871780 — MonB5G — ICT-20-2019-2020

Deliverable D4.3
Report on Integration and testing of the MonB5G Decision Engine

Document Summary Information

Grant Agreement No	871780	**Acronym**	MonB5G
Full Title	Distributed Management of Network Slices in beyond 5G		
Start Date	01/11/2019	**Duration**	42 months
Project URL	https://www.monb5g.eu/		
Deliverable	D4.3 – Report on Integration and testing of the MonB5G		
Work Package	WP4		
Contractual due date	M37	**Actual submission date**	07/12/2022
Nature	Report	**Dissemination Level**	Public
Lead Beneficiary	IQU		
Responsible Author	Luis A. Garrido (IQU), Kostas Ramantas (IQU)		
Contributions from	Kostas Ramantas (IQU), Luis A. Garrido (IQU), Anestis Dalgkitsis (IQU), Anne-Marie Bosneag (LMI), Ashima Chawla (LMI), Lanfranco Zanzi (NEC), Francesco Devoti (NEC), Farhad Rezazadeh (CTTC), L. Blanco (CTTC), E. Zeydan (CTTC), L.		

	Vettori (CTTC), J. Mangues (CTTC), S.Kahvazadeh (CTTC), Robert Kołakowski (ORA-PL), Rafał Tępiński (ORA-PL), Pavlos Doanis(EUR), Sihem Cherrared (ORA-FR)

Revision history

Version	Issue Date	% Complete	Changes	Contributor(s)

Disclaimer

Copyright message

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

Acronym	*Description*
3GPP	*Third Generation Partnership Project*
5GC	5G Core
A3C	Asynchronous Advantage Actor Critic
AE	Analytic Engine
AI	Artificial Intelligence
API	Application Programming Interface
BS	Base Station
CL	Closed Loop
CLC	Control Loop Coordination
COMS	Common Storage
CSS	Cascading Style Sheets
CT	Control Trigger
DA	Decision Agent
DB	Database
DDQN	Duelling Deep Q-Network
DE	Decision Engine
DMA	Decision-Making Algorithm
DMO	Domain Manager and Orchestrator
DNN	Deep Neural Networks
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
eMBB	Enhanced Mobiled Broadband
eNB	eNodeB
EPC	Enhanced Packet Core
ETSI	European Telecommunications Standards Institute
ECATP	Enhanced Context-Aware Traffic Predictor

ENI	Experiential Networked Intelligence
EUI	External User Interface
FDRL	Federated Deep Reinforcement Learning
GCN	Graph Convolutional Networks
gNB	gNodeB
HA-DRL	Heuristically-Assisted DRL
HTML	Hypertext Markup Language
IDMO	Inter-Domain Manager and Orchestrator
IDQN	Independent Deep Q Network
KPI	Key Performance Indicator
LCM	Lifecycle Management
LTE	Long Term Evolution
LTE-M	TE Machine Type Communication
LXD	Linux Container Hypervisor
MDP	Markov Decision Process
ML	Machine Learning
MS	Monitoring System
NB-IoT	Narrowband Internet of Thing
NBI	North-bound Interface
NFV	Network Function Virtualization
NR	New Radio
NSA	Non-Standalone
NSP	Network Slice Placement
NSPR	Network Slice Placement Request
OPEX	Operational Expenditure
P2C	Power of two choice
PNF	Physical Network Function
PRB	Physical Resource Block
PreBAC	Prediction-based Admission Control
PSN	Physical Substrate Network

QL	Q-Learning
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
Rpi	Raspberry Pi
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SBI	South-bound Interface
SCHE2MA	Service CHain Energy-Efficient MAnagement
SLA	Service Level Agreement
SDK	Software Development Kit
SDN	Software-Defined Network
SFC	Service Function Chain
SFL	Slice Functional Layer
SFTP	Secure File Transfer Protocol
SML	Slice Management Layer
SM	Slice Manager
SSH	Secure Shell
TASAC	Time-of-day Aware Slice Admission Control
TSDB	Time-series Database
UE	User Equipment
uRLLC	Ultra-Reliable Low-Latency Communication
USR	User Service Request
VM	Virtual Machine
VNF	Virtual network Function

1 Executive Summary

The scope of this deliverable focuses on the way the Decision Engine (DE) is integrated and interacts with the rest of the components of the MonB5G project, namely the Analytics Engine (AE) and the Monitoring System (MS). This deliverable differentiates itself from D4.2 [1] in which the latter focuses on "the devised DE algorithms" for which detailed descriptions of the problem domains they target (i.e. admission control, intra-slice orchestration, among others) are provided, as well as formulations for the contributed solutions. With these contributions in hand, D4.3 presented here analyses the integration of the DE with the rest of the MonB5G components (AE and MS), its integration with controlled systems and implementation results when available. The emphasis of this deliverable is on the appropriate functional testing, validation of interoperability of the MonB5G components with the DE, as well as the used data models (when necessary) and any other additional components and functionalities added to ensure its correct deployment.

The first part of this deliverable (Section 2) starts with a review of the generic architecture of the DE and the overall MonB5G architecture in order to provide enough contextual information for the subsequent sections. This section will also review the details of the different components and interfaces involved among the communication of its components, so that it is feasible to understand the corresponding integrations of the DE.

The second part of this document (Sections 3) details the DE algorithms that have been integrated and have had functional validation when integrated with other MonB5G components. Each of the corresponding sub-sections in this part includes:

- Brief description of the Artificial Intelligence (AI) or Machine Learning (ML) Algorithm of the corresponding solution (already explored in more detail in D4.2 and previous MonB5G deliverables),
- Description of the messaging interfaces used by the DE implementation to communicate with other (MonB5G) components,
- Description of the integration APIs, which refers to the application level details that allow for the messages interfaces to be implemented
- Description on the Development Frameworks used to develop and implement the DE, its APIs and ease its integration with other components, and
- Key Performance Indicator (KPI) mappings of the DE to the performance objectives set by the MonB5G project.

The final part of this deliverable (Section 4) will provide insight on the final deployment of the DE and the rest of the MonB5G components on an experimental platform, with a very brief description on the immediate work and a very rough timeline for the completion of this deployment. This last section works as a prelude to the technical reports on system integration and operation of the MonB5G components.

2 Overall DE Architecture and MonB5G Component Integration

The generic architecture of the DE was first introduced in [2], and has been improved since then in order to provide an updated version in the current deliverable. The DE can be described as being composed of several internal sub-blocks:

- The Input/output Pre-processors (presented already in [2]),
- the Control Trigger (CT) [2], and
- the Decision Algorithm (DA) [2]
- the Input/Output Interfaces, formally introduced in the current version.

The improvements have been added in order to make it compliant with the ETSI-ENI standard [3]. The generic architecture of the DE and its internal sub-blocks are shown in Figure 1. The specific implementation of each of these components is mutually dependent, being defined by the particularities of the problem domain targeted by the Decision-Making Algorithm (DMA) and the underlying controlled system. The DMA can be an algorithm of any type (i.e. linear programming, fuzzy logic, etc), including ML algorithms, or any type of combinatorial or optimization solution.

.

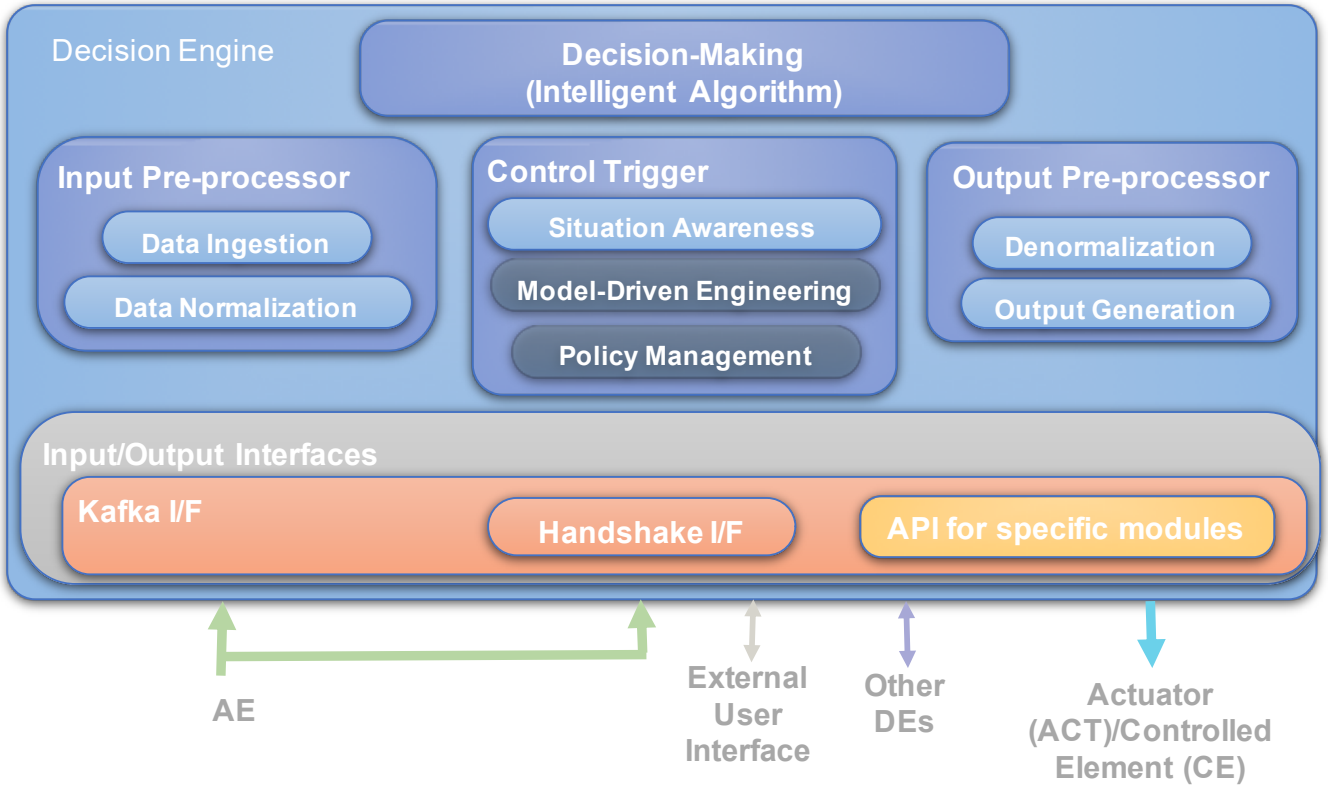

Figure 1. Generic Architecture of the DE.

As mentioned in [2], the Input Pre-processor will sample multiple metrics from the controlled system (arriving via de MS/AE) and re-format and normalize them into a bounded domain and based on this re-formatted variables, it will build an instance of the state vector that will be part of the state representation of the underlying Markov Decision Process (MDP) for which the Reinforcement Learning (RL) algorithm is deployed.

Similarly, the Output Pre-Processor will re-format the output of the DMA and de-normalize it into a domain that matches the constraint of the controlled system and the communication interfaces. As mentioned earlier, the specifics of the (de-)normalization and re-formatting for both the Input and Output Pre-Processors is dependent on the specifics of the DMA and the underlying controlled system.

The Input/Output Interfaces have been added to the current version in order to provide a single point of entry and exit from the DE, and to ease the way it interfaces with the other MonB5G components and other elements in the underlying system. Given that the MS has been implemented with a Kafka Bus Interface, it is expected that the AE/DEs communicate with each other extending the Kafka Bus, as it will be shown in Section 3. The Input/Output Interfaces also expect to provide support to enable communication across DEs and to an External User Interface (EUI) in order to allow for a human operator to interact with the DEs.

The Control Trigger (CT) was added to allow for the DE to be able to respond to operating conditions of the controlled system that would require immediate decision making. As explained in [1], the DE is the highest level of a Control Loop (CL), and the time window of this CL can vary depending on the problem domain targeted by the DMA. However, it is expected that this time window is in the order of a couple of minutes for most problem domains, since it will usually take an approximate time window of said size for the system to respond to the decisions issued by the DMA and reach the state that the DMA intends for the controlled system (i.e. for the loop to close). During this time window, a series of conditions might prompt an immediate change of behavior by the system. Examples of these situations could be a fault that triggers a system malfunction, a surge in load, a malicious attack, sudden changes to the underlying infrastructure and available physical resources, or even changes in the energy consumption for multiple reasons. The CT then offers the functionality to enable the DE to trigger actions as a response to these conditions.

It is important to mention that the implementation of some of these sub-blocks within the DE are optional, and their presence will be subjected to the particulars of the DMA and the environment in which it is operating, while other sub-blocks like the DMA is of fundamental importance, since it is the sub-block that generates the action for the system. Similarly, as it was previously mentioned, the particular implementation of each of this sub-blocks is also dependent on the specifics of the use case.

