all reachable devices. All devices whose field of view has been intersected by the fire send an image representing a flame; otherwise, an image representing a forest is sent. The simulation runs until all the installed devices are in high-alert mode. The simulation of the spreading of the fire is currently being tested.

In the next cycle, the models will be updated for their use with the new COGNIT release and refined to be as representative as possible of a realistic scenario.

Use Case Testbed

UC2 testbed is composed of the client devices, the on-premise machine and the two simulation software applications.

The use of the *esp32-CAM* equipped with the *OV2640* camera and the design of a custom board based on the *STM32L433RCT6* microchip are discussed in D5.3. The Internet connectivity of the designed board, as well as the image acquisition using the *esp32-CAM* board, was tested. Devices are currently being developed, with improvements to the tested functionalities and integration of the remaining sensors to be ready to interact with the COGNIT Framework in the following project cycles.

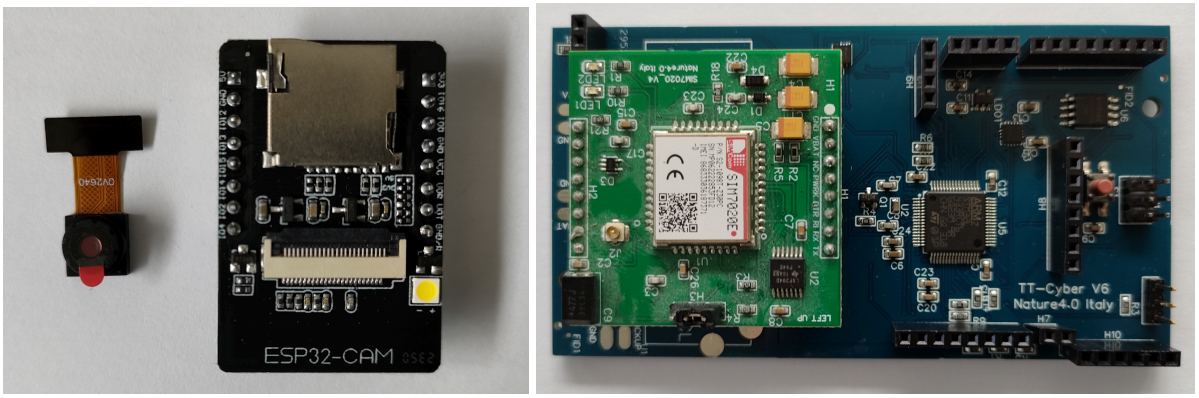

Figure 4.4. Images of the esp32-CAM and the prototype custom board based on STM32L433RCT6 microchip

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 GeForce RTX® 4090 24GB.

5. Use Case #3: Energy

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 intensity of greenhouse gas emissions, and to eliminate transmission energy losses because energy is produced and consumed locally. To make this a reality, there is a strong need to manage energy consumption as well as its 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.

5.1 Use Case Scenario and Architecture

The use case scenario and architecture are described in more detail in the previous deliverable “D5.3 Use Cases - Scientific Report - c”, section 5.1 “Current architecture and scenario”. To assist the reader, a summary is provided below.

In this scenario, the smart electricity meters run a number of user applications to manage important appliances and energy assets installed (from a 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 installations, heat pumps, electric vehicle chargers, and electric floor heating. By empowering electricity meters with apps, connected to services equipped with advanced decision-making algorithms and pre-trained AI models, they turn into highly personalised Energy Assistants.

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

This use case will mainly test the performance of the COGNIT Framework for:

- running the COGNIT client on highly resource-constrained devices;
- supporting a large number of clients (devices) in the same geographical area (energy community);
- handling dynamic changes in Serverless Runtime requirements due to changes in user preferences.

5.2 Summary of developments during cycle M16-M21

During the last research cycle, M16-M21, the work has been focused on the following topics:

- Getting the C-version of the COGNIT Device Client to run on the physical device.
- Updating the end device simulators, to support testing and demonstration of the COGNIT Framework.
- The development of an AI-based version of the decision algorithm, which will be offloaded to the COGNIT platform.

Running the C-version of the Device Client on the device

Since smart energy meters are devices with limited resources, this use case has focused on working with the rest of the project to get the Device Client in C (DCC) to work on the device, since it was deemed more suitable than the Python version. Before attempting to run DCC on the device, we tried integrating it into our Phoenix-RTOS using the Qemu emulator of IA32 architecture. This work raised some issues, most of them concerned with our network stack, especially IPv6. After resolving them, we successfully ran the DCC on the IA32 emulator and offloaded some basic functions to COGNIT Serverless Runtime.

Thereafter, we started working on putting the DCC on the smart energy meter. The current status at the time of reporting is that the communication with the COGNIT Platform works, but the function offloading is experiencing problems. These issues are connected to the network stack, but we suspect the problem might be on the side of the modem that supplies the energy meter with an Internet connection. This matter will be further investigated.

Updates to the end device simulators

End-device simulators are an important part of the testbed, used to test and verify the behaviour of the devices and the decision-making algorithm before its implementation into a real environment. They are expected to closely reproduce the operation of real devices. With this in mind, the simple battery storage model developed earlier has been replaced by a piecewise linear model with a charging power limit that decreases above a certain energy state value. It better reflects the battery charging characteristic, which consists of two distinctive parts: constant current and constant voltage (CC-CV).

A simulator of the electric vehicle (EV) battery has been implemented and integrated with the smart energy meter simulator. We used a similar linear CC-CV battery model as the one used for energy storage. The difference is that the EV version can be charged only when an EV is parked and available at home. Otherwise, the car battery can be discharged by driving with driving power. This EV battery operating logic has been added to the baseline algorithm, enabling it to take EV battery charging management into account in its decision-making process.

Development of an AI-based version of the decision algorithm

The work during the last cycle was mainly focused on research of a more advanced decision-making algorithm using AI methods. A literature review of up-to-date solutions in the field of home energy management systems (HEMS) was performed. The most attention was paid to articles in which the designed system included elements consistent with our actual test environment, i.e.: a renewable energy source, especially PV panels, a battery energy storage system (ESS), a heating and air conditioning system, an electric vehicle with a charger and uncontrolled load. The challenge turned out to be finding publications in which the proposed solution was tested not only in a modelled environment, but also implemented and verified in a real-life scenario.

Based on the literature analysis, we decided to use reinforcement learning (RL) which is currently the most common and promising approach for optimising the control of devices with unknown characteristics. In this area of machine learning, an agent interacts with a specific environment to maximise a numerical reward. The agent's goal is to find an optimal policy that defines a strategy for taking actions in relation to the environment in all possible states. In the learning process, the agent tries to balance the exploration of unknown territory with exploitation based on current knowledge. Following an action, the environment changes state and returns the reward resulting from it. This continues until the agent maximises the total cumulative reward received from the environment.

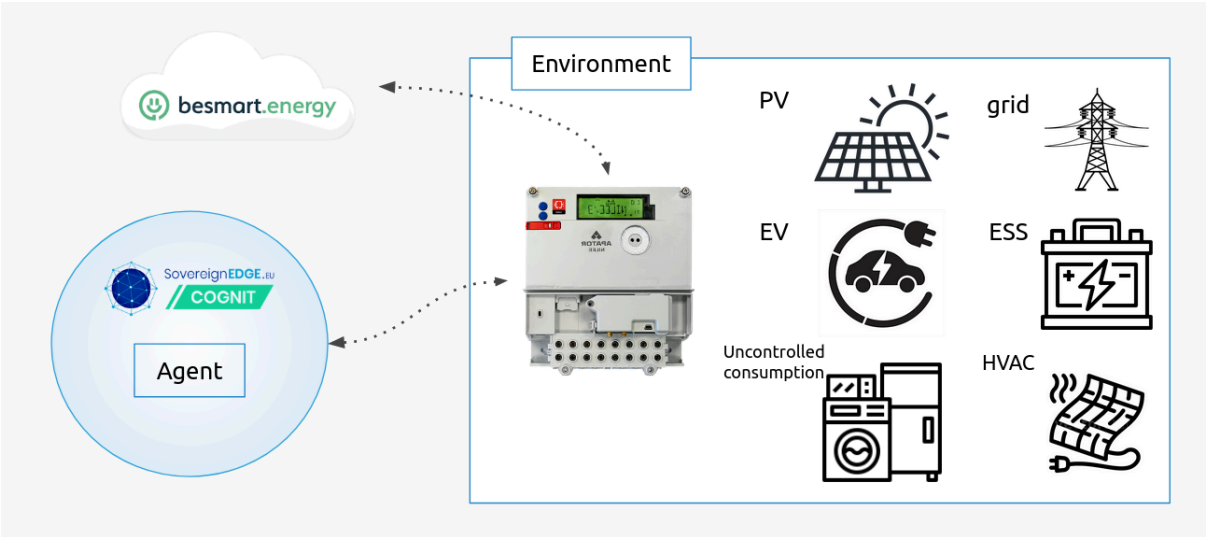

Figure 5.1. Scheme of home energy management system with AI-based decision algorithm.

The scheme of our proposed solution is presented in figure 5.1. The agent communicates directly with the Smart Meter that manages all appliances in the smart home by measuring energy production and consumption, reading current values and setting parameters on controllers of end devices. The meter has too limited memory and CPU to store trained AI models or perform complex calculations, so the algorithm responsible for selecting actions is offloaded to the Serverless Runtime. The meter also queries the besmart.energy platform for predictions of energy generation, consumption and weather forecasts so the actions can be chosen taking the future into consideration. In the environment all

important elements previously mentioned could be identified, but also uncontrollable loads are shown. Power demands of non-shiftable loads (e.g. microwaves, refrigerators and computers) must be satisfied completely without delay and cannot be interrupted. In contrast, controllable loads, such as HVAC systems, can be controlled to flexibly adjust their operating time and amount of energy consumption while meeting certain operational requirements, e.g. temperature ranges.