As mentioned in [1], the DE presents different levels of instantiation that can be used at the different scopes of data analytics and decision-making in order to minimize the data exchange and allow faster local analysis and decisions. This applies to the MS and the AEs as well. The different levels of instantiation are:

- Virtual Network Function (VNF)/Physical Network Function (PNF)
- Network Slice Level, where each slice template includes an MS/AE/DE
- Domain Manager and Orchestrator (DMO)-Level, in which the domains are the Radio Access Network (RAN), Edge and Cloud, in which same-level MSs/AEs/Des can perform domain-wide management

- Inter-Domain Manager and Orchestrator (IDMO)-level, which manages the lifecycle of end-to-end network slices.

When an instantiation of the MS/AE/DE triplet is deployed, they can be flexibly bounded to each other, and communication is established between them through the corresponding interfaces. The design and implementation of these communication interfaces will depend on the use case for which the triplet is deployed and the specifics of the problem domain. Figure 2 reviews the communication between the MS/AE/DE triplet, revisiting the information presented in [1].

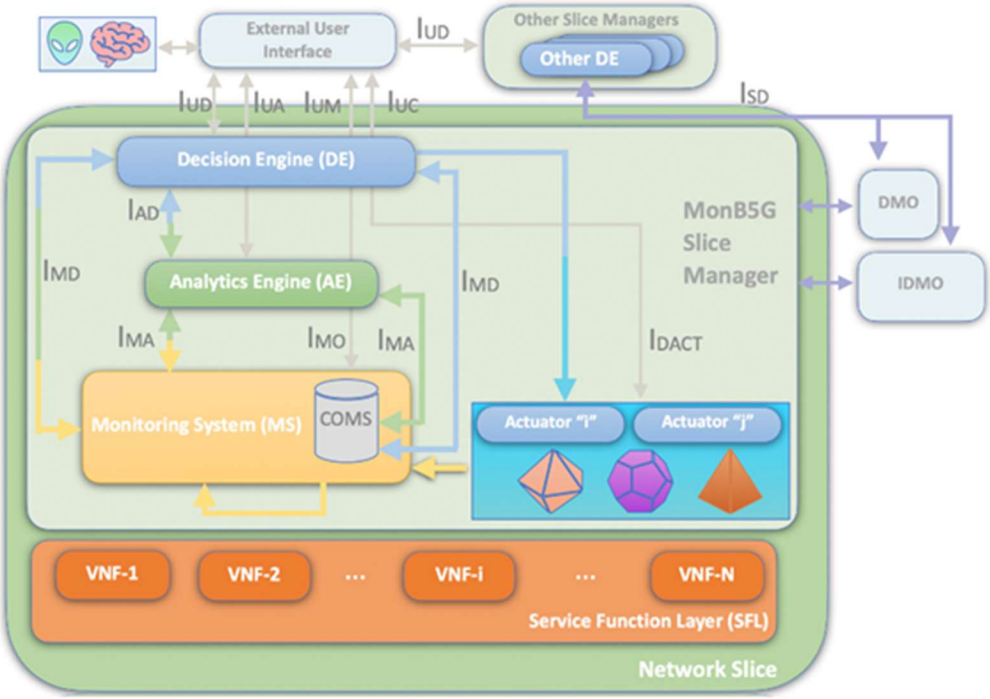

Figure 2. Interfaces between the MS/AE/DE components of the MonB5G architecture.

The DE will read the inputs to its DMA component either from the AE or from the MS. The input coming from the MS can be the current system status sampled by the MS, or from the Common Storage (COMS) component, which is included inside the MS. A more detailed description of the interfaces show in Figure 2 was given in [1], which we recap here for convenience:

- I_{AD}: DE Reads the predicted KPI from AE
- I_{MD}: DE reads MS measurements, either directly or from the COMS
- I_{MA}: AE reads MS measurements, either directly or from the COMS
- I_{UD}: EUI reads/Changes DE configuration
- I_{UA}: EUI reads/Changes AE configuration (e.g., prediction interval, learning rate)
- I_{UM}: EUI reads/changes MS configuration (e.g., granularity)

- I_{UC}: EUI reads/changes actuation configuration (e.g., API primitives' parameters)
- I_{DACT}: DE sends decisions to Actuators.

The Actuators encompass the interface to the controlled system, and it is the point of entry of the commands coming from the DE. The actuators are the ones in charge of executing the actions in the controlled system and transitioning its state. These are the lowest level of the CL in the MonB5G architecture. The Actuators will vary depending on the use case and its implementation, and the solutions provided in Section 3 will demonstrate different actuators being used with the MonB5G components.

3 DE Integration: Implementation of AI Algorithms and Corresponding Modules

3.1 Heuristically assisted DRL approach for network slice placement

In order to optimize the acceptance ratio in the case of random slice request arrivals and random holding times of resources by slices, we proposed a hybrid placement solution based on Deep Reinforcement Learning (DRL) and a dedicated optimization heuristic based on the "Power of Two Choices" principle. The DRL algorithm uses the so-called Asynchronous Advantage Actor Critic (A3C) [4] algorithm for fast learning, and Graph Convolutional Networks (GCN) [5] to automate features extraction from the physical substrate network. The proposed Heuristically-Assisted DRL (HA-DRL) enables a significant acceleration of the learning process and substantial gain in resource usage when compared against other state-of-the-art approaches, as evidenced by evaluation results. In the following, we will present the framework and technologies applied to integrate the HA-DRL solution.

3.1.1 DEVELOPMENT FRAMEWORKS

The architecture of the proposed Network Slice Placement (NSP) solution is illustrated in the Figure 3. The proposed framework is divided into three main components:

A. <u>The Analytics Engine (AE):</u>

The AE contains the Physical Substrate Network (PSN) database that stores the updated data about the available resources of the PSN. It also contains the Network Slice Placement Request (NSPR) generator that is used to generate NSPR arrivals according to a specific network load regime. It generates slice requests arrivals considering three different network load scenarios: stationary, cycle-stationary, and non-stationary network load scenario. The stationary network load scenario static while the cycle-stationary and non-stationary loads vary in time. The former varies with a predictable periodic load while the latter in a non-predictable change. The NSPR requirements and PSN available resources data are used as inputs to the DRL algorithm by the placement module.

B. <u>The Decision Engine (DE):</u>

The placement module implements the DRL-based algorithms and also the Power of two choice (P2C) heuristic algorithm referred to in the following as HEU used by HA-DRL and HA-eDRL algorithms to accelerate convergence. Both algorithms calculate:

i) a VNF placement decision, that is, where each VNF of the NSPR is to be placed and

ii) a VNF chaining decision, that is, which paths in the network to use to interconnect the different VNFs. The Placement module can be configured to use one of the Placement algorithms or both if comparison of Placement solutions is necessary.

Once the calculation of the Placement decision is done for one NSPR, an update of the available resources in the PSN is made and some key performance metrics are registered in the form of data series; the key metrics are the acceptance ratio of network slices and the resource usage.

C. Visualization modules:

The time series introduced above are used by the Data visualization component to build two dashboards: an acceptance ratio dashboard and a network load dashboard.

Both dashboards are used to show the performance of the algorithms according to variations on the network load in real time. Finally, the graph visualization component is used to allow the visualizations of the PSN and NSPR graphs.

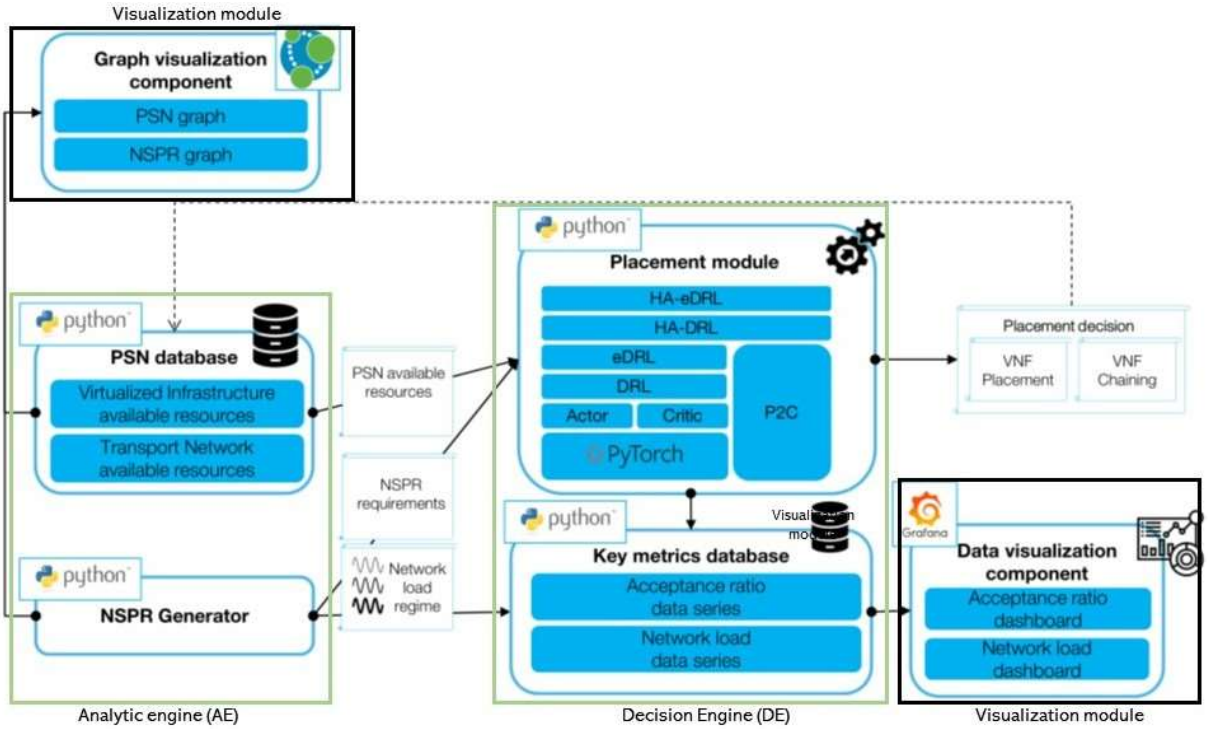

Figure 3. The DRL-HA deployment frameworks.

The technologies applied for the implementation of the different components are:

- Python: We use Python and PyTorch for implementing the different elements of the Analytic and Decision Engines.
- Neo4j: A Neo4j graph database represents and displays the PSN graph and the NSPR graph together with its requirements.
- MySQL: We use the MySQL database manager system to implement the Key metrics database with one table for the Acceptance ratio data series and another one for the Network load data series.
- Grafana: We use the Grafana tool to implement the Data visualization component in which we represent two dashboards using the MySQL database of Key metrics as data-source.

3.1.2 MESSAGING INTERFACES

To learn the model, the reinforcement learning algorithm applies a set of actions on the environment. Each action results in a reward that evaluates the new state of the environment. In the proposed framework, the learning process generates messages between the AE and DE modules. Figure 4 presents the message flow diagram between the DC and DE modules including the internal messages exchanged in each module. In the learning procedure, first the DE receives a NSPR request from the network generator with the slice description. The PSN graph is then send to the GCN module in the DE to maximize the graph learned features and used as inputs to the DRL algorithm. Each NSPR represents a set of actions to place VNFs of the slice. An action represents a NSPR VNF placement in one PSN node. If the action is successful and the path is defined between the VNF to be placed and the old placed VNFs, the DE sends an update request to the AE in order to update the PSN. One the learning is concluded; the acceptance ratio and the network load metrics are stored in the database.

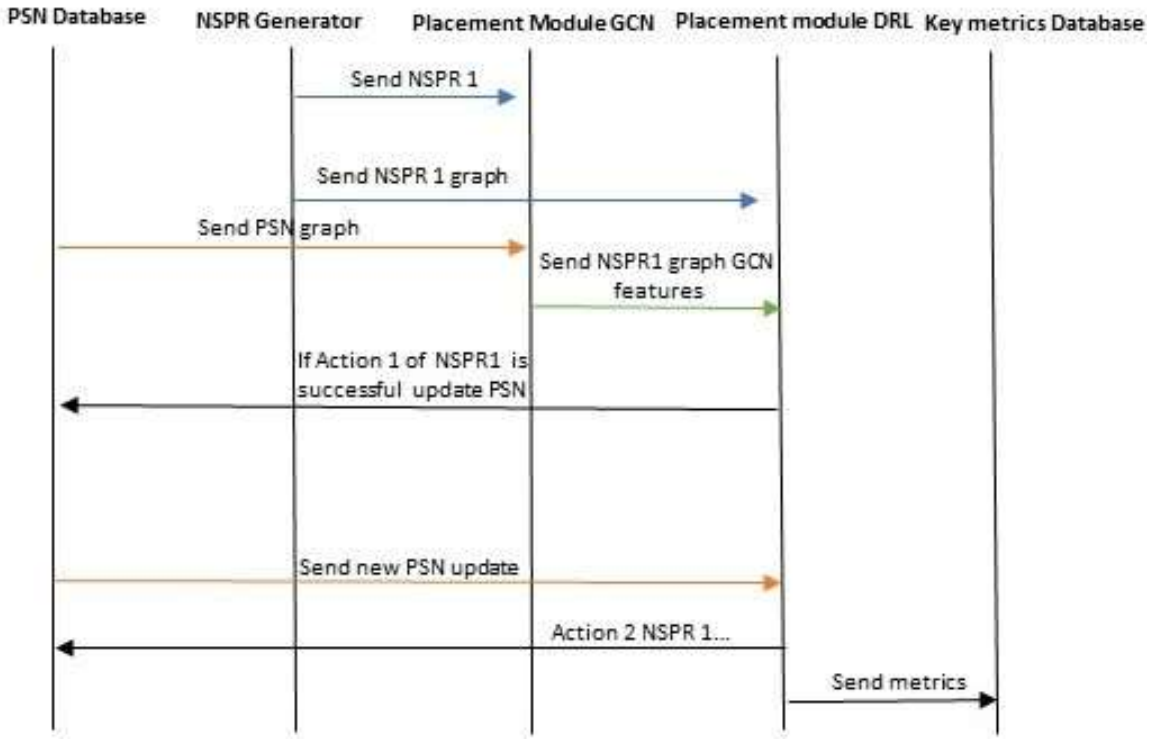

Figure 4. AE and DE messages data flow diagram.

3.1.3 INTEGRATION APIS WITH EXTERNAL ELEMENTS AND AE

In the framework, we define two APIs the internal API is the interface between the AE and DE. The external API provides the results and metrics from the DE to external modules such as the visualization modules. Table 1 showcases the different endpoints.

Endpoint	Type	method	Data
API/PSN	Internal AE	GET	Returns the description of the current PSN graph status.
API/PSN/update	Internal AE	POST	Updates the PSN with the new slice and VNFs placements.
API/NSPR	Internal AE	GET	Returns the description of the slice described in the current NSPR.
API/NSPR/next	Internal AE	GET	Returns the next received slice request to be placed
API/learn	External DE	POST	Learns a new model with the specified parameters.
API/solve	External DE	POST	Provide solutions with the last learned model.
API/metrics	External DE	GET	Returns the performance Metrics for the visualization tools.

Table 1. Internal and external APIs endpoints.

3.1.4 MAPPING TO KPIS

The proposed solution covers a number of Monb5G KPIs with regards to scalability and the AI algorithm performance. The simulation results applied to the DRL-HA solution presents a fast convergence of the algorithm, for instance 1000 episodes less training phases than the state-of-the-art baseline. The algorithm execution time is less than 1s. The algorithm was also tested under different network loads up to 90%. Figure 5 showcases the results of DRL approach in the case of the use of a heuristic (HA-DRL) and without (DRL) in two different network loads (normal 50% and critical 90%). We notice that even in an excessive network load (90%) the DRL with heuristic $\beta = 2$ can achieve high slice acceptances ratio (>90%) compared to other methods. This responds to the Monb5G KPIs that consider to maximize the NS acceptance ratio (UC1/ES1 KPI-5, UC1/ES2 KPI-3).

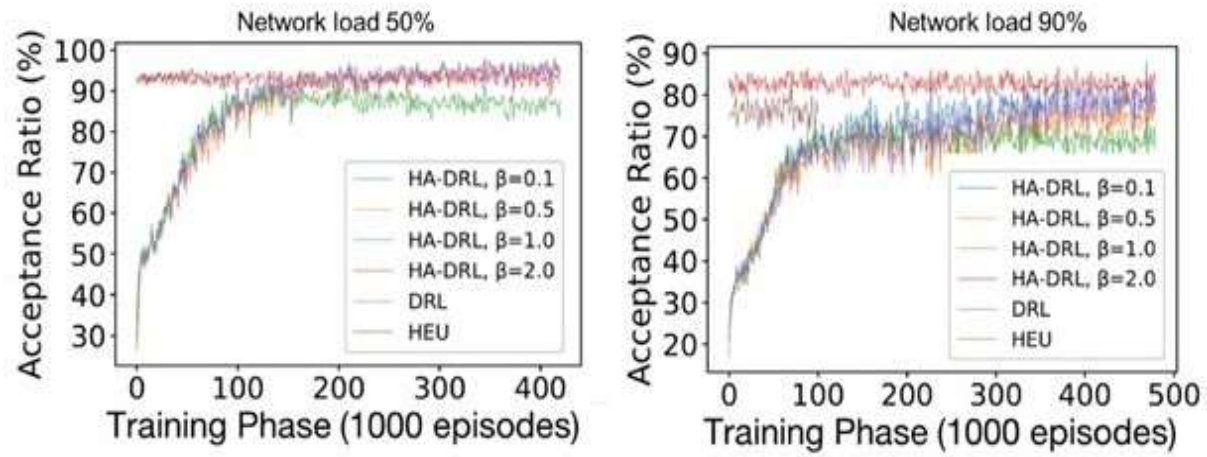

Figure 5. Evaluation Results of the DRL approach in 50% and 90% network load. (β parameter is used to control how much HEU influence the policy).

3.2 Multi-domain Slice Orchestration: SafeSCHEMA and SCHE2MA Orchestration Frameworks

SafeSCHEMA and SCHE2MA are both Service CHain Elastic MAnagement frameworks for VNF placement and migration, with the difference that SafeSCHEMA is based on Safe RL to ensure the safety of the systems and the agents upon deployment. On the other hand, SCHE2MA focuses on optimizing latency and energy jointly in a purely distributed RL framework. SafeSCHEMA and SCHE2MA share the same core infrastructure for deployment and integration, and also share a lot of the software constructs. They differ in the following two ways:

- SafeSCHEMA is designed to optimize end-to-end latency of Service Function Chains (SFCs), while SCHE2MA focuses on jointly improving latency and energy consumption for end-to-end slices.
- SafeSCHEMA employs safe RL in order to restrict the states/actions that the intelligent agents can visit in the process of learning a VNF placement policy, while SCHE2MA has no such restriction.

SafeSCHEMA is a modular architecture capable of safe and automated slice management for networks that consist of multiple interconnected domains that span multiple locations. The architecture consists of multiple distributed intelligent agents that co-operate to orchestrate the slice elements, specifically to manage the placement of the slice VNFs. The RL agents are wrapped with a Safety Shield, which prevents the execution of unsafe placement actions that can be proven dangerous for the operation of the E2E slice performance.

Previous results demonstrated improved performance over competing solutions, while ensuring the safety of the performed actions during real-time slice orchestration.

Along the same lines, SCHE2MA has been implemented as a DE using pure distributed RL that can intelligently deploy SFCs with shared VNFs per se into a multi-domain network. It is a distributed zero-touch management and orchestration algorithm that requires no intervention. It comprises of multiple distributed DEs based on DRL that orchestrate the VMs locally, limiting the computationally and energy costly frequent inter-domain migration and total slice orchestration. SCHE2MA was evaluated through model validation and simulation while demonstrating its ability to jointly reduce average service latency by 103.4% and energy consumption by 17.1%, contributing to the minimization of the Operational Expenditure (OPEX), compared to a centralized RL solution.

3.2.1 DEVELOPMENT FRAMEWORKS

The distributed intelligent agents of the SafeSCHEMA and SCHE2MA frameworks were developed on a custom-made OpenAI Gym environment using Python 3.8 and deployed using Docker containers. TensorFlow 2.4.0-rc0 [7] and with the high-level Keras 2.4.2 open-source libraries were used to build their Neural Networks. The network environment used for the emulation is a fork of Containernet, an advanced branch of Mininet network emulator used for evaluation by many works in related literature. It simulates a realistic virtual network, VM or container hosting, switching, and application code for developing and experimenting with SDN-NFV networks. The network topology used is a variation of the 2005 Nordu European network, from The Internet Topology Zoo online database, adjusted to accommodate multiple computational domains and fit the requirements of the study.

The FastAPI Python module was used for the development of a REST-ful inter-domain communications system, called the Auction Mechanism. For the case of SCHE2MA, a Graphical User Interface (GUI) was developed using Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS), and other web-based tools. This GUI is shown in Figure 6. The purpose of this GUI is to ease the monitoring of SCHE2MA's behaviour, observe the way VNF placements are occurring and in general to improve the controllability of SCHE2MA.

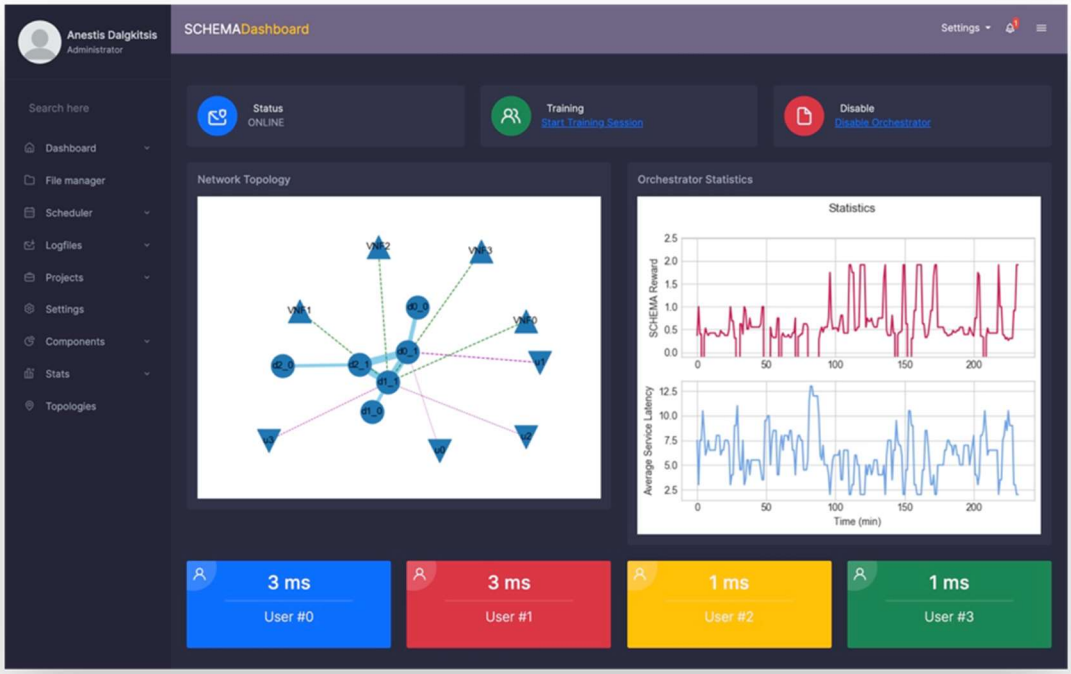

Figure 6. The GUI developed for the SCHE2MA Framework.

3.2.2 MESSAGING INTERFACES

Internally, all SafeSCHEMA and SCHE2MA modules communicate through multiple REST API calls. The interface used to transfer system state information is used to transfer network and computational state messages and post data relevant to the local decision-making of the domain agents. All messages are serialized and then stored locally in multiple SQLite3 databases.

Auction Mechanism

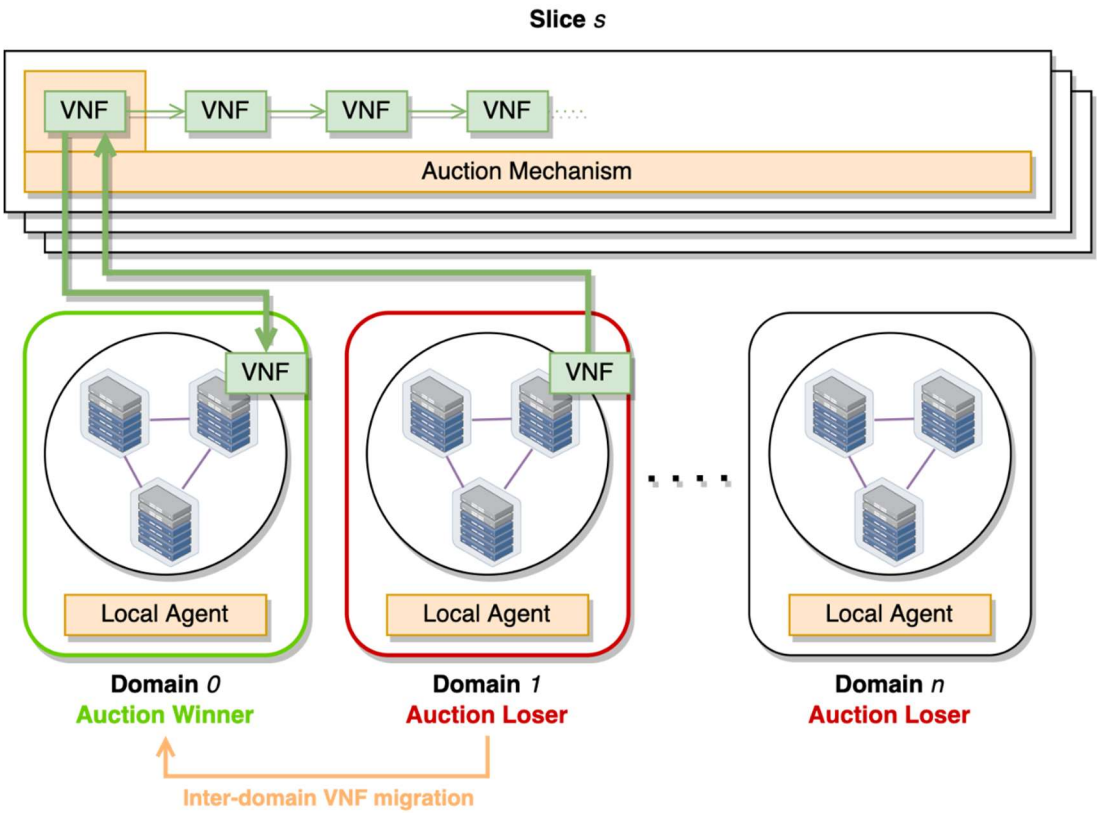

Figure 7. Overview of the SafeSCHEMA multi-domain and distributed Auction Mechanism.