We consider the situation in which the supposed HEMS algorithm performs optimal 24-h ahead scheduling of appliances with a 1-h scheduling resolution. For $\forall t = 1, 2, ..., 24$, the state of the environment comprises the seven following variables:

- t_t time remaining until planned EV departure,
- E_t^{PV} energy generation from PV panels,
- E_t^{cons} energy consumption of uncontrollable loads,
- T_t^{ind} indoor temperature,
- T_t^{out} outdoor temperature,
- SOE_t^{ESS} energy level of energy storage system,
- SOE_t^{EV} energy level of EV battery.

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

$$s_t = \left(t_t, E_t^{PV}, E_t^{cons}, T_t^{ind}, T_t^{out}, SOE_t^{ESS}, SOE_t^{EV}\right). \quad (5.1)$$

It is assumed that the indoor temperature at the next time slot depends only on the indoor temperature, HVAC power input and weather in the current time slot. Moreover, according to the linear battery model, the energy level of the ESS in the next time slot is defined using only the current energy level and the current discharging/charging power. The same applies to the EV battery charge level at the next time step, which can be determined using the current energy level and charging power in case of car availability or driving power otherwise. Thus, the environment can be characterised by a Markov decision process (MDP), in which the next state s_{t+1} depends only on the current state, along with the action chosen by the agent in the current state, ignoring all previous states and actions.

The RL algorithm target is to maximise self-consumption while maintaining a comfortable temperature range and keeping the energy levels of the ESS and the EV battery within their respective thresholds. To accomplish that, the agent aims to optimally decide the values of a set of actions A that consists of:

- T^{set} temperature setting on the HVAC controller in range $\left[T_{min}^{HVAC}, T_{max}^{HVAC}\right]$,
- p^{ESS} ESS charging/discharging power in range $\left[-P_{max}^{ESS}, P_{max}^{ESS}\right]$,
- p^{EV} EV 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, ESS and EV related to the consumer's preferred comfort and appliance's operation characteristics. The total reward is expressed as:

$$R_t = -\left(\beta^E \left|E_t^{grid}\right| + \beta^{HVAC} c_t^{HVAC} + \beta^{ESS} c_t^{ESS} + \beta^{EV} c_t^{EV}\right), \quad (5.2)$$

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

$$E_t^{grid} = E_t^{PV} - \left(E_t^{PV} + E_t^{HVAC} + E_t^{ESS} + E_t^{EV}\right), \quad (5.3)$$

where E_t^{HVAC} represents the energy consumption of the HVAC and E_t^{ESS}, E_t^{EV} represent the energy charging/discharging of the ESS and EV, 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:

$$c_t^{HVAC} = max\left(T_t^{min} - T_t^{ind},\ 0\right) + max\left(T_t^{ind} - T_t^{max},\ 0\right). \quad (5.4)$$

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 ESS is expressed as:

$$c_t^{ESS} = max\left(SOE_{min}^{ESS} - SOE_t^{ESS},\ 0\right) + max\left(SOE_t^{ESS} - SOE_{max}^{ESS},\ 0\right). \quad (5.5)$$

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

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

$$c_t^{EV} = max\left(SOE_{min}^{EV} - SOE_t^{EV},\ 0\right) + max\left(SOE_t^{EV} - SOE_{max}^{EV},\ 0\right),\ if\ t \in \left[\omega_{arr},\ \omega_{dep}\right] \quad (5.6)$$

$$c_t^{EV} = max\left(SOE_{dep}^{EV} - SOE_t^{EV},\ 0\right),\ if\ t = \omega_{dep} \quad (5.7)$$

$$c_t^{EV} = 0,\ otherwise\,. \quad (5.8)$$

Similar to the operation of ESS, the penalty results from the SOE_t^{EV} exceeding the allowable range specified by SOE_{min}^{EV} and SOE_{max}^{EV}. Unlike the ESS reward function, it occurs only when the EV is available at home, so for time t between time of EV arrival ω_{arr} and departure ω_{dep}. Additionally, if SOE_t^{EV} is lower than consumers preferred SOE_{dep}^{EV} at the time ω_{dep}, the dissatisfaction cost increases due to the insufficient level of energy in 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. $R = \sum_{i=0}^{\infty} \gamma^i R_{t+i}$, where a discounting factor $\gamma \in [0,\ 1]$ is used to explain the relative importance of the current and future rewards.

To solve the MDP problem defined above and find the optimal policy π under which agent acts, we propose an algorithm based on Deep Deterministic Policy Gradients (DDPG). DDPG is a type of actor-critic method in which an agent is divided into two parts. The actor network returns the probability of the action a the agent chooses in a given state s_t. Then, a and s_t are provided as input to the critic network, output of which is the numerical future value the agent would obtain in the final state. The critic network updates a function that distinguishes between action and value based on two mechanisms, i.e., memory replay and target networks. At the same time the policy gradient can be computed in the direction suggested by the critic network and used to update the actor network parameters. In contrast to simpler RL methods, DDPG is capable of dealing with continuous spaces of both states and actions.

The algorithm has been trained on real data from a smart energy meter providing energy production and consumption data covering a couple of months in 1-hour-slots. Historical weather forecasts from our numerical model for the same localisation and time were used to provide outside temperatures. One training episode over which reward was accumulated was assumed to be 24 hours long, but the episode starting hour was random. Initial state values of indoor temperature, storage and EV battery levels of energy were also randomly drawn from normal distributions in the allowable ranges. Transitions to the next state were simulated with the support of models of heating systems, storage and EV batteries. Example learning curves for different coefficients of individual reward function factors are shown in Figure 5.2.

Based on the attached curves, it can be seen that the algorithm converges, but the training is not well stabilised. During the experiments, there were cases when the reward at the end of a given number of episodes was worse than the best result during training. We are still in the testing phase and we need to perform more experiments in order to determine the right parameters and final architectures of the actor and critic networks.

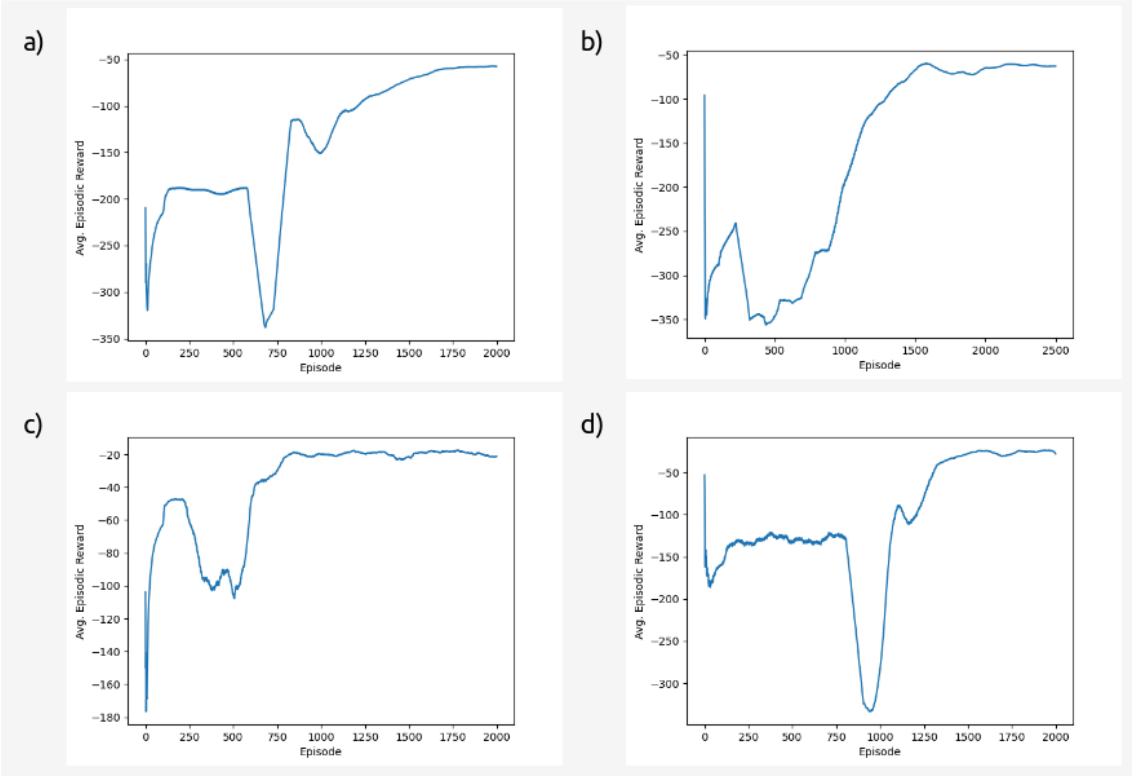

Figure 5.2. Learning curves of DDPG-based HEMS algorithm for different coefficients of individual reward function factors: a) $\beta^{E} = 0.3, \beta^{HVAC} = 1.5, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; b) $\beta^{E} = 0.3, \beta^{HVAC} = 2.0, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; c) $\beta^{E} = 0.1, \beta^{HVAC} = 1.5, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$; d) $\beta^{E} = 0.1, \beta^{HVAC} = 2.0, \beta^{ESS} = 1.0, \beta^{EV} = 1.0$.

In the simulations performed so far, the agent has learned to optimally control devices in terms of energy, assuming knowledge of the exact values of energy generation and consumption for 24 hours in advance. Once this is achieved, we made sure that the training conditions in which the agent makes decisions are more realistic by introducing predictive data generated by ML models within the besmart.energy platform. The algorithm managed to converge on a small artificially generated dataset, but we still need to overcome some difficulties in training on real household data. This step is extremely important in terms of the ability to make correct decisions in real time, since generating perfect forecasts is impossible. We expect the agent to learn the error characteristics of these signals and find a policy that takes them into account.

Updates to the Use Case codebase

In this cycle, the Energy Use Case developed the software by making modifications and introducing the new functionality in the repositories, which are part of the COGNIT Project's GitHub. These are the following:

1. use-case-3 – repository with Energy Use Case Basic Demo. In version 1.0.1 we introduced the home_energy_management package as the source code of end device simulators and the decision-making algorithm. Next, the demo in version

1.0.2 was extended to include the use of an EV simulator in the user application. Also preferences and parameters for the new device were added. Moreover, we are working on releasing version 1.0.3 in which the demo is being integrated with the AI-based version of the decision algorithm.

2. `use-case-3-home-energy-management` – repository with the Decision Algorithm and Device Simulators. Version 1.0.0 was an initial release with a baseline decision algorithm used to optimise energy management at home and simulators of end devices such as photovoltaic arrays, simple electronic devices with uncontrollable load, heating systems and battery energy storage systems. Version 1.0.1 introduced a simulator of a new appliance - EVs, whose batteries can be charged when at home and discharged by driving. The possibility of managing EV charging was added as part of the decision algorithm. In the near future, we will release version 1.0.2 including the first approach to running the trained HEMS algorithm based on DDPG. The repository contains the example of offloading the decision-making function to the Serverless Runtime.

Both repositories are provided with the BSD 3-Clause licence and are publicly available.

5.3 Integration with the COGNIT Framework

Running user apps on the smart energy meter is possible thanks to the Phoenix-RTOS operating system, which offers the needed mechanisms for effective partitioning. As a result full safety through the separation of the partition for user apps and Distribution System Operator (DSO) partition can be achieved, including the legally relevant Measuring Instruments Directive part, see Figure 5.3. However, the smart energy meter is not powerful enough to run AI algorithms directly on the device. Therefore, user apps are equipped with the COGNIT Device Client which allows for offloading compute-intensive tasks to the Cloud-Edge Continuum.

In the Energy Use Case, the common situation is that many devices are in the same geographical area (district of houses). Because of this, the Cloud-Edge Continuum needs to have enough resources available to handle requests from a large number of devices in a given time. Identified metrics/features relevant to the common situation are:

- availability of resources in the Continuum;
- response time with results of execution;
- locality and confidentiality of sensitive data;
- usage of green energy for executing calculations.

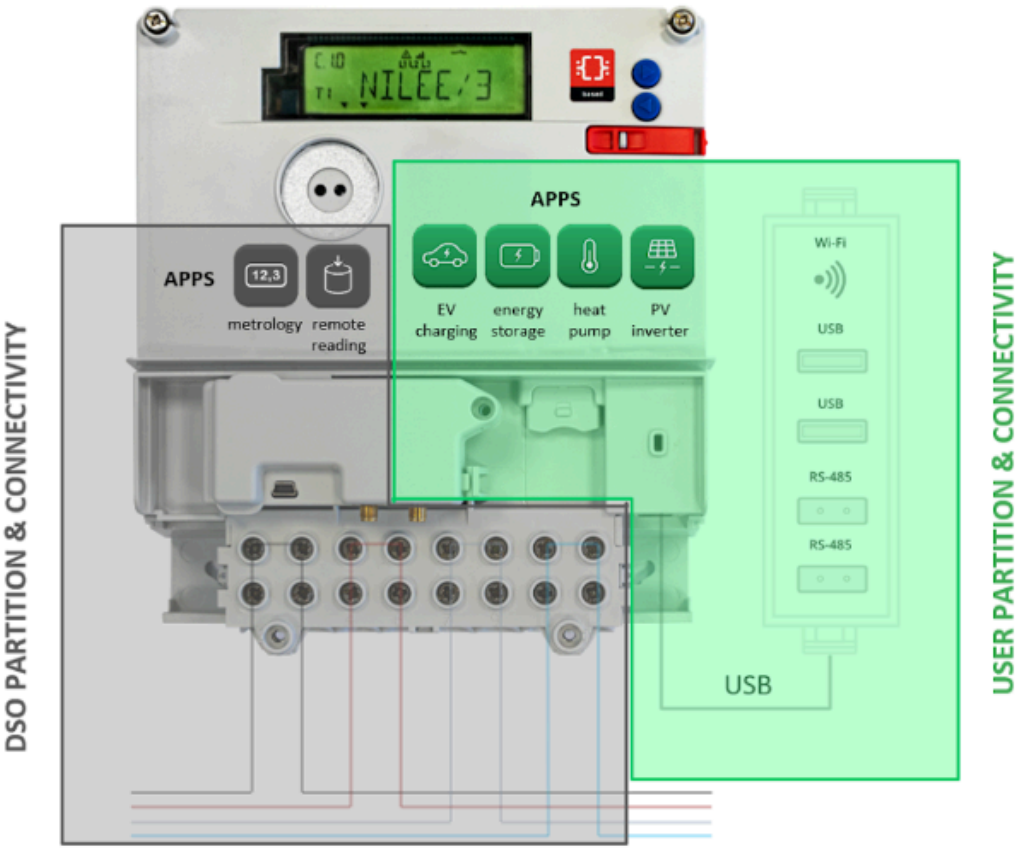

Figure 5.3. Smart energy meter

Application structure

To optimise green local energy usage, decision algorithms should consider weather forecasting data and other relevant predictions, e.g. energy cost, energy consumption or production. Such services are provided by the external smart energy management cloud platform *besmart.energy*, which provides weather forecasts data and predictions about energy consumption and production, as well as energy cost predictions.

Energy communities could benefit by using rented (from DSO) space in the user partition for running personalised user apps and lower energy usage from a grid by using locally produced green energy more efficiently. Energy communities could own distributed edge computation resources and use the COGNIT Framework to manage these resources. Figure 5.4 shows the basic application architecture.

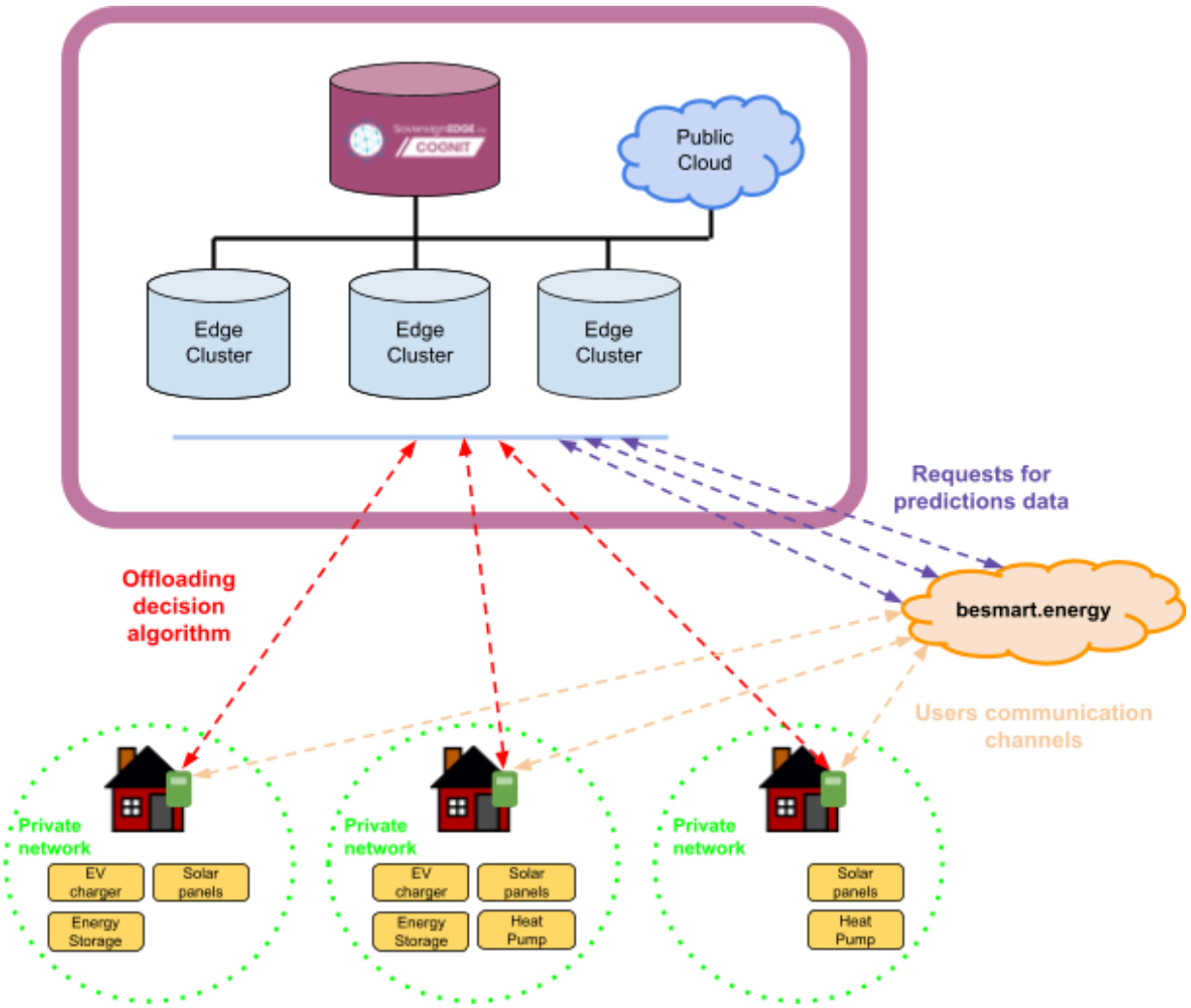

Figure 5.4. Architecture for the Energy Use Case.