The Auction Mechanism is a system that enables inter-domain VNF migration in a distributed multi-domain network. The Auction Mechanism enables scalability and parallel operation between the slice domains, as shown in Figure 7.

The operation of the Auction Mechanism can be described in the following steps:

1. **Selection:** The Auction Mechanism chooses the next service VNF and advertises to the distributed domains the requirements of the VNF placement.
2. **Participation:** The distributed Safe RL agents of the domains generate their local action or Confidence Vector to propose a local placement for the advertised VNF, ensuring minimum data transfers.
3. **Auction:** The Auction Mechanism receives the Confidence Metric of each domain and chooses the highest bidder or the domain with the maximum Confidence Metric as a candidate to receive the VNF currently in auction. The Auction Mechanism notifies the candidate domain with an acknowledgment response.
4. **Orchestration:** If the candidate domain is different from the current domain that hosts the VNF in the auction, the inter-domain migration is initiated. Contrariwise, the domain agent performs an

intra-domain migration to the node with the highest Confidence Metric of the local Confidence. If the VNF is already instantiated in the same node, the procedure of migration is declined.

3.2.3 INTEGRATION APIS

All SafeSCHEMA intelligent agents, as well as the intelligent agents of SCHE2MA and their respective modules, communicate via REST API calls. The messages are structured with JSON format. Python and the FastAPI module were used for the construction of the interfaces.

API Endpoints

The communication between the distributed agents of SafeSCHEMA and SCHE2MA and their respective Auction Mechanisms through the API are described in Table 2:

Endpoint	Method	Message
/modules/state/	POST	Network state JSON Message.
/modules/heartbeat/	GET	Acknowledgement and health status messages channel for the local Safe RL agents.
/migrate/	POST	Migration request from the Auction Mechanism to the source domain.

Table 2. API endpoints of the SafeSCHEMA agents and the Auction Mechanism.

3.2.4 MAPPING TO KPIS

The evaluation and testing of our proposed framework, show that, compared to our proposed solution, unrestricted RL agents explore unsafe regions of the state-action space during exploration, leading to the SLAs being broken. In contrast, SafeSCHEMA leads to better performance, scalability and stability thanks to the use of safe RL. Figure 8 presents the performance of the compared algorithms during the operation of multiple slices in distributed domains, hence their ability to maintain performance, demonstrating the ability to scale horizontally. The average slice latency is plotted against the number of chained VNFs on each of 4 slices. It is clear, especially in the case of 2 and 4 chained slice VNFs that SafeSCHEMA was able to maintain a lower slice latency by 126.62% from Static in the case of 8 VNFs.

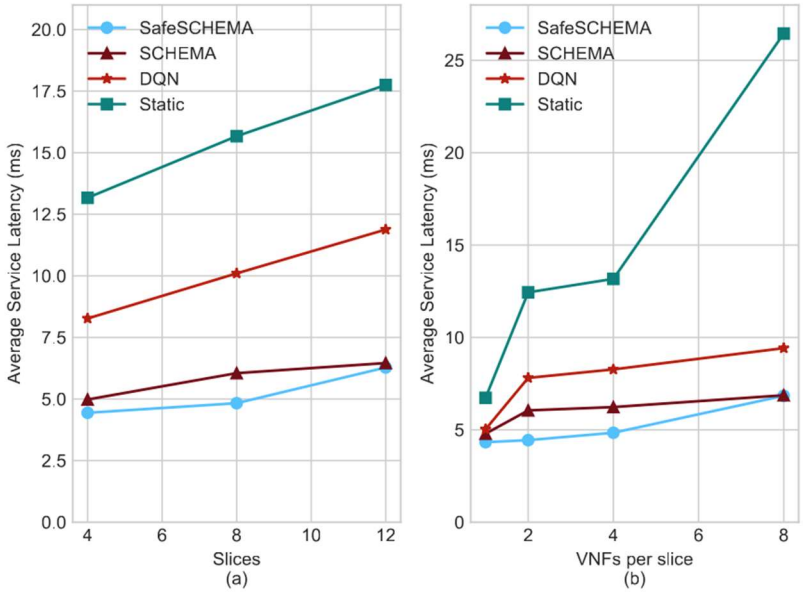

Figure 8. SafeSCHEMA evaluation results in low-latency services.

The SafeSCHEMA framework contributes and improves the following MonB5G KPIs:

- **[UC1/ES1 KPI-1]:** Reduction of SLA Violations.
- **[UC1/ES1 KPI-7]:** Improve the accuracy of the AE/DE mechanisms for detection of slice performance degradation.

In the case of the SCHE2MA framework, as previously mentioned, it is designed to minimize the end-to-end average slice latency and energy consumption via monitoring and orchestrating the slice VMs in the multiple local network domains, which leads to a reduction of the OPEX incurred by infrastructure providers. The SCHE2MA framework mainly contributes and improves the following MonB5G KPIs:

- **[UC1/ES1 KPI-5] OPEX reduction due to the automation of service management:** Approximately 20% reduction in energy consumption and 103.4% reduction in E2E service latency were validated via testing. Energy efficiency targets and service KPIs and SLAs were met during evaluation.

3.3 RL-based Slice Admission Control

This section describes the integration of another DE instance called Prediction-based Admission Control (PreBAC), which is an RL-based Admission Control solution that exploits future state prediction from an Enhanced Context Aware Traffic Predictor (ECATP) [8]. PreBAC works by admitting or rejecting (optionally

can also delay it) a User Service Request (USR) into the network slice for which it requires to get service from. The admission or rejection is done based on bandwidth resource availability at the Base Station (BS), and PreBAC dynamically allocates the available capacity among the active network slices. The admission/rejection and optional delay occurs depending on how PreBAC allocates the bandwidth resources to the slices in a specific control cycle.

3.3.1 DEVELOPMENT FRAMEWORKS

The DE implementation of PreBAC has been implemented using Python 3.8, with Tensorflow 2.7.0 [7]. The Reinforcement Learning algorithm used in its implementation is the Soft Actor-Critic (SAC) [9]. The agent runs inside a container supported by the Linux Container Hypervisor (LXD), from which it communicates with the MS and the AE running within their own respective containers. All the MonB5G components deployed for this use case are running on site with a BS with its respective gNB.

The development of the interfaces between the EUI and the DE for this instance were done using cURL and FLASK, which are both available as libraries in Python 3.8. The utilization of these two frameworks fall in line with the design and implementation provided by the MS, which basically set the blueprint for the communication interfaces.

For performance and compatibility purposes, a PreBAC Proof of Concept has been implemented, deployed and functional tested in an experimental platform at Iquadrat premises that will be used exclusively for the project that consists on the components shown in Figure 9. The performance results will be reported in the D6.1-Technical report on system integration and operation. This platform consists of two Amarisoft mini gNBs with their MEC nodes directly associated to it. The Amarisoft mini gNBs are connected to a 5G Core working in 5G Standalone (SA) mode running in a container in a LXD server. The UEs in these setting consist on two 5G OnePlust 8T phones, and a SIM8200EA-M2 5G HAT attached to a Raspberry Pi (Rpi) 3/4.

The HAT component is a 5G module that can be attached to the Rpi in order to allow connectivity of the Rpi to the Amarisoft gNB through the 5G network. The Rpi has computational capabitilities that can be used to perform some data processing, or to use it as a 5G hotspot to a remote LAN attached to the Rpi. The utilization of the HAT opens the possibility to try out a plethora of different use cases.

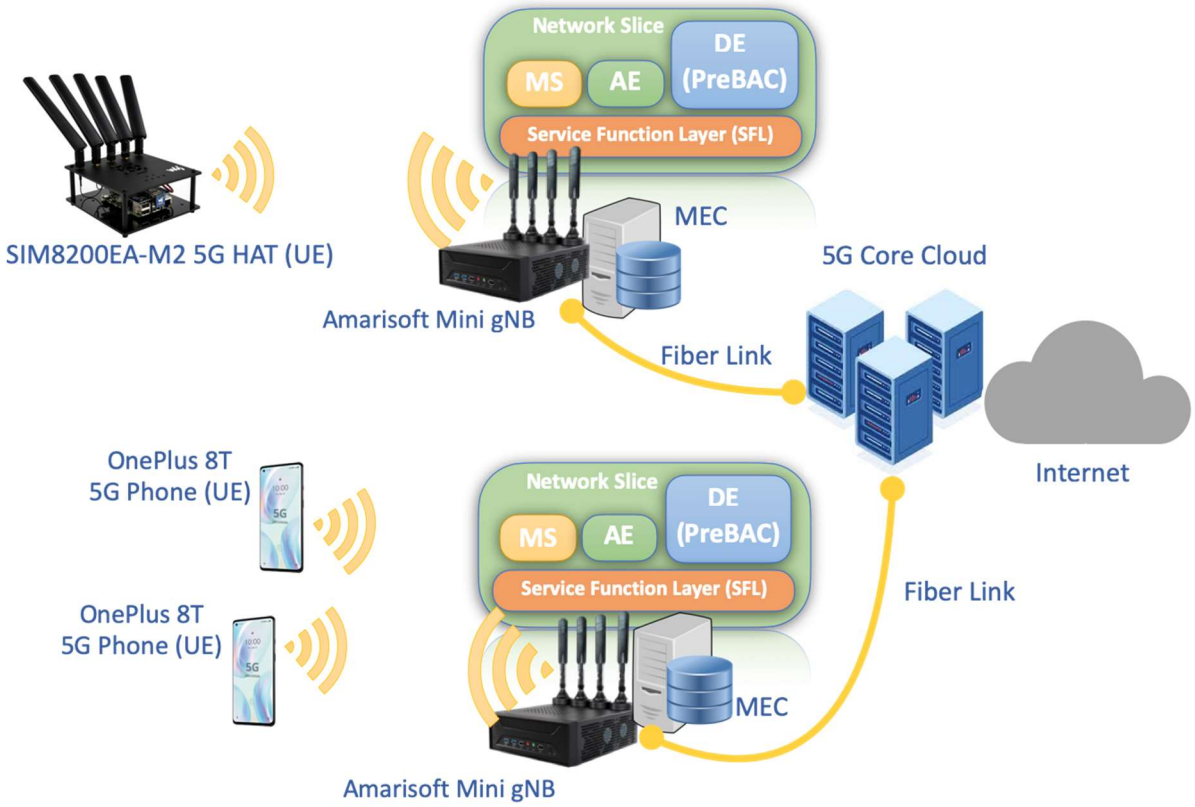

Figure 9. Experimental Platform for the implementation of PreBAC (DE).

3.3.2 MESSAGING INTERFACES

In its current deployment, PreBAC communicates with the rest of the MonB5G elements by extending Kafka Bus Interface deployed in the MS. This is feasible for PreBAC since it is deployed on-site within the same technological domain in which the MS and AE are deployed.

The communication from the EUI towards PreBAC is used mainly to alter configuration parameters of PreBAC, and some details about its control loop behaviour. The messaging interface used for this are POST requests through a REST interface. These requests are used to load JSON files that contain information on the configuration that controls PreBAC's behaviour. In the current implementation as of the time of this writing, the request can be issued to upload new configuration files, but the configurations won't take effect until the DMA sub-block is restarted. Future implementations, will include functionality for the configuration changes to take effect while running, as long as it is feasible for the specific type of configuration. All configuration files used by PreBAC are in JSON format.

3.3.3 INTEGRATION APIS

The MS and the AE are able to send messages to the DE using the Publisher/Subscriber interface provided by the Kafka Bus [10]. The DE includes a Kafka Interface that subscribes to two topics: one belonging to the MS and another to the AE. The DE communicates back to the COMS of the MS in order to store the action it issued for the current control cycle. To send this message, it also uses the Kafka Bus, with a separate topic of its own. The information flowing from the MS towards the AE are all marshalled in the same message, formatted according to the topic to which the DE is subscribed to. A similar procedure is applied do the messages coming from the AE. Table 3 shows the messages exchanged by the DE and the rest of the MonB5G components.