The process diagram for the application is shown in Figure 5.5 below:

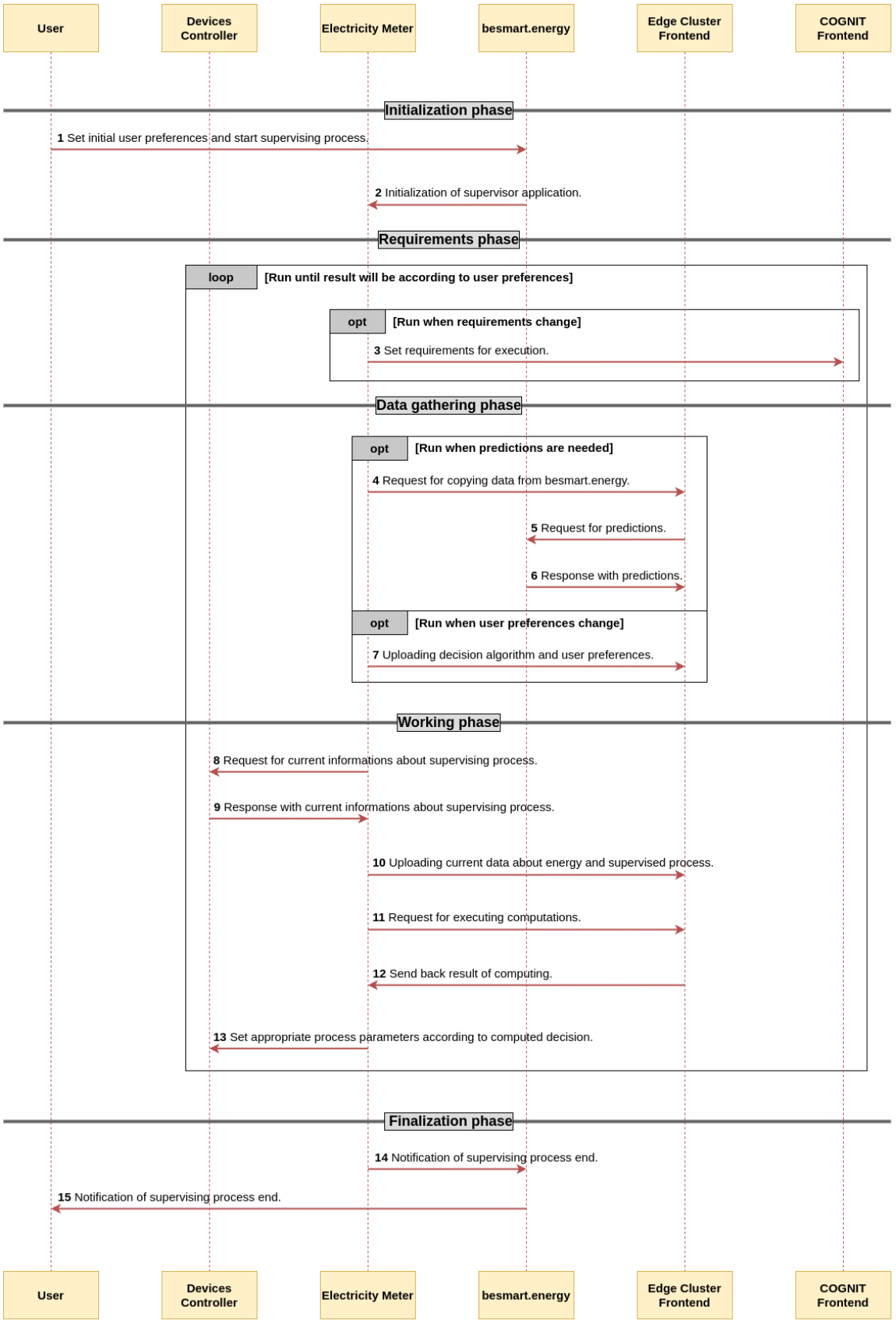

Figure 5.5. Process diagram for a single energy meter.

Relevant Key Features of COGNIT

Powering smart electricity meters with AI algorithms is a difficult task because of the resource constrained devices. The sensitivity of operated data is high (energy consumption data and energy production data of households). Another challenge is device heterogeneity: there are a large number of devices with varying resources and capabilities for execution of functions. Due to these challenges key features of COGNIT are:

- Device Client with low memory footprint (SDK in C language);
- keeping sensitive data close to its origin;
- automatic orchestration of workloads across the Cloud-Edge Continuum;
- automatic scaling up/down of Serverless Runtimes due to changes in requirements.

Validation Scenario

Due to validating responsibilities in the project the following validation scenarios environments have been identified:

- A laboratory environment will consist of smart electricity meters equipped with user applications which, by using COGNIT's Device Client, offload the decision algorithm and wait for execution results. Simulators of energetically important appliances will be run on external devices (other than electricity meters). On-prem edge nodes will be run in a laboratory environment. Local edge nodes should be the most preferred place for executing decision algorithms due to their near geographical placement.
- The simulation environment will consist of a large number of Energy Use Case demo instances, which are going to simulate different scenarios with dynamic changes of requirements for Serverless Runtimes. Here the local edge node should be the most preferred place for executing decision algorithms due to near geographical placement.
- A real-life testground in the form of a household with real appliances like an EV charger, PV installations, a heating system and an energy storage. This environment will also be equipped with a smart energy meter.

These environments will allow for validation of the relevant scenarios related to the Energy Use Case.

Request patterns

In the presented reference scenario regarding a single house and a single smart electricity meter, the following request patterns are relevant:

- Maximum resolution of requests for offloading the decision-making algorithm from the Device Client to the COGNIT Framework is 1 minute. It means that every minute new results (devices parameters) should be calculated and propagated to the supervised processes (e.g. energy storage charging).

- Due to the dynamic needs of the end user an asynchronous event requesting a stop of a supervised process could happen at any time (e.g. need for stopping an EV charging process).
- Due to the dynamic needs of the end user an asynchronous event requesting a change of some user preferences could happen at any time (e.g. room temperature or EV charging rate), of which some could influence the Serverless Runtime requirements.

Use Case Testbed

The current deployment is made in a house in Poland, forming the testground. The testground consists of following elements:

- Two photovoltaic installations with power of 14,6 kWp and 7,3 kWp. These are managed by inverters connected to WLAN.
- The energy storage unit with capacity of 12,0 kWh which is connected to the photovoltaic installation with power of 7,3 kWp. Energy storage is connected to the inverter using an RS485 interface and the Modbus protocol.
- The electric vehicle with battery capacity of 107,8 kWh and an EV charger with power of 22 kW connected to the WLAN network with possible integration using the OCPP protocol.
- The heating system consists of a smart home HUB connected to the WLAN network, with an HTTP REST API, resistive heating mats, heaters and temperature sensors. The heating system is used in 25 rooms and all of the actuators could be controlled via smart home HUB.
- The smart electricity meter is connected to the LAN which is measuring the energy parameters of the house, and is also capable of communicating with other appliances.

6. Use Case #4: Cybersecurity

Use Case 4 of the project focuses on Cybersecurity and highlights the utilisation of the COGNIT Framework through the implementation of an anomaly detection scenario within a rover (vehicle).

6.1 Use Case Scenario and Architecture

The use case scenario and architecture are described in more detail in the previous deliverable “D5.3 Use Cases - Scientific Report, section 6.1: Current architecture and scenario”. A summary is provided below.

The current architecture consists of the following components:

- **Rover data collection**: Rovers collect data, including system logs and metrics such as location, speed, and distance between vehicles.
- **Data transfer**: On the one hand, the collected data is transmitted to the anomaly detection, which is deployed at the edge cluster. This step aims to ensure fast (low latency) and secure transmission of the data to the detection system. On the other hand, data will also be transmitted via DaaS when it is available for post-processing (e.g. in-depth analysis, federated learning)
- **Anomaly detection**: Anomaly detection is performed using a Serverless Runtime, with lightweight models running on the edge to reduce latency and heavier models running in the cloud. This component analyses the incoming data to identify any significant deviations from normal behaviour patterns.
- **Migration Management**: A crucial aspect of Use Case 4 is to demonstrate the ability of the COGNIT Framework to manage vehicle migration and Serverless Runtime availability based on the rover’s route and movement. This feature ensures service continuity and operational efficiency.

6.2 Summary of developments during cycle M16-M21

During the M16-M21 cycle, we trained a LAnoBERT-based model for anomaly detection in authentication logs. We focused on aspects concerning the collection of data from rovers to be sent to anomaly detection, while integrating the use of COGNIT components, in order to ensure compliance with requirements and specifications.

Rover Data Collection

We developed a custom program to facilitate the automated collection of new entries from the authentication log (auth.log) on the rover system. This program functions by continuously monitoring the log file, detecting any newly written lines or blocks of lines in real-time. As soon as a new entry is added to the log file, whether it is a single line or a set of multiple lines written together, the program captures the latest addition.

Once the program detects a change, it retrieves the newly appended log entries and immediately passes them on to the LAnoBERT model for analysis using the COGNIT

Framework. By feeding this real-time data into the model, we enable LAnoBERT to promptly analyse and detect any potential anomalies, such as unauthorised access attempts, brute-force attacks, or irregular patterns of failed logins. This integration ensures that the system can continuously monitor authentication activities on the rover and respond quickly to any security threats as soon as they are identified.

The combination of real-time data collection and anomaly detection through LAnoBERT enhances the system's ability to identify suspicious activity without requiring manual log inspection, thus improving the overall security posture and responsiveness of the rover's operations.

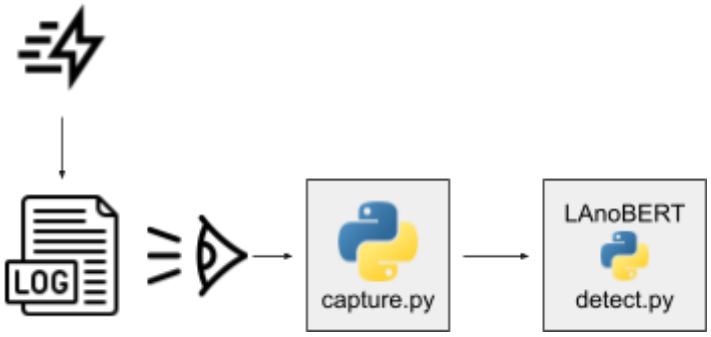

Figure 6.1. Log file capture and analysis diagram.

Anomaly Detection: Authentication Logs

LAnoBERT is an anomaly detection model itself based on BERT (Bidirectional Encoder Representations from Transformers). It is specifically designed to identify anomalies in system logs using a parser-free model, meaning it does not need predefined templates to understand the logs. This model is pre-trained to predict hidden elements in log sequences, which makes it possible to identify anomalies as deviations from normal patterns.

LAnoBERT works in four main steps:

1. Data preprocessing: Logs are normalised, in particular by replacing IP addresses (IP), port numbers (NUM), username (USER) and other specific information with generic tokens. This makes it possible to standardise the data.
   Before normalisation :

```
Unset
Oct 14 12:34:56 server sshd[12345]: Failed password for invalid user admin
from 192.168.1.100 port 22 ssh2
Oct 14 12:35:05 server sshd[12345]: Failed password for root from
192.168.1.100 port 22 ssh2
Oct 14 12:35:08 server sshd[12345]: Accepted password for user1 from
192.168.1.101 port 22 ssh2
```

After normalisation :

```
Unset
Oct 14 TIME sshd[NUM]: Failed password for invalid user USER from IP port
NUM ssh2
Oct 14 TIME sshd[NUM]: Failed password for root from IP port NUM ssh2
Oct 14 TIME sshd[NUM]: Accepted password for USER from IP port NUM ssh2
```

2. Training: The model uses BERT's hidden modelling. Part of the log sequences is hidden, and LAnoBERT learns to predict the hidden elements based on the surrounding context. LAnoBERT is pre-trained on normal logs. In these logs, occasional authentication failures may be normal, but a repeated series of failures or unknown user logins would be less frequent.

3. Testing phase: During the testing phase, LAnoBERT will mask certain words and try to predict them based on the context. For example, it could mask the word "Accepted" or "Failed" and try to predict the correct word based on the sequence. If a sequence like the one with multiple failed attempts coming from the same IP address (e.g. 192.168.1.100) is encountered, the model could identify this series of failures as anomalous because it differs from what it has seen in the normal logs.

4. Anomaly score calculation: For each keyword (e.g. "Failed", "password", "root", etc.), an anomaly score is calculated. The words with the highest scores (indicating anomalous behaviours) are selected to indicate suspicious areas of the log. If for example, failed login attempts for non-existent users or for the root user (which is often a sign of attack) are too frequent in a short period of time, LAnoBERT could assign a high anomaly score to these events.

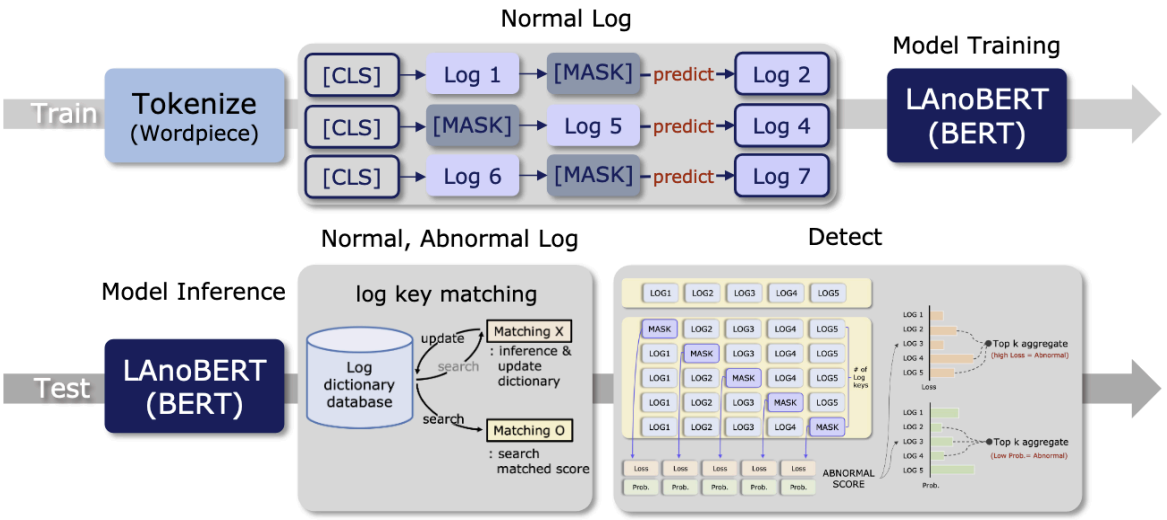

Figure 6.2. Architecture of LAnoBERT

In this research, LAnoBERT was applied to the analysis of authentication logs, and more precisely on the auth.log file coming from the rovers.[7] This file contains crucial information about login attempts (successful or failed) as well as other security-related events.

Using LAnoBERT for anomaly detection in these logs has several advantages:

- It can identify suspicious patterns in login attempts, such as brute force attacks, where an attacker tries multiple combinations of credentials.
- LAnoBERT is able to spot abnormal access attempts, such as repeated failed logins for invalid users or for the “root” account, which is often a target for attackers.

Below, we provide a concrete example of normal and abnormal authentication logs to illustrate how the model works.

Normal Logs:

The example log below shows successful logins and some occasional password failures, likely related to human error. The model will identify these logs as normal, as they follow typical authentication patterns.

```
Unset
Oct 14 12:45:01 server sshd[12345]: Accepted password for roveradmin1 from
192.168.1.101 port 22 ssh2
Oct 14 12:46:07 server sshd[12346]: Failed password for roveradmin1 from
192.168.1.102 port 22 ssh2
Oct 14 12:46:12 server sshd[12346]: Accepted password for root from
192.168.1.102 port 22 ssh2
Oct 14 12:50:33 server sshd[12347]: Accepted password for roveradmin2 from
192.168.1.103 port 22 ssh2
```

Abnormal Logs:

The example log below, however, shows repeated failed login attempts, coming from the same IP address. This behaviour is typical of a brute-force attack. LAnoBERT will identify these logs as abnormal due to the high frequency of failures and the use of invalid accounts.

```
Unset
Oct 14 12:55:03 server sshd[12348]: Failed password for invalid user admin
from 192.168.1.150 port 22 ssh2
Oct 14 12:55:10 server sshd[12349]: Failed password for root from
192.168.1.150 port 22 ssh2
Oct 14 12:55:15 server sshd[12349]: Failed password for root from
192.168.1.150 port 22 ssh2
```

[7] Yukyung Lee, Jina Kim, and Pilsung Kang. 2023. LAnoBERT: System log anomaly detection based on BERT masked language model. Appl. Soft Comput. 146, C (Oct 2023). https://doi.org/10.1016/j.asoc.2023.110689

```
Oct 14 12:55:20 server sshd[12349]: Failed password for root from
192.168.1.150 port 22 ssh2
Oct 14 12:55:25 server sshd[12350]: Failed password for invalid user guest
from 192.168.1.150 port 22 ssh2
```

The research conducted during the M16-M21 cycle allowed us to develop and test the LAnoBERT model in the context of anomaly detection in authentication logs. The results are promising, demonstrating the model's ability to efficiently identify suspicious behaviours in system logs while avoiding the need for traditional log analyzers. These first results open up prospects for applying the model to other types of logs in various operational contexts.

However, to improve both efficiency and adaptability, our next goal is to refine the LAnoBERT model. The main objective of this refinement phase is twofold:

- Optimisation: we aim to refine the model to improve its accuracy in detecting anomalies while reducing the computational load to be more efficient when running in a serverless execution. This process will involve additional training of LAnoBERT on more diverse log data, which will help it generalise better and detect a wider range of suspicious behaviours with fewer false positives.
- Lightweight deployment via COGNIT: A key aspect of the fine-tuning effort is to make the model more lightweight for efficient deployment at the edge. This optimisation is particularly important because LAnoBERT is deployed in a Serverless Runtime.

6.3 Integration with 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 rover and the need for real-time anomaly detection. Figure 6.3 shows a high-level diagram of the architecture for the use case integration with COGNIT.

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 (SR) 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.

The Device Client plays a crucial role in ensuring that the Service Level Agreement (SLA) is respected. If a violation occurs, such as excessive latency, the Device Client can autonomously trigger a switch to a more suitable edge node defined by the AI-Enabled Orchestrator.

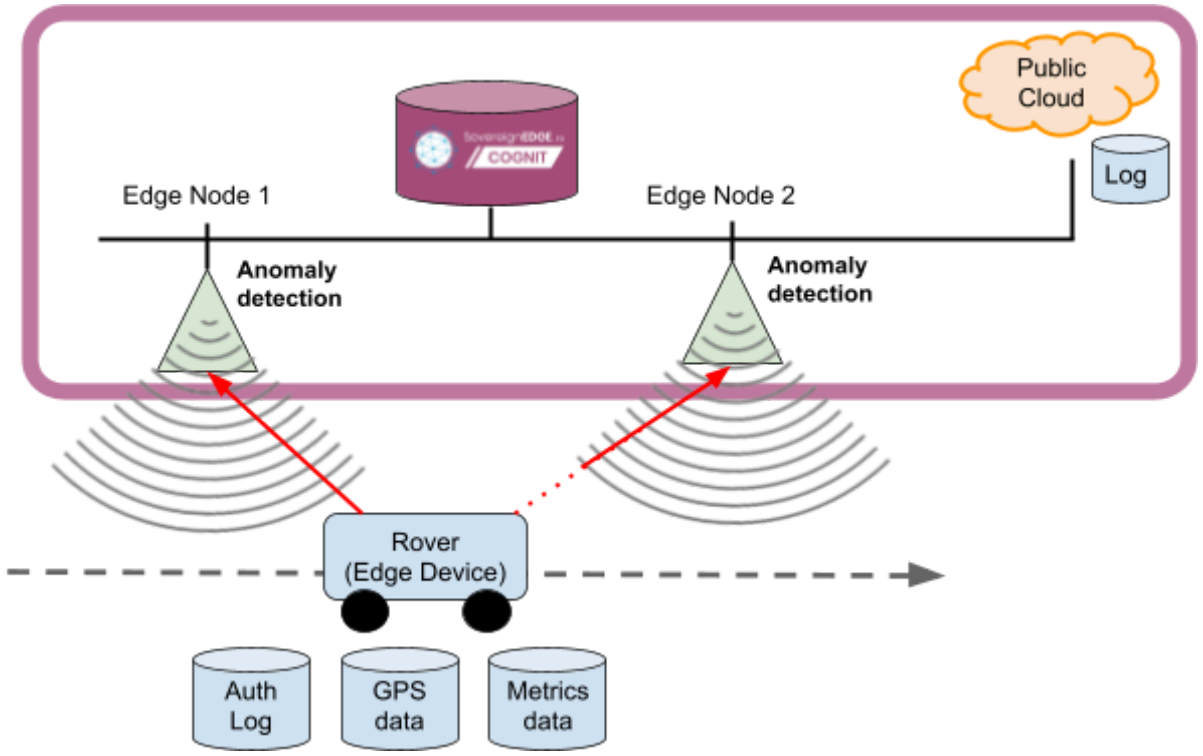

Figure 6.3. High-level architecture for the Cybersecurity Use Case.