The information arriving to the DE from the AE and MS are firstly given to the Input Pre-Processor, which normalizes each field independently from each other, and then builds the state space representation necessary for PreBAC to carry out its decision-making process.

Sampled Metric	Source/Destination	I/O Data
Timestamp	MS/DE	Current aggregate bandwidth consumption in the domain (for all deployed slices)
Traffic load of slice 's' in current timestamp	MS/DE	The traffic load for a slice 's' in the current timestamp. The message sent includes the traffic load of each slice separately.
Total bandwidth capacity of BS	MS/DE	The total bandwidth capacity available in the BS.
Predicted traffic load of slice 's'	AE/DE	The predicted traffic load for the next timestep for slice 's'. This prediction is generated in the AE through an inference process performed by ECATP.
Decision for current control cycle	DE/MS	This is the resulting action issued by the DMA of the DE. It is stored in the COMS of the MS for visualization and evaluation purposes.

Table 3. Integration APIs with other MonB5G components

For a human user to communicate with the REST API of the DE, the POST requests that can be issued are shown in Table 4. The configuration files for PreBAC contains the parameters used by the SAC algorithm, which makes up the core of PreBAC. These parameters include the file path to the internal neural network parameters of SAC, the frequency of training, parameters of the neural networks of the SAC algorithm, and other related ML configurations.

REST API call	Source/Destination	Description

/config/prebac	EUI/DE	Upload configuration file for PreBAC, which is one of the DMAs implemented within the DMA.
/config/static	EUI/DE	Upload configuration file for static allocator, which is one of the DMAs implemented within the DA. This static allocator is deployed temporarily whenever PreBAC is being restarted or unavailable due to other maintenance processes.
/config/utilityParameters	EUI/DE	Upload configuration about the relative utility of the traffic load from different slices and their impact on the behavior of PreBAC.

Table 4. EUI integration APIs with the DE.

The utility parameter configuration files establish the relative revenue generated by servicing USRs from each slice. It also contains information of the penalties incurred when USRs are delayed and rejected. All these parameters are a representation of the actual penalties and utilities associated to each slice, but are represented in a way that can be consumed by PreBAC, and alter its behavior and policy.

3.3.4 MAPPING TO KPIS

PreBAC's ultimate objective is to reduce the overall rate of rejection of incoming USRs for the different slices, while increasing the overall utility of users, slice owners and infrastructure providers. The rate of rejection of USRs can be specified as an SLA.

PreBAC seeks to ensure that traffic that is considered more critical in terms of its Quality of Service (QoS) constraints and the amount of revenue it generates gets processed as it arrives, while at the same time preventing service starvation by the USRs for other slices that generate less marginal benefits.

In more concrete terms, the KPIs that PreBAC seeks to improve is the reduction of the probability of SLA violations. In this context, the probability of SLA violations translates into an overall traffic reject rate across the slices in the system, which PreBAC seeks to reduce. Specifically, we can map the KPI to the following:

[UC1/ES2 KPI-1]: Reduce the number of SLA performance violations by 20%

3.4 Time-of-day Aware Slice Admission Control (DE) based on traffic prediction, LSTM-based anomaly and fault detection

The goal of Time-of-day Aware Slice Admission Control (TASAC) DE is to derive optimal slice admission policy on the basis of current resources usage by the network slices and the predicted future consumption (see

MonB5G Deliverable D4.2 [1]. The TASAC DE is placed in DE sublayer of MonB5G architecture either in the DMO or IDMO. Figure 10 shows the implementation architecture of the TASAC DE in the DMO scenario along with the type of information it exchanges with specific MS (Resource KPI Aggregator) and AE (Aggregate Resource Consumption Predictor) components within MonB5G architecture.

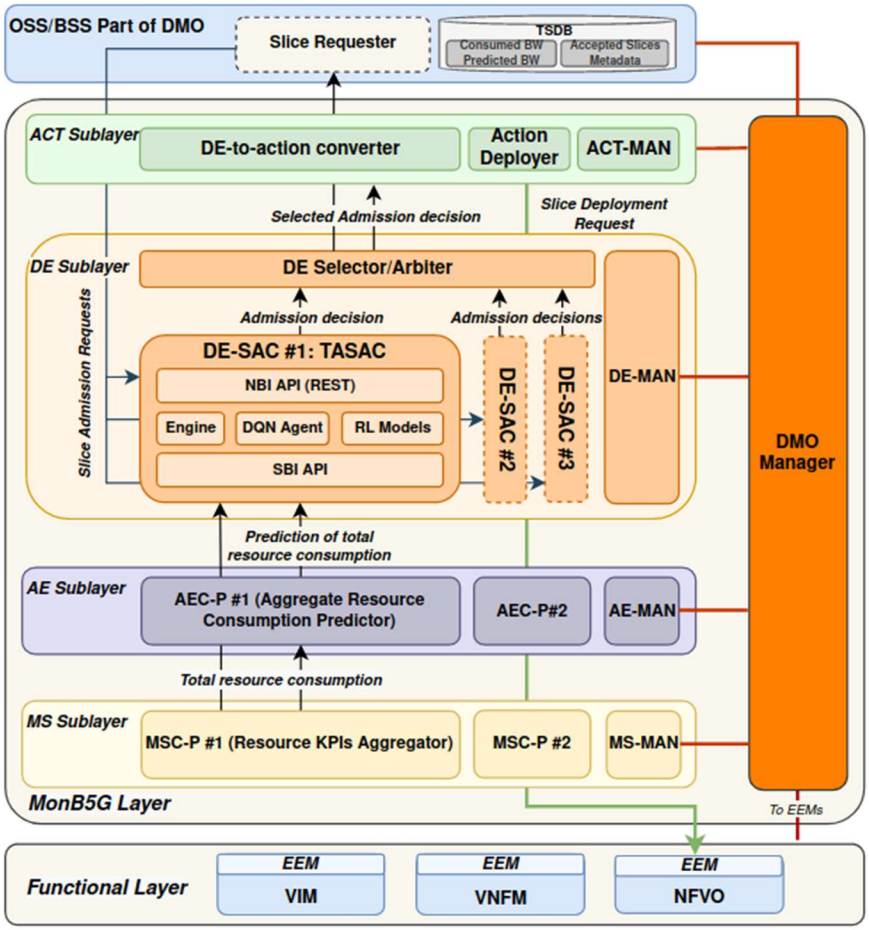

Figure 10. Example placement of TASAC DE in DMO and its integration with MonB5G Architecture components

The TASAC DE is integrated with the Time-series Database (TSDB), which is the current implementation of the COMS component, and Slice Requester solutions. The TSDB is responsible for storing the information about consumed resources, predicted resource consumption, TASAC DE algorithm-specific metrics, and metadata of all accepted slices. The algorithm-specific metrics of TASAC DE include variables such as the Deep Q-Network (DQN) rewards, DQN penalties, output Q-values and DQN decision history. The Slice Requester simulates the influx of slice admission requests issued by the slice tenants (sent over Iid/Ipi

interface to DMO/IDMO). The most common use case would involve using the TASAC DE as a block assisting DMO or IDMO. It has to be noted however that the implementation is generic and the scope of operation of the TASAC DE can be adjusted to specific use cases (e.g. better usage of a part of domain's resources).

3.4.1 DEVELOPMENT FRAMEWORKS

Both TASAC DE as well as Slice Requester have been implemented using Python 3.10 and containerized with Docker. TASAC exposes the REST API complementary with the OpenAPI framework [11]. The AI-driven mechanisms (such as DQN agent) are implemented using Keras and Tensorflow [7] as the compute backend. The example results of Kubernetes-based TASAC DE deployment that aims to maximize the utilization of system bandwidth is presented in Figure 11.

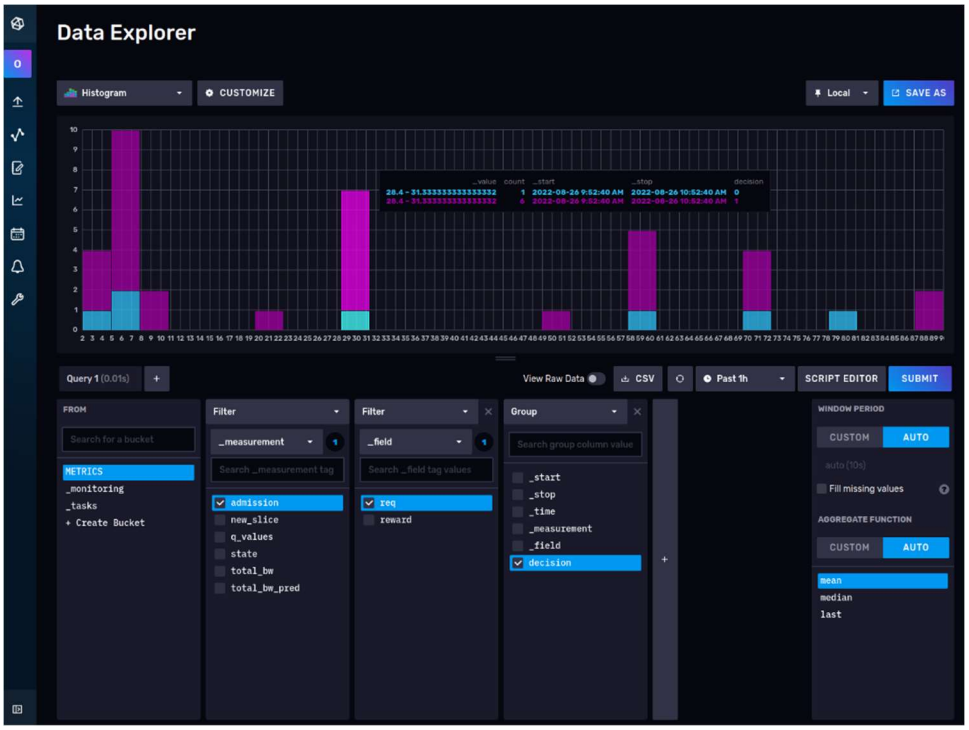

Figure 11. Histogram of slice admission and rejection depending on the requested resources.

3.4.2 MESSAGING INTERFACES

TASAC DE implements two interfaces to communicate with MonB5G entities the web-based North-bound Interface (NBI), implemented as a REST interface, and message-based a South-bound Interface (SBI) implemented with Kafka [10]. The NBI exposes the methods for requesting Slice Admission Decisions based

on the parameters of a slice request issued by a tenant. The SBI is a Kafka-based interface that is used to consume network state messages (current and predicted resource consumption) as well as post data related to the undertaken slice admission decisions and other metrics that can be leveraged by other MS/AE/DE (e.g. to verify the module stability, DQN agent performance, etc.). The messages published to the message bus are serialized to the line protocol format to facilitate easier consumption by the metric collector (Telegraf [12]) and injection to the TSDB (InfluxDB [13]).

3.4.3 INTEGRATION APIS

The TASAC DE communicates with the MonB5G entities via the message bus and web-based interface (Section 3.4.2). Therefore, the southbound integration with MonB5G system is achieved by specifying a common list of message keys. The analyzed use case involves the deployment of TASAC DE cooperating with MS and Resource Consumption Predictor (described in [14]) to maximize the utilization of bandwidth in the domain. The exemplary list of messages published and subscribed by the TASAC DE (marked as PUB and SUB respectively) is presented in Table 5.

Message key	Type	I/O Data
ms.ms.total_bw	SUB	Current aggregate bandwidth consumption in the domain (for all deployed slices)
ae.rcp.total_bw_pred	SUB	The total bandwidth prediction for the next sample (based on the predefined interval)
ae.rcp.api.prediction.req	PUB	Prediction requests containing the time range of prediction, unique request id
de.sac.new_slice	PUB	The message generated in case of slice admission request acceptance (used by RCP to monitor the list of all accepted slices for the purpose of prediction)
ae.rcp.api.prediction.res	PUB	Prediction response containing the predicted values of the resource consumption in the requested time period together with the unique request id

Table 5. Messages exchanges via the SBI.

The upper-layer entities (e.g. DMO, IDMO) can communicate with TASAC DE using the API described in Table 6.

Endpoint	Method	Data
/api/v1/admission/request/	POST	Slice Admission request containing the request ID, slice type, the maximum amount of requested resources, the time of slice

		deployment and termination, priority and the REST-based endpoint to communicate the slice admission decision
/api/v1/healthcheck/	GET	Returns the health of TASAC DE
/api/v1/	GET	API Index

Table 6. TASAC DE Norhbound API.

3.4.4 MAPPING TO KPIS

The primary goal of the TASAC algorithm is to increase the utilization of resources in the given time period by taking optimal slice admission decisions. The approach can also potentially lead to indirect gains such as e.g. improved OPEX (higher return under the same available resource pool). Also, the adopted DQN-based algorithm penalizes heavily any possible excessive admission (i.e. over the available resource pool) to mitigate SLA violations.

Altogether, the TASAC enabler contributes to the maximization of NS acceptance ratio and improves the folowing MonB5G KPIs [15]:

- **(UC1/ES2 KPI-1)**: Reduction of SLA Violations
- **(UC1/ES2 KPI-4)**: OPEX reduction due to the automation of service management

3.5 Distributed Slice Resource Allocation in the RAN domain

While advanced admission and control mechanisms could select the set of slices to be admitted to the system and set static resource allocation limits to satisfy the available capacity, the dynamic and heterogeneous nature of the slice's traffic load and wireless channel statistics may lead to suboptimal network performance in the long run. Due to rapid traffic fluctuations, slice resource allocation decisions in the RAN domain should be dynamic, proactive, and flexible to avoid degradation of service and performance.

For these reasons, we use an FDRL-based architecture to address RAN resource allocation in the slicing scenario. In particular, we rely on local Decision Agents DAs, one per slice, running as software instances within the premises of each BS, as shown in Figure 12. Each agent is responsible for making slice Physical Resource Block (PRB) allocation decisions based on local monitoring information received from the underlying network monitoring system. We refer to such monitoring information as *Base Station context*.

To solve the above problems simultaneously, we introduce a FL layer that allows inter-agent information exchange and expedites the learning procedure of local knowledge sharing. Unlike multi-agent reinforcement learning, which defines a set of autonomous agents that observe a global state (or partial state) of the system, select individual actions and receive individual rewards, FL enables training of machine learning models over

multiple decentralized entities that have access to a limited portion of the overall available data, i.e., in our case, access to local monitoring information only.

FL allows local DEs to collaboratively learn a shared prediction model by iteratively aggregating multiple model updates and returning a refined version of it that combines multiple local models according to specific federation strategies. This decouples the learning procedure from the need for centralized data sources. The refined model is then shared with the agents which significantly allows to improve the learning rate, ensure privacy and enable better generalization.

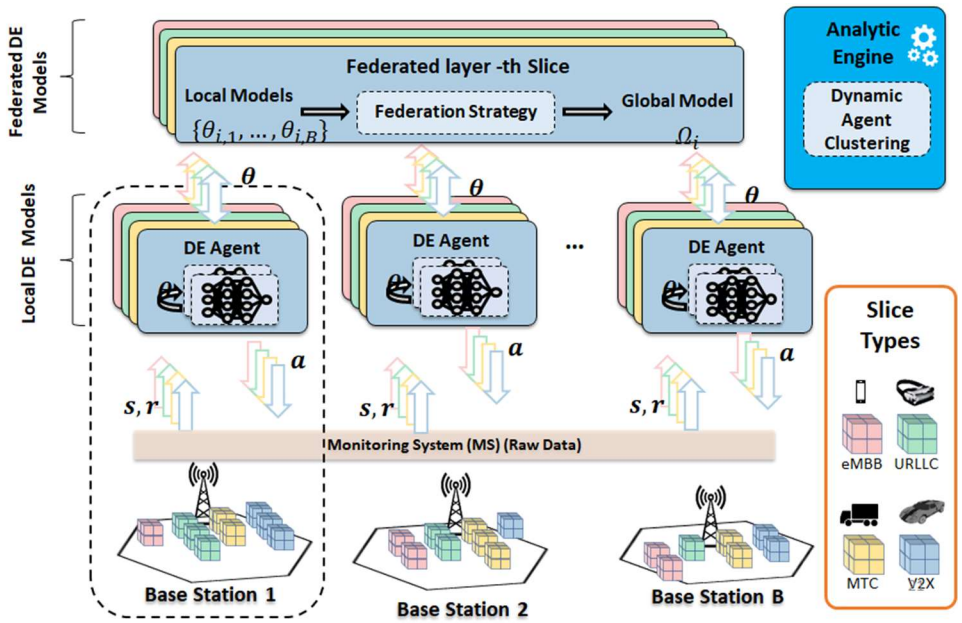

Figure 12. Generic Federated DRL architecture for RAN slicing.

Related work and large-scale simulation results of the overall framework have already been presented in Deliverable 4.2. To further validate our key idea and architectural components, we plan to develop a working testbed consisting of 5G equipment and control software. Below, we outline the key architectural components and perquisites for full implementation as well as the initial APIs required to monitor KPI metrics and enforce allocation decisions.

3.5.1 DEVELOPMENT FRAMEWORKS

We implement our framework in Python programming language, exploiting OpenAI Gym library [6] and interfacing DRL agents with a custom base station simulator environment, which includes virtual transmission queues and main PHY/MAC/RLC functionalities, together with O-RAN E2 interface to allow gathering the networking statistics from each distributed unit (O-DU), and to enforce PRB policy decisions in the BS slice scheduler based on defined state space and action space, as depicted in Figure 13.

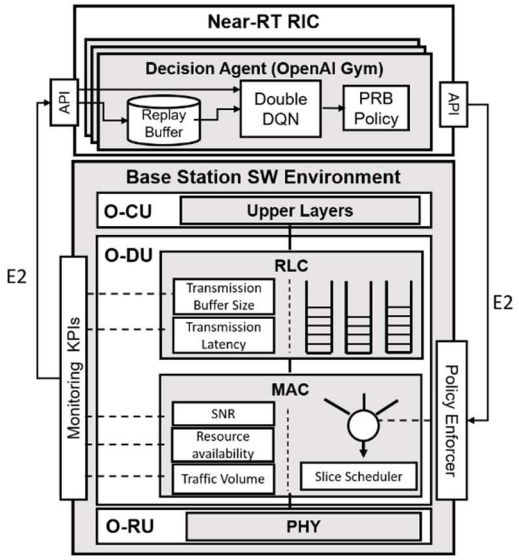

Figure 13. Software architecture and protocol stack overview.

.

In order to validate our framework in realistic settings, we consider the city of Milan, Italy, as scenario of study. We collect city-wide RAN deployment information including more than 50 BSs from publicly available sources, and simulate realistic human mobility patterns leveraging the work of [16]. An example of the generated data is available in the following Figure 14.
The softwarized agent instances interact with the simulation environment and collects simulated base station traffic. We evaluate our proposed architecture through an ad-hoc simulator running on a dedicated server that is equipped with two Intel(R) Xeon(R) Gold 5218 CPUs @ 2.30GHz and two NVIDIA GeForce RTX 2080 Ti GPUs.

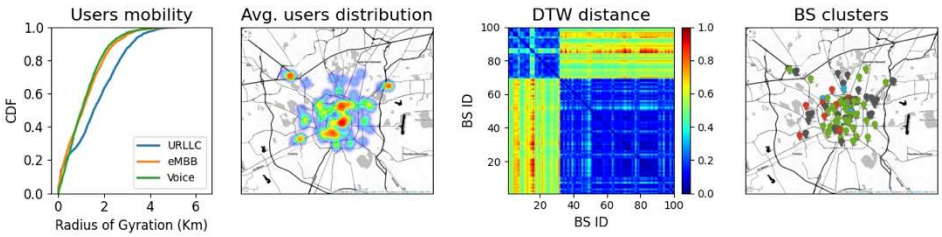

Figure 14. Simulation data overview.

We refer the reader to Deliverable 4.2 for a more in-depth evaluation of the overall framework.

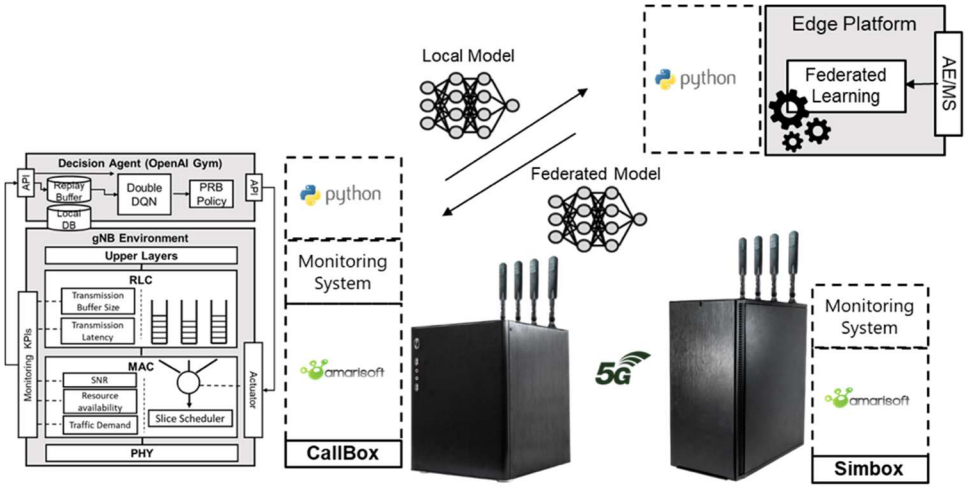

Figure 15. Testbed Architecture.

Additionally, a preliminary implementation exploiting a fully-fledged 5G emulator has been already achieved, as part of WP6 activities. Figure 15 shows the architecture of our distributed slice resource allocation solution in the RAN domain.

The preliminary testbed implementation includes the following main components:

- **Amarisoft Callbox:** It acts as a 3rd Generation Partnership Project (3GPP) compliant with eNodeBs (eNBs)/gNodeBs (gNBs) and Enhanced Packet Core (EPC)/5G Core (5GC) and enables functional and performance testing. Thanks to its multi-cell configuration, it is also suitable for handover and reselection tests. It also supports 5G New Radio (NR) Non-Standalone (NSA) mode. Table 7 summarizes the technical specification of the device.
- **Amarisoft Symbox:** It is capable of simulating hundreds of User Equipments (UEs) sharing the same spectrum with different types of traffic within multiple cells. Each UE can be independently configured as a Long Term Evolution (LTE), 5G NR, Narrowband Internet of Things (NB-IoT) or LTE Machine Type Communication (LTE-M) device.
- **Monitoring System:** It allows real-time monitoring information to be retrieved from the gNBs platform. Such information is processed by the local decision agent to develop its radio resource allocation policy.
- **Local Decision Agent:** It is developed within the MonB5G framework, runs as a container instance on the base station premises, collects and consumes local monitoring information from MS and adjusts the radio resource allocation policies accordingly. The decision-making task is supported by the AI/ML algorithm.
- **Federated Learning Layer:** It acts as an aggregation point for the local decision engines deployed in RAN. It collects locally trained (and therefore heterogeneous) decision models and combines them to gain global knowledge about the underlying infrastructure behaviour and improve the generalization of the decision process.

.

3GPP release	*release 16*
Frequency bands	*All FDD and TDD bands in sub-6GHz*
Bandwidth	*Up to 50 MHz*
Supported number of cells	*3*
Number of active UEs	*Up to 1000 UEs distributed within the configured cells*
Carrier aggregation	*Up to 3 carriers in DL and 3 in UL allowing mixed FDD/TDD combinations in DL*
Supported modes	*NSA, SA*
Transmission modes	*1 (single antenna) to 10 (MIMO 4x4)*
Modulation schemes	*Up to 1024QAM in DL and 256QAM in UL*
AS encryption and integrity protection	*AES, SNOW3G, ZUC*
Handover	*NG, Xn and 5GS to EPS handover support*
eNodeB network interfaces	*S1AP and GTP-U to EPC X2AP between eNodeBs M1 and M2 for eMBMS*
ng-eNodeB network interfaces	*NGAP and GTP-U to 5GC XnAP between ng-eNodeBs*
Subcarrier spacing	*Data subcarrier spacing: 15, 30, 60 or 120 kHz SSB subcarrier spacing: 15, 30, 120 or 240 KHz*

Table 7. Amarisoft Callbox gNodeB Technical Specifications.

3.5.2 MESSAGING INTERFACES

Local Decision agent <–> Monitoring System

Decision engines are responsible for making reconfiguration decisions to optimize the performance of the functional layer according to the collected data from the MS sublayer and the analysis results from the AE sublayer. The input to the DE sublayer is the output of the AE and MS sublayers. For details regarding the MS/AE sublayers integration and interaction, we refer the reader to the deliverable D3.3. The generic structure of the proposed **MS** is shown in Figure 16. In this example, it consists of 11 sampler functions monitoring and reporting various key parameters belonging to the different protocol stack layers composing the Amarisoft and 5G RAN software.