Application structure

The cybersecurity case study will use the COGNIT Platform to manage offloading of an anomaly detection function to the edge. The Device Client communicates with the COGNIT Frontend to determine a suitable edge cluster for offloading the function, and maintains a connection with the selected edge cluster, until the device or the COGNIT Frontend requests a migration 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 create the necessary software stack to deploy the anomaly detection function. The Device Client then offloads the anomaly detection function to the edge, by passing system logs as parameters to the function. This is demonstrated in Figure 6.4 below.

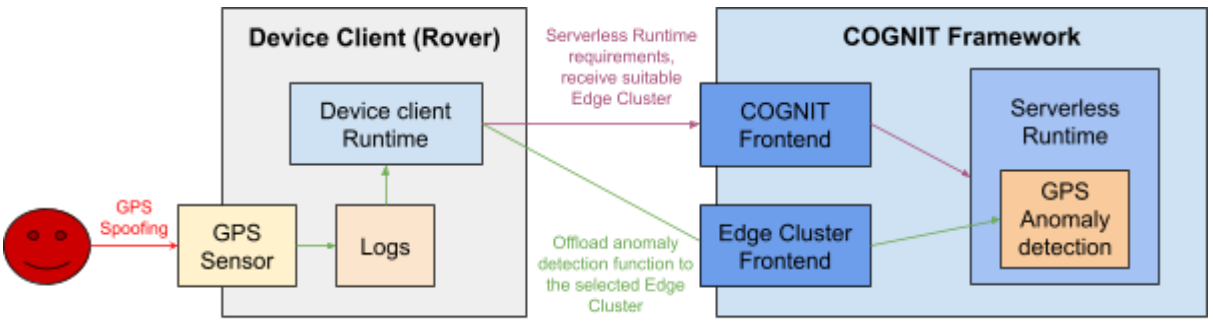

Figure 6.4. Device Client interaction with COGNIT Framework for log's anomaly detection.

As the platoon of rovers progresses towards its destination on the road network, the response time (RT) of the anomaly detection function that is running on an edge cluster needs to be monitored with respect to its SLA. If the RT deteriorates too much, then the anomaly detection function needs to be redeployed from one edge cluster to another with a better RT that will respect the SLA RT. Based on the latest version of the COGNIT Framework, it is the device that is responsible for deciding when to connect to another edge server. This philosophy is similar to 5G where the radio access network part of the 5G infrastructure and the device works together to monitor network conditions. In 5G the device measures the signal quality of its current connection and nearby 5G cells (antennas). These measurements include signal strength, signal quality, and other parameters like latency and interference.

In COGNIT for the device to make the switch from one edge platform to another without violating the RT SLA, it needs to know how long it will take to redeploy the anomaly detection function on a new edge cluster. This is the responsibility of COGNIT. With this information the device knows when to request the creation of a new Serverless Runtime and anomaly detection function on the new edge cluster so that there is no interruption in anomaly detection and no violation of RT SLA. This places the following requirements on the COGNIT Framework:

- It should be able to predict SLA RT violation.
- It should be able to identify the Edge Cluster with the right type of resources and their availability in time.
- It should be able to inform the device of the time needed to provision Serverless Runtime on the Edge Cluster and start the anomaly detection function.

These requirements can be supported by the AI-Enabled Orchestrator and a generic location based RT prediction function for devices that move and need to reconnect to another Edge Cluster with better RT.

The sequence diagram in Figure 6.5 below shows the interactions between the Device Client and the Edge Cluster Frontend:

- The Device Client will store its location data in the DaaS, when the service will be available in the future. In the security case study the data is the location of the rover platoon.
- The AI-Enabled Orchestrator continuously reads the location of the mobile device (rover platoon).
- The AI-Enabled Orchestrator uses an AI-based generic location based RT prediction function to determine if it needs to start planning preparation of the Serverless Runtime for the next edge cluster. In the above sequence diagram, it produces a warning that the RT will soon be violated.
- The AI-Enabled Orchestrator verifies that the Edge Cluster has the right types of resources and enough capacity to deploy a Serverless Runtime for the anomaly detection function.
- Since the required resources are available, the AI-Enabled Orchestrator reserves them, deploys the Serverless Runtime along with the anomaly detection function image via the Edge Cluster Frontend.

- The AI-Enabled Orchestrator then informs the device, via the COGNIT Frontend, that a switch to a new edge cluster is required to avoid violating the RT SLA.
- The Device then makes the switch to a new Edge Cluster for the next call of the anomaly detection function.

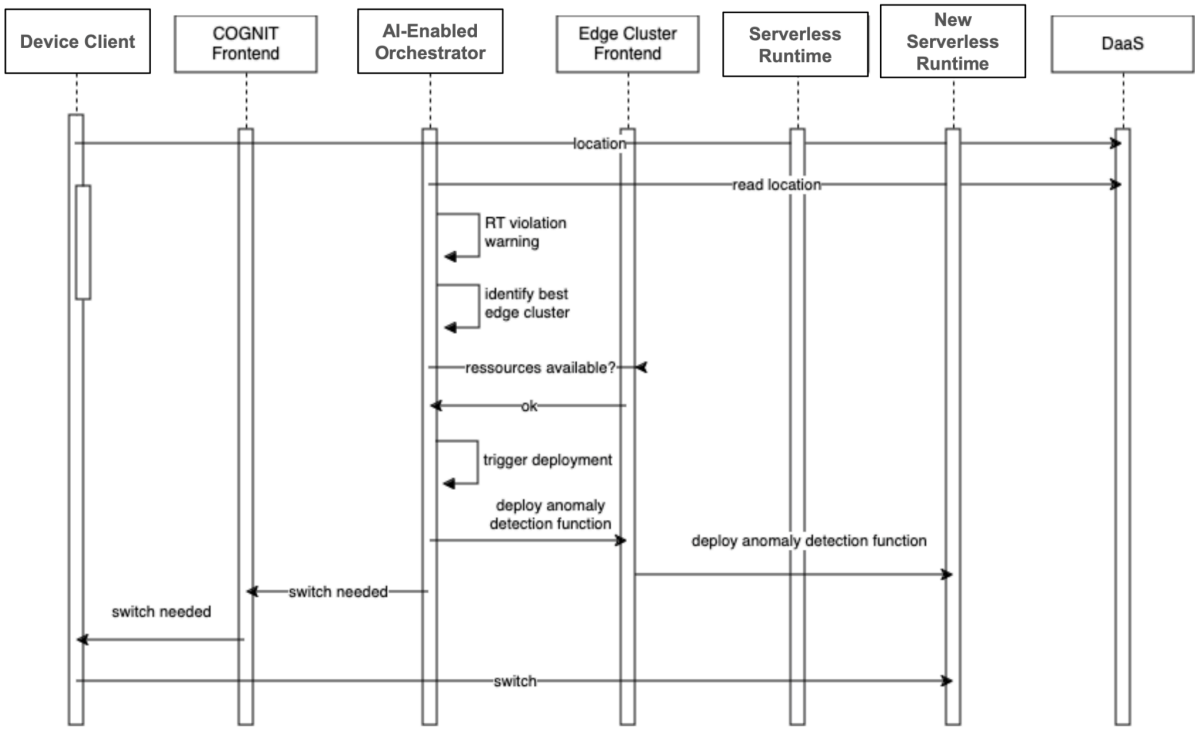

Figure 6.5. Interactions between the edge device and COGNIT Framework for switching Edge Clusters.

As described in the sequence diagram, the edge device, i.e. the rover platoon 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 monitors the location of the mobile device, i.e. rover platoon, and invokes an AI based generic location based RT prediction function to determine when 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. When it is ready, the AI-Enabled Orchestrator informs the device via the COGNIT Frontend that it can switch to another edge cluster for the next execution of the anomaly detection function. The device can then switch to the new edge cluster for the next execution of the anomaly detection function.

Validation Scenario

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. **Creating and deploying the Serverless Runtime:**
   After initialization, the rover requests the creation 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 location and using the appropriate Serverless Runtime flavour of UC4, that contains all the dependencies necessary for the anomaly detection to work. (Flavour as defined in Deliverable D3.1.)
3. **Executing the anomaly detection function with collected data:**
   The anomaly detection function is executed by processing the authentication data. The last block of lines written in the auth.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 function. Each new block of logs triggers the re-execution of the function. COGNIT must ensure that these data flows are processed in near real-time, while maintaining minimal latency between data collection and function execution.
5. **Performance measurement and response time comparison:**
   Throughout the process, we measure execution speed, data transmission times, and response times. A performance comparison will be made between using the serverless environment via COGNIT and using local anomaly detection (without using COGNIT and the edge).
6. **Edge node switching:**
   As the rover moves, and its latency deteriorates beyond the SLA, COGNIT should orchestrate a switch to another edge node to maintain low-latency processing and respect the SLA.

To validate these requirements, we will simulate the rover's operations, 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, will be measured to ensure that the COGNIT capabilities meet the requirements of the Use Case.

Use Case Testbed

To validate our use case, we deployed a complete stack of the COGNIT Framework in a virtualized environment hosted at CETIC. This setup includes the following hardware configuration:

- 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.

The COGNIT Framework operates across four virtual machines (VMs), each with:

- 2x CPUs.
- 4GB RAM.
- 100GB disk space.

In this stage, resource allocation on the VMs has been limited to prioritise integration and deployment testing. Moving forward, we plan to migrate to a more powerful setup to assess performance in conditions that closely replicate real-world deployments.

The rover itself has been virtualized and deployed on a separate hypervisor, with resources aligned to mimic the specifications of a Raspberry Pi, with:

- 4 CPU cores.
- 2GB RAM.
- 32GB disk space.

This environment also includes a simulation of necessary components, replicating interactions between an edge node and a connected vehicle. After configuring and testing the testbed to ensure it mirrors real-world conditions, including the addition of a node, network latency, and transitions between edge nodes, we will deploy an Edge cluster on-site at CETIC and connect it to the main COGNIT testbed.

PART II. Software Integration and Verification

7. Software Integration Process and Infrastructure

In this development cycle the OpsForge[8] tool has been updated to automatically deploy the new version of the COGNIT software stack, which is aligned with the new architecture defined in this cycle. This matches the second version of the COGNIT Software Stack, labelled with *release-cognit-2.0*. Several accommodations were made in the tool to: a) add Biscuit support for the OpenNebula front-end, b) allow for cross-site live migration of Serverless Runtimes and c) build images of the Serverless Runtimes leveraging the KIWI NG[9] project.

7.1 OpenNebula Biscuit Auth Extension

The Biscuit authorization driver[10] has been developed for OpenNebula in this development cycle. This driver enables the use of biscuit tokens[11] as a replacement of a user password when issuing API Calls. Each user can generate tokens with their own private key, and register their public key as their password when creating the user. This public key will authenticate the tokens signed by the private key. This contribution has not been added upstream, and therefore it needs to be added on top of a stock OpenNebula installation.

OpsForge has been extended to automatically deploy and also configure this new OpenNebula auth driver when the COGNIT stack is deployed.

7.2 Cross-Site Live Migration Configuration

A multi-edge cluster private configuration is a network topology that enables Serverless Runtime instances to run spread across multiple edge clusters. Each cluster is a group of KVM nodes and can be geographically separated from other edge clusters.

The Serverless Runtimes Virtual Machines will run using an overlay network using the OpenNebula VXLAN driver on EVPN mode. This overlay network creates an L2 network for VMs to run. Serverless Runtimes will be able to reach each other regardless of the KVM node where they are running. Serverless Runtimes will also be able to live-migrate from one KVM node to another while keeping the same IP address on the L2 overlay network.

This configuration is automatically performed in the OpenNebula frontend, rendering a configuration as described in Figure 7.1. The OneGate endpoint running in the frontend is reverse proxied through an Ingress instance (implemented by a Virtual Router VM), which is automatically managed by the Cloud/Edge Manager and made available in any Edge Cluster to be used to run Serverless Runtimes. This endpoint needs to be reachable by the

[8] https://github.com/SovereignEdgeEU-COGNIT/cognit-ops-forge
[9] https://documentation.suse.com/appliance/kiwi-9/single-html/kiwi/index.html
[10] https://github.com/SovereignEdgeEU-COGNIT/opennebula-extensions/tree/main/biscuit
[11] https://doc.biscuitsec.org/getting-started/introduction

Virtual Router (VR) in order to perform SDNAT. The EVPN VXLAN virtual network is created with the proper configuration required by the VXLAN driver in EVPN mode.

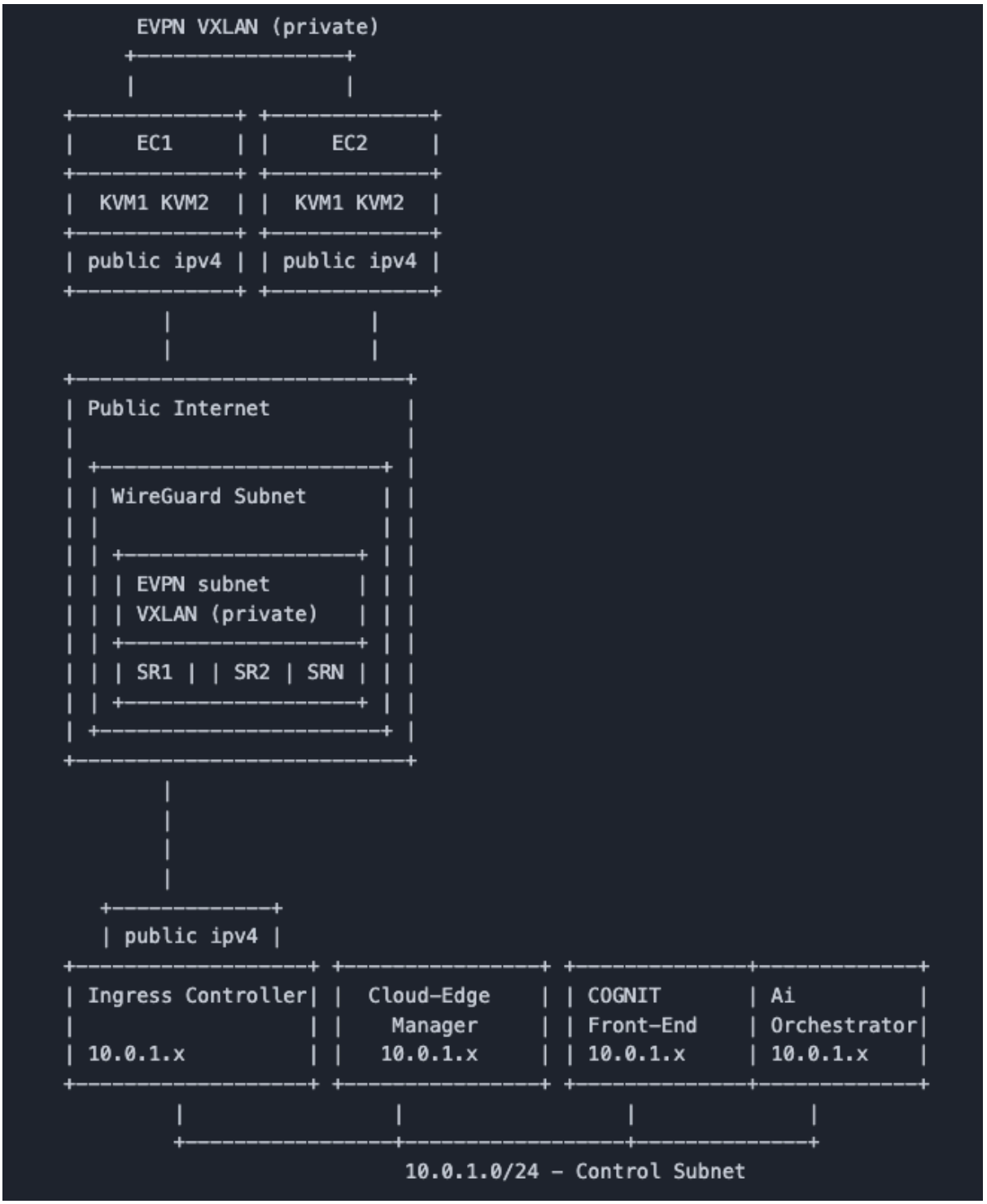

Figure 7.1 OpsForge Network Topology Deployment

Edge clusters need to be created with the OpenNebula OneProvision[12] tool. Due to this requirement, OpsForge automatically instals all the OneProvision dependencies in the

[12] https://docs.opennebula.io/6.10/provision_clusters/edge_clusters/overview.html

OpenNebula frontend instance deployment. These include tools like Ansible, Terraform and Python libraries that are not installed as part of the standard package dependencies in the frontend installation.

7.3 OpsForge KIWI Integration

OpenSUSE KIWI is a project oriented to Appliance building. An appliance is a ready-to-use image of an operating system including a pre-configured application for a specific use case. The appliance is provided as an image file and needs to be deployed to, or activated in the target system or service.

OpsForge has been integrated with KIWI in order to build the Serverless Runtimes for the different Use Case flavours, treating a Serverless Runtime as a KIWI Appliance. A new separate workflow exists to build the Serverless Runtime appliance. In order to trigger the build process, the following command must be run:

```
Unset
./opsforge build_sr <host>
```

where <host> needs to contain the hostname/IP address of a SUSE server with the needed dependencies to build the Serverless Runtime appliance and enough space available to bootstrap a new image (about 10 GB is typically enough). The needed Ansible playbooks to coordinate the KIWI processes are included in the OpsForge repository.[13]

The result of running the above command will be a *qcow2* image containing a vanilla OpenSUSE Guest OS with the Serverless Runtime software[14] installed. The output of the process would look as follows:

```
Unset
./opsforge build_sr 172.20.0.5


PLAY [Setup kiwi builder]
*****************************************************


TASK [Gathering Facts]
*********************************************************
[WARNING]: Platform linux on host kiwi1 is using the discovered
Python
```

[13] https://github.com/SovereignEdgeEU-COGNIT/cognit-ops-forge/tree/main/ansible/playbooks/roles/kiwi
[14] https://github.com/SovereignEdgeEU-COGNIT/serverless-runtime

```
interpreter at /usr/bin/python3.6, but future installation of another
Python

interpreter could change the meaning of that path. See

https://docs.ansible.com/ansible-

core/2.16/reference_appendices/interpreter_discovery.html for more
information.

ok: [kiwi1]


TASK [kiwi : Verify openSUSE distribution]
***********************************

skipping: [kiwi1]


TASK [kiwi : Install required packages]
****************************************

ok: [kiwi1]


TASK [kiwi : Copy kiwi image description files]
*******************************

ok: [kiwi1]


TASK [kiwi : Check if output directory exists and is not empty]
****************

ok: [kiwi1]


TASK [kiwi : Clean up output directory]
****************************************

changed: [kiwi1]


TASK [kiwi : Create empty output directory]
***********************************

changed: [kiwi1]


TASK [kiwi : Build kiwi image]
************************************************

changed: [kiwi1]
```

```
TASK [kiwi : Convert raw image to qcow2]
************************************

changed: [kiwi1]


PLAY RECAP
*********************************************************************

kiwi1                      : ok=8    changed=4    unreachable=0
failed=0    skipped=1    rescued=0    ignored=0
```

The above process generates an image in the host *'172.20.0.5'*, with the path to the appliance being *'/root/kiwi-image/output/cognit-sr.x86_64-1.0.0.qcow2'*.

The objective for the next development cycle is to add the needed KIWI resources to build the 4 different Serverless Runtime flavours for the Use Cases.

7.4 Second Version of the COGNIT Software Stack

The OpsForge component has been updated to automatically deploy the 2.0 version of the COGNIT software stack, which includes the components of the revised architecture presented in "D2.4 COGNIT Framework - Architecture - d". The list of components deployed by OpsForge that constitutes the 2.0 version of the COGNIT software stack, can be found in Table 7.1. This can be deployed in AWS or on premises, in the same way as described in "D5.3 Use Cases - Scientific Report - c".