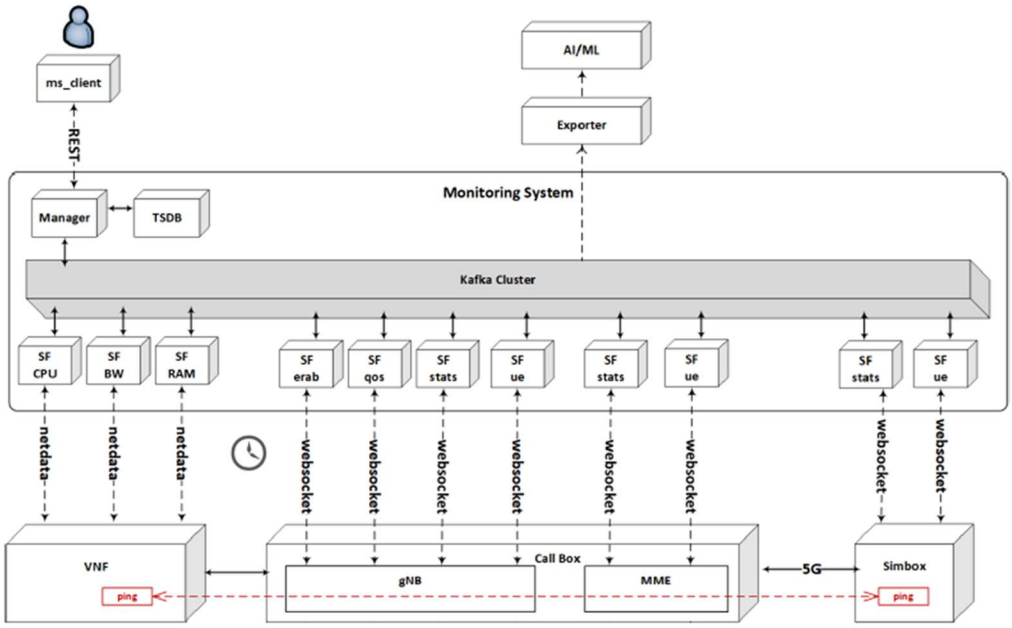

Figure 16. Monitoring System.

The MS sublayer deals with both telemetry data (continuous time series) and event data (e..g., alarms, faults, and topology changes). It enables monitoring with varying temporal granularity and varying degrees of data aggregation, depending on the optimization goal.

Local Decision agent <– > Federated Layer

As shown in Figure 12, each DE agent in our Federated Deep Reinforcement Learning-based framework embeds and trains a local Duelling Deep Q-Network (DDQN) model. The trained model, that has learned a policy from the available data, is shared under the form of model weights to the entities that belong to the corresponding federation layer. Each federation layer aggregates the accumulated knowledge of each agent into a global updated model, which is typically stored in a cloud platform or a nearby edge platform to enable faster feedback.

To increase efficiency and avoid communication overhead, we allow the federation layer to collect the local models only every $\hat{T}$ decision intervals, being $\hat{T}$ a tunable parameter. We define this time period as a *federation episode*.

Local Decision agent <–> Analytic Engine

The aggregated mobile traffic demands follow recurring spatiotemporal trends due to human activities. In this context, it is expected that a good characterization of such processes would allow a more accurate forecasting of network utilization and thus a more efficient and even proactive planning of resource allocation. We rely on the Analytic Engine to provide this kind of functionality and use its API to obtain this additional information.

3.5.3 INTEGRATION APIS

The decisions of the DE are based on the information available on the underlying MS/AE sublayer. In particular, the DE creates the observation state by collecting the quality of the channel, and the aggregate traffic demand per slice. We refer the reader to D4.2 for more details about the proposed DE. The information mentioned earlier is available at the MS/AE sublayer via the stats sampler function in the gNB, which returns the aggregated traffic demand per slice and the channel quality.

The PRB allocation decisions made by the DE are then sent to the gNB via the config_set web socket message, which then allocates the selected PRBs to the radio interface.

3.5.4 MAPPING TO KPIS

- **[UC1/ES2 KPI-4]** OPEX reduction due to the automation of service management.
- **[UC1/ES2 KPI-7]** Optimize the convergence time for the distributed/federated AI algorithms so that it does not exceed that of the centralized solutions.
- **[UC1/ES2 KPI-9]** Improve network slice performance prediction
- **[UC1/ES2 KPI-11]** Reduce time to manage RAN resources dedicated to network slices, particularly for uRLLC (AE and DE are located at the edge).
- **[UC1/ES2 KPI-12]** Improve on slice performance isolation by ensuring the latency and reliability (uRLLC), as well as bandwidth (eMBB) requirements of coexisting slices (measured in terms of related SLA violations and other lower-level metrics).
- **[UC1/ES2 KPI-13]** Reduce the management overhead of the RAN by reducing the monitoring overhead for RAN-level slice resource (and other) reconfigurations.

3.6 Independent DQN Agents for Slice Reconfiguration

In D4.2 (Section 6.2) we proposed a DE utilizing Independent Deep Q Network (IDQN) agents for dynamic end-to-end slice reconfiguration. This DE is well in line with the key characteristics of the inter-slice orchestration problem, since it has a wide view of the system. The goal in the slice reconfiguration problem is to dynamically decide the system's configuration (each IDQN agent is associated with a VNF and decides which will be the host server in the next time-slot), towards maximizing the accumulated reward over a discounted infinite time horizon (while not knowing a priori how slice demands evolve over time).

The scalability and effectiveness of the IDQN algorithm was extensively validated in D4.2 [1] with realistic simulations. In this deliverable we will describe the implementation details.

3.6.1 DEVELOPMENT FRAMEWORKS

The DE was implemented with Python, while PyTorch was utilized to implement the Deep Neural Networks (DNNs) of the agents. Towards a cloud native implementation, the whole DE can be placed inside a Docker container. There are two main entities involved in the DE, which are the following:

System Model. This entity contains three components:

- Physical Network: It contains all the information related to the physical network, like system configuration, domains, server IDs, server capacities, connectivity, etc.
- Slices: It contains all the relevant information for the slices to be orchestrated, like slice IDs, VNFs, Virtual Links, etc. Also, the timeseries of the slice resource demands are imported from a dataset (either Milano or a synthetic Markovian dataset) and are loaded to the corresponding slices.
- Reward function: This gets as an input the previous state and the current state of the system and outputs the reward as defined in D4.2 [1].

Controller. This may be either the proposed IDQN agents or the baseline solution, which include a single-agent DQN, Q-Learning (QL) and static policies. The input to this component is the state (current configuration and resource demands) and the reward given by the system model, while the output is the configuration to be applied in the next time-slot. Note that in this implementation the agents operate sequentially. However, in a future implementation they could operate in parallel since they are independent, either in a centralized way by assigning them to different CPUs, or in a more distributed architecture.

In Figure 17 there is a schematic representation of the interaction between the IDQN agents of the controller and the system model.

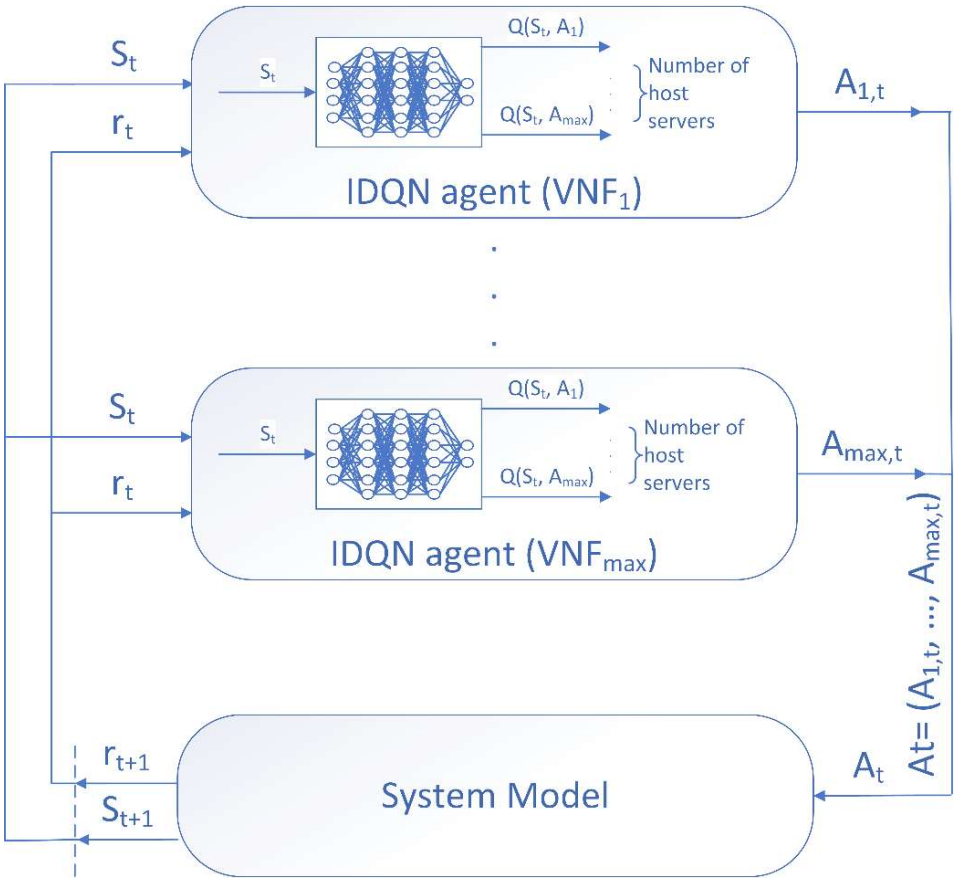

Figure 17. Schematic representation of the DE.

Towards constructing a very basic cloud-native testbed, we have additionally implemented a simple actuator in Python that can enforce the decisions of the DE. The testbed considers a centralized DE (controller + system model + actuator) with a global view of the system, while the system consists of VMs hosting containerized VNFs (Figure 18). Note that VMs map to the physical nodes of the system model we defined in [1]. The actuator can enforce the reconfiguration decisions of the controller using Docker commands to migrate containers from the host VMs to the destination VMs. This is implemented can be implemented with the Docker Software Development Kit (SDK) for Python.

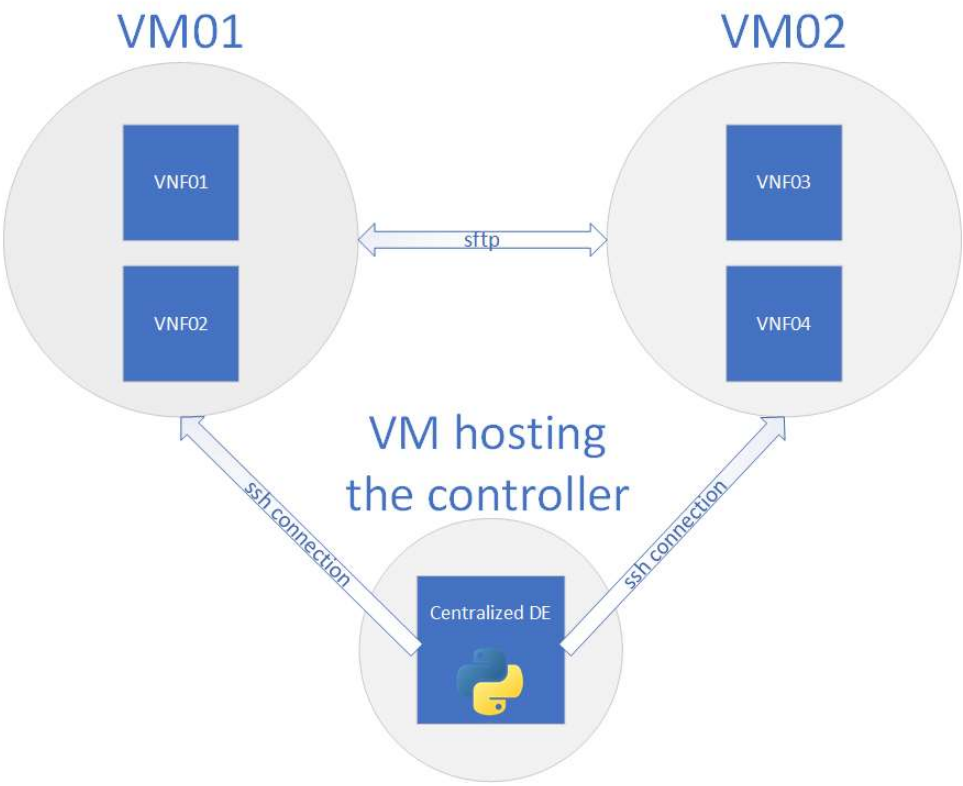

Figure 18. A simple testbed example.

3.6.2 MESSAGING INTERFACES

Controller (IDQN agents) <-> System model

The two main entities of the DE communicate internally at each time-slot. The agents give as an input to the system model the next configuration, and the system model returns the next state and the reward (Figure 17).

Centralized DE –> VMs

A secure shell (SSH) connection must be established between the centralized controller and each of the VMs so that the controller can communicate with the remote Docker Daemons and enforce its decisions (stop a container, commit it into an image, remove a container, or run a container).

VM <–> VM

The secure file transfer protocol (SFTP) is used between the VMs in order to transfer files related to container migration from the host VM to the destination VM.

3.6.3 INTEGRATION APIS

The code is written in a Python script, and the physical network, the slices, the reward function, and the controller are just different objects with public attributes. Therefore, these attributes can be accessed by other objects just by passing them as arguments in corresponding methods.

The MS in our system stores the resource demand vector at each timeslot. It is equipped with a method called "get_demand_vector", which gets as input argument a list with all the slice objects. So, this method collects the resource demands from the different slices and stores them as the current resource demand vector.

The controller object includes a method called "choose_action", which gets as input arguments the configuration vector from the physical network object, the resource demand vector from the monitoring system (these 2 form the state representation of the learning agent), and a reward from the reward function object, and gives at its output the next configuration to be applied on the system. It also stores the experiences to the buffers of the agents and updates the DNN parameters of the agents.

The above information is summarized in Table 8.

Object	Method	Args	Functionality
Monitoring system	get_resource_demands	A list with the slice objects	Collects resource demands from all slices and stores them as the current resource demand vector
Controller	choose_action	physical_network.configuration, monitoring_system.resource_demands, reward_function.reward	Returns the next system configuration, stores experience, updates DNN parameters

Table 8. Methods for interaction with the DE.

Note that the controller could be possibly used separately as a containerized component that gets as an input all necessary information from an external monitoring system (possibly through a Kafka bus) and gives as an output the reconfiguration decision (a new system configuration). This decision should be then transferred to the component responsible for enforcing container migration.

3.6.4 MAPPING TO KPIS

The IDQN solution was validated by realistic simulations in D4.2 [1]. It relates to the following project KPIs:

- **[UC1/ES1 KPI-1], [UC1/ES2 KPI-1]:** Reduction of SLA violations
- **[UC1/ES2 KPI-2]:** Reducing static slicing overhead will result in 30% higher utilization (will be achieved with dynamic reconfiguration techniques)
- **[UC1/ES2 KPI-3]:** Compared to Static Slicing, demonstrate the same or better SLA tolerances (or risk of missing SLAs) when dynamic slicing techniques are used
- **[UC1/ES1 KPI5], [UC1/ES2 KPI-4]:** OPEX reduction due to the automation of service management
- **[UC1/ES2 KPI-7]**: Optimize convergence time for the distributed/federated AI algorithms so that it does not exceed that of centralized solutions

The cost to be minimized by IDQN has been defined as the weighted sum of three different cost terms, the node utilization cost (it is equal to the number of active nodes and relates to energy consumption and rent costs), the reconfiguration cost (the cost for migrating VNFs), and the SLA violation cost (a penalty paid to the slice tenants when an SLA violation is observed for their slice). These cost terms can target all the KPIs of the first four bullets above. Then, regarding KPI-7 of ES2, the multi-agent IDQN scheme demonstrates faster convergence compared to the single-agent DQN (validation section of D4.2 [1]). A remark here is that IDQN conceptually distributes the decision to different agents, but that doesn't mean that the agents must be also spatially distributed (they require global information and not local).

3.7 FISH Recommender for Control Loop Coordination

Control Loop Coordination (CLC) is a common problem in network optimization. When multiple CLs with common modifiable parameters exist in a system, a ping-pong reconfiguration effect can occur, where both CLs continuously and selfishly opt for a parameter reconfiguration. In MonB5G architecture [17], we assume CLs to be sets of Monitoring Services, AEs and DEs, followed by Actuators. DEs are outermost modules responsible for parameter modification inside a CL. The FISH Recommender works as a **DE Selector/Arbiter** located within DE-Sublayer in MonB5G Layer either in DMO, IDMO, or Slice Management Layer (SML) [17], evaluating possible impact of reconfiguration on other functions and either allowing or rejecting such reconfiguration request.

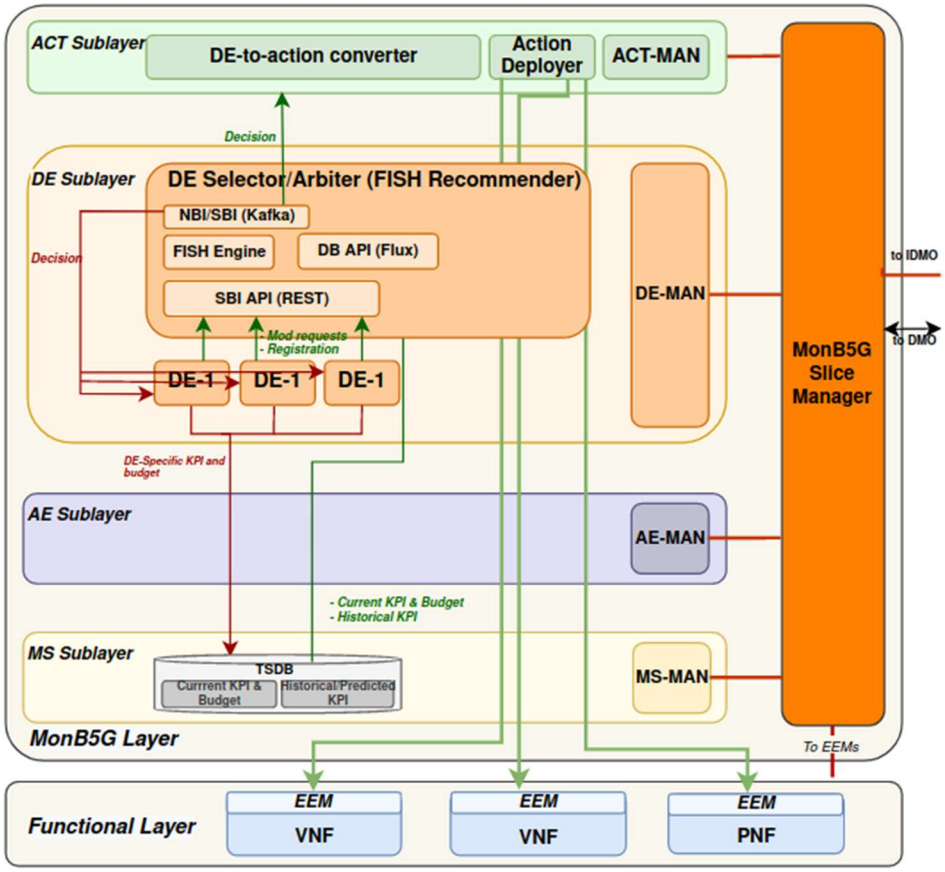

Figure 19. Cooperation of FISH Recommender with in MonB5G Architecture.

3.7.1 DEVELOPMENT FRAMEWORKS

The FISH Recommender uses a slightly modified Fisher Market Model as an evaluation method. The key concept of this approach revolves around treating DEs as "buyers" that want to "buy" certain parameters from a reconfiguration request. Every buyer has a certain budget and, based on historical/predicted data, each parameter has a certain weight (importance) for each buyer. The Fisher Market Model resolves the market in the way that all products on the market are bought and all the money from buyers is spent. The final decision regarding reconfiguration is made depending on resulting utility function, which represents which of the "buyers" would be affected the most by the proposed reconfiguration. More detailed description of described coordination framework is available in Deliverable D4.2 [1]. FISH has been written using Python 3.10 and can be deployed in Kubernetes [18].

3.7.2 MESSAGING INTERFACES

Intercommunication between FISH and other modules has been presented on Figure 19.

In order to perform recommendation, FISH needs to collect the following information as an input:

- proposed parameter modification vector;
- current KPI and Budget of each considered DE;
- historical KPI/ Predicted KPI for proposed parameter set for each considered DE.

3.7.3 INTEGRATION APIS

- DEs can interact with FISH via web-based SBI (**REST** API) with endpoints:

  /register -> for DE registration, after which they can be passively included in any reconfiguration request assessment

  /reqmod -> in order to send parameter modification request

- FISH queries historical/predicted as well as current KPI data using a database (DB) API. The type of DB used in this solution is a TSDB implemented with InfluxDB [13].
- After assessment, FISH publishes its decision onto the Message Bus (i.e. Kafka [10])**.**

3.7.4 MAPPING TO KPIS

By providing impact-based function coordination, the FISH Recommender can eliminate unnecessary reconfigurations and avoid possible conflicts between different DEs, leading to overall stability and system performance improvements. Limiting the number of unfavorable reconfigurations in the RAN domain can contribute to a reduction of management overhead of the RAN (by not needing to perform nor fix such reconfiguration), which is the goal of **UC1/ES2 KPI-13**.

4 Timeline for Deployment and Completion

The DE and its associated MS/AE implementations for their respective use cases are currently being deployed in the available experimental platforms for real use case testing shown in Figure 9 and in Figure 15. Both of these experimental platforms conform an industry and state-of-the-art standard 5G platforms in which the DEs and its corresponding MSs/AEs will be deployed and validated.

It is expected that the deployment of the DEs and the respective MonB5G components will be completed by February 2023, with enough time to allow for all partners to deploy and converge on their DE implementations, and generate the upcoming deliverables for the project.

5 Conclusions

In this deliverable, we have presented details on the integration of the different state-of-the-art DEs proposed within the framework of the MonB5G project to solve different open problems regarding the management and orchestration of 5G and Beyond 5G communication networks. In this document, each DE implementation was presented with detailed information on how it was integrated with its corresponding MS and AE, also implemented for particular use case for which a specific DE implementation is deployed.

This work is clearly a continuation of the work already presented in Deliverables D4.1 [2] and D4.2 [1], with an incremental improvement of the generic DE architecture that makes it more compliant to the ETSI-ENI standard [3], as well as expanding deeper on the way this generic architecture maps to actual implementations. This is better understood after including a recap of the generic DE architecture.

The sections corresponding to each implementation of the DE, presented in Section 3, showed the successful functional testing and validation of the integrated DE with its corresponding MS and AE components, demonstrating the feasibility of deployment and integration.

6 Bibliography

[1] F. Guillemin, L. Blanco, F. Rezazadeh, E. Zeydan, H. Chergui, J. Mangues, S. Kahvazadeh, S. Kuklinski, R. Kolakowski, R. Tepinski, M. Rosinski, J. Jegier, T. Spyropolous, P. Doanis and K, "Deliverable D4.2: Final Report on AI-driven Techniques for the MonB5G Decision Engine", June 2022.

[2] T. Spyropoulos, T. Giannakas, P. Doanis, M. Constantini, C. Verikoukis, H. Chergui, D. Pubill, J. Serra, L. Blanco, S. Kahvazadeh, L. A. Garrido, A. Dalgkitsis, L. Zanzi and Devoti, "Deliverable D4.1: Initial Report on AI-Driven Techniques for the MonB5G Decision Engine", March 2021.

[3] ETSI GS ENI 005 v2.1.1, "Experiential Networked Intelligence (ENI); System Architecture", December 2021.

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning", New York: In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16), 2016.

[5] S. Zhang, H. Tong, J. Xu and M. Ross, "Graph convolutional networks: a comprehensive review", Computational Social Networks, November 2019.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540,* 2016.

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard and others, ""TensorFlow: A System for Large-Scale Machine Learning"," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, Savannah, GA, USA, November 2016.

[8] L. A. Garrido, P. V. Mekikis, A. Dalgkitsis and C. Verikoukis, """Context-Aware Traffic Prediction: Loss Function Formulation for Predicting Traffic in 5G Networks"," in *ICC 2021 - IEEE International Conference on Communications*, 2021.

[9] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, ""Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor"," in *Proceedings of the 35th International Conference on Machine Learning*, July 2019.

[10] J. Kreps, N. Narkhede and J. Rao, ""Kafka: A Distributed Messaging System for Log Processing"," in *6th Workshop on Networking Meets Databases (NetDB)*, 2011.

[11] Linux Foundation, ""OpenAPI Specification v3.1.0"," Linux Foundation, February 2021. [Online]. Available: https://spec.openapis.org/oas/v3.1.0. [Accessed 28th November 2022].

[12] InfluxData, "Telegraf 1.24 documentation," InfluxData, [Online]. Available: https://docs.influxdata.com/telegraf/v1.24/.

[13] InfluxData, "InfluxDB OSS 2.5 Documentation," InfluxData, [Online]. Available: https://docs.influxdata.com/influxdb/v2.5/.

[14] X. Xu, B. Bakhshi, L. Blanco, E. Zeydan, H. Chergui, J. Mangues, J. Serra, G. Tsolis, K. Adlen, M. Costanti, S. Ben Saad, T. Spyropoulos, A. Dalgkitsis, L. A. Garrido Platero and Bosneag, ""Deliverable D3.2: Final Report on AI-driven Techniques for the MonB5G AE/MS"," MonB5G, March 2022.

[15] OTE, ""Deliverable D2.2: Techno-economic analysis of the beyond 5G environment, use case requirements and KPIs"," November 2020.

[16] L. Pappalardo and F. Simini, ""Data-driven Generation of Spatio-Temporal Routines in Human Mobility"," *Data Mining and Knowledge Discovery,* vol. 32, no. 3, p. 787–829, 2018.

[17] S. Kukliński, L. Tomaszewski, R. Kołakowski, J. Fabisiak, A. Ksentini, S. Ben Saad, A. N. Abbou, O. Hireche, C. Benzaid, T. Taleb, A. Boubendir, J. J. Alves Esteves and A. M. Bosneag, ""Deliverable D2.4: Final release of the MonB5G architecture (including security)"," MonB5G, October 2021.

[18] The Kubernetes Authors, ""Kubernetes: Production-Grade Container Orchestration"," 2022. [Online]. Available: https://kubernetes.io. [Accessed 2022].