Name	Documentation	Testing	Installation
Device Client (Python)	Wiki documentation	Test folder	README
Device Client (C)	N/A	Test folder	README
COGNIT Frontend	User guide	Test folder	Install guide
Edge Cluster Frontend	User guide	Test folder	Install guide
Cloud-Edge Manager	Official doc	Q&A	Install guide
Serverless Runtime	Wiki documentation	Test folder	README
AI-Enabled Orchestrator	User guide	See docs	Install guide

Table 7.1. Main COGNIT 2.0 Framework components

The main differences in the list of components with respect to the 1.0 COGNIT stack components are:

- **New** component **COGNIT Frontend**, which is the new entry point to the COGNIT stack functionality.
- **New** component **Edge Cluster Frontend,** which acts as proxy and load balancer for function execution requests to the different active Serverless Runtimes.
- **Deprecated** component **Provisioning Engine,** this previously developed component doesn't serve a purpose in the new architecture, since there is no need to facilitate the creation of new SRs to the Device Client, but rather this action is now initiated by the AI-Enabled Orchestrator, which consumes this functionality directly from the Cloud Edge Manager.

At the end of this third development cycle, OpsForge fully automates the deployment of the following components:

- OpenNebula as the Cloud-Edge Manager. Extensions made to OpenNebula in the project context that are not contributed upstream, and which can be found in the opennebula-extensions repository[15] are applied automatically. The OpenNebula services are deployed in a single VM.
- COGNIT Frontend deployed in a dedicated server or Virtual Machine.
- AI-Enabled Orchestrator, deployed on its own dedicated server or Virtual Machine. In the V1.0 of the architecture this was completely automated, but due to the change in architecture, most of the AI-Enabled Orchestrator deployment and configuration is still a manual step.
- Serverless Runtime image, using a built-in function *build_sr* (described in Section 7.3 of this document).

After the initial deployment, the OpenNebula *OneProvision* tool can be used to add computing resources to the COGNIT infrastructure, to add processing capacity in the desired geographic locations using remote OpenNebula clusters. After the computing resources are deployed in these remote OpenNebula clusters, an Edge Cluster Frontend needs to be deployed manually to proxy requests from Device Clients to these clusters. Lastly, the Device Client software can be used to offload application-specific functions to the COGNIT Platform, serving a FaaS paradigm in the cloud edge continuum.

The 2.0 version of the COGNIT stack can be deployed using OpsForge with an input YAML file such as the following:

```
Unset
:infra:
  :hosts:
    :ingress: 172.20.0.1
    :cloud: 172.20.0.4
    :frontend: 172.20.0.9
```

[15] https://github.com/SovereignEdgeEU-COGNIT/opennebula-extensions

```
    :ai_orchestrator: 172.20.17
:cognit:
  :app:
    :base: http://app_server.cognit/base_app # Replace with publicly
available app
  :certificate:
    :crt: '~/certificate.crt'
    :key: '~/certificate.key'
  :frontend:
    :version: release-cognit-2.0
  :ai_orchestrator:
    :version: release-cognit-2.0
  :cloud:
    :version: 6.10
    :ee_token: <enterprise_edition token for Cloud Edge Manager>
    :extensions:
      :version: main
```

8. Testbed Environment

This section is a brief summary of the changes made to the main testbed environment compared to what was documented in "D5.3 Use Cases - Scientific Report, section 8: Testbed Environment". All implementation and setup details of the testbed infrastructure are documented and available to all project partners in a private repository on GitHub.[16]

8.1 Central testbed setup

There have been no major updates to the central testbed setup at RISE's ICE Datacenter in Luleå (Sweden) during this cycle. For updates to the use-case-specific testbeds, we refer the reader to the corresponding Use Case sections in this document (Sections 3 to 6).

8.2 Upgraded hardware

NVMe PCI disks have been installed to support both compute hosts in the **ice** cluster (*p02r11srv01* and *p02r11srv15*). This decreases the instantiation time of FaaS Serverless Runtimes (VMs) by an order of magnitude.

8.3 Network adaptations to support the new architecture

The new architecture does not require public IP addresses assigned to instantiated Serverless Runtimes, which removes the dependency on IPv6 altogether. Because the new architecture now supports both IPv4 and IPv6, we decided to switch to using IPv4 in the testbed. The configured IPv6 networks will still be available for IPv6 specific testing, so this configuration setup is not removed. However, private IPv4 subnets have been created on edge sites from which IP addresses are assigned to FaaS Serverless Runtimes.

8.4 New components deployed

The new architecture includes two new components: the COGNIT Frontend, and the COGNIT Edge Cluster Frontend. Furthermore, the Provisioning Engine component is now deprecated.

COGNIT Frontend

This is a central component running on the ICE Datacenter. The service is hosted by the same VM that previously hosted the Provisioning Engine service. A new API endpoint has been added to load balancer:

- https://cognit-lab-frontend.sovereignedge.eu

[16] https://github.com/SovereignEdgeEU-COGNIT/infrastructure

Edge Cluster Frontend

Every edge site is fronted by one of these instances. It acts as a proxy to Serverless Runtimes running on the private network. On the ICE Datacenter site the COGNIT Edge Cluster Frontend VM is running alongside the dynamic Serverless Runtimes with two interfaces, one with a public IP and one on the private network.

9. Software Requirements Verification

Possible status are: NOT STARTED | IN PROGRESS | COMPLETED

9.1 Device Client

SR1.1 Interface with COGNIT Frontend
Status: IN PROGRESS
Description: Implementation of the communication of the Device client with the COGNIT Frontend. Following the instructions[17] 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.[18]
Completed Verification Scenarios: • [VS1.1.1] The Device Client is able to get authorization 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. **Pending Verification Scenarios:** • [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: IN PROGRESS
Description: Implementation of the communication of Device client with the Edge Cluster.

[17] https://github.com/SovereignEdgeEU-COGNIT/device-runtime-py/blob/main/README.md
[18] https://github.com/SovereignEdgeEU-COGNIT/device-runtime-py/tree/main/cognit/test

Completed Verification Scenarios:

- [VS1.2.1] The Device Client is able to execute functions (either preloaded or uploading it at the moment of execution) on the assigned Edge Cluster.

Pending Verification Scenarios:

- [VS1.2.2] The device Client is able to handle (upload/read) data privately in the assigned Edge Cluster.

SR1.3 Programming languages

Status: IN PROGRESS

Description: Support for different programming languages.

Completed Verification Scenarios:

- VS[1.3.2] Test previously described validation scenarios implemented in Python language.

Pending Verification Scenarios:

- VS[1.3.1] Test previously described validation scenarios implemented in C language.

SR1.4 Low memory footprint for constrained devices

Status: NOT STARTED

Pending Verification Scenarios:

- [VS1.4.1] Test validation scenarios described above on a device with less than 500kB of RAM.

SR1.5 Security

Status: IN PROGRESS

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.

Pending Verification Scenarios:

- [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: NOT STARTED

Pending Verification Scenarios:

- [VS1.6.1] The Device Client is able to measure latencies to different Edge Clusters concurrently with other activities of the Client, to be able to monitor and make COGNIT aware of the effective latency of the potential Edge Clusters to be used.

9.2 COGNIT Frontend

SR2.1 COGNIT Frontend

Status: IN PROGRESS

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.

Pending Verification Scenarios:

- [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: IN PROGRESS

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.

Pending Verification Scenarios:

- [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: NOT STARTED

Description: Implement a backend to persist information about the available providers that can be used to extend the capacity of the COGNIT infrastructure.

Pending Verification Scenarios:

- [VS4.1.1] Listing the providers belonging to the Provider Catalog.
- [VS4.1.2] Filtering the providers according to a desired latency threshold on a geographic area.
- [VS4.1.3] Filtering the providers according to a cost per hour threshold.
- [VS4.1.4] Filtering the providers according to energy consumption per hour threshold.
- [VS4.1.5] Filtering the providers according to some specific hardware characteristics (e.g. GPUs, Trusted Execution Environments).

SR4.2 Edge Cluster Provisioning

Status: NOT STARTED

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.

Pending 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: IN PROGRESS

Description: Edge-Clusters monitoring, Serverless Runtimes metrics collection and continuous security assessment.

Pending 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: IN PROGRESS

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.

Pending Verification Scenarios:

- [VS4.5.2] Assign ACLs to designated users and test the creation of new Edge Clusters and Serverless Runtimes.
- [VS4.5.3] Communicate with the COGNIT Frontend and the Edge Cluster Frontend using tokens.

SR4.6 Plan Executor

Status: NOT STARTED

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.

Pending 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 creating a new Edge Cluster and verify that the Edge Cluster is created correctly.
- [VS4.6.3] Submit a plan to the Plan Executor for resizing and migrating a Serverless Runtime.
- [VS4.6.4] Submit a plan to the Plan Executor for increasing the number of hosts (horizontal scaling) of an existing Edge Cluster.

9.5 AI-Enabled Orchestrator

SR5.1 Building Learning Models

Status: IN PROGRESS

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.

Pending Verification Scenarios:

- [VS5.1.3] 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: IN PROGRESS

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.

Pending Verification Scenarios:

- [VS5.2.1] Assess the ability of workload and resource optimization in terms of cost and performance trade-off.

9.6 Secure and Trusted Execution of Computing Environments

SR6.1 Advanced Access Control

Status: IN PROGRESS

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: IN PROGRESS

Description: Enable privacy protection for the application workloads at the hardware level using Confidential Computing (CC) techniques.

Pending 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. Conclusions and Next Steps

On the basis of the first three versions of the Use Cases Scientific Report (Deliverables D5.1, D5.2 and D5.3), this fourth version provides an overview of the overall status of the integration of the use cases with the COGNIT Framework. It provides further details about the research and technology developments that have been achieved by the use cases in the Third Research & Innovation Cycle (M16-M21). Finally, this report provides an update on the integration process and infrastructure for the updated version of the integrated COGNIT software stack, and on the progress of the software requirement verification tasks per component.

This report complements the Project's global overview provided by Deliverable D2.4, as well as the component-specific research and development activities reported in Deliverables D3.3, D3.8, D4.3, and D4.8.

Two more incremental versions of this report will be released at the end of the two remaining research and innovation cycles (i.e. M27 and M33).