A Secure and Reusable Artificial Intelligence Platform for Edge Computing in Beyond 5G Networks

D3.2 Final report on systems and methods for AI@EDGE platform automation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 10101592

D3.2 Final report on system and methods for AI@EDGE platform automation	
WP	WP3 – AI@EDGE platform for network automation
Responsible partner	RISE Research Institutes of Sweden AB (RISE)
Version	1.0
Editor	Bengt Ahlgren (RISE)
Authors	Bengt Ahlgren (RISE), Fehmi Ben Abdesslem (RISE), Akhila Rao (RISE), Anders Lindgren (RISE), Daniel Perez-Ramirez (RISE), Flávio Brito (EAB), Josue Castaneda (EAB), Neiva Linder (EAB), Zere Ghebretensaé (EAB), Nour Yellas (CNAM), Salah Bin Ruba (CNAM), Omar Anser (INRIA), Wei Jiang (DFKI), Estefania Coronado Calero (i2CAT), Javier Palomares Torrecilla (i2CAT), Juan Camargo (i2CAT), Claudia Torres (i2CAT), Miguel Catalán-Cid (i2CAT), Cristina Cervelló-Pastor (UPC), Alejandro Llorens-Carrodeguas (UPC), Sebastià Sallent (UPC), Nicola di Pietro (ATH), Javier Renart (ATOS), Enrique Lluesma (ATOS)
Reviewers	Roberto Riggio (UNIVPM), Flávio Brito (EAB), Nour Yellas (CNAM), Anders Lindgren (RISE), Claudia Torres (i2CAT), Cristina Cervelló-Pastor (UPC), Omar Anser (INRIA), Daniel Reti (DFKI), Bengt Ahlgren (RISE)
Deliverable Type	R
Dissemination Level	PU
Due date of delivery	2023-09-30
Submission date	2023-09-29

Version History				
Version	Date	Authors	Partners	Description
1.0	2023-09-29	Bengt Ahlgren	RISE	Final version

Disclaimer

Table of Contents

List of Tables

List of Figures

Glossary	
AI	Artificial Intelligence
AIF	Artificial Intelligence Function
AMF	Access and Mobility management Function
CPU	Central Processing Unit
CSI	Channel State Information
DL	Deep Learning
DNN	Data Network Name
DCP	Data Collection Pipeline
FL	Federated Learning
GPU	Graphic Processing Unit
IARM	Intelligent Acceleration Resources Manager
IOC	Intelligent Orchestration Component
IoT	Internet of Things
KPI	Key Performance Indicator
LCM	Life Cycle Management
LSTM	Long Short-Term Memory
MEC	Multi-access Edge Computing
ML	Machine Learning
NFV	Network Function Virtualization
NSAP	Network and Service Automation Platform
ORAN	O-RAN Alliance, developing open RAN standards

QoS	Quality of Service
RAN	Radio Access Network
rAPP	Microservice on the non-RT RIC in ORAN
RIC	RAN Intelligent Controller
RSRP	Reference Signal Received Power
SBA	Service-Based Architecture
SDN	Software-Defined Networking
SMF	Session Management Function
UC	Use Case
UPF	User Plane Function
xAPP	Microservice on the near-RT RIC in ORAN
5G, 6G	Fifth, Sixth Generation of cellular networks
5G&B	5G and Beyond

Executive Summary

A final report is given on the systems and methods developed for the automation of the AI@EDGE connect-compute platform. The reported achievements largely correspond to the project overall Objective 3 on a general-purpose network automation framework.

The Network and Service Automation Platform (NSAP) hosts the network automation functionality within the overall AI@EDGE system architecture. The NSAP makes use of the concept of closed-loop network intelligence, where the goal is to relieve the human operator as far as possible from the need to take manual action to operate the system. The data-driven AI/ML-based automation functions are deployed as Artificial Intelligence Functions (AIFs) in the connect-compute platform. An update is given on the architecture of the NSAP together with a summary of its components.

A Data Collection Pipeline is defined with the purpose to efficiently collect, store, and distribute platform monitoring data for the data-driven automation functions. Its interfaces and components are described with a focus on data reusability.

The Model Manager supports the life-cycle management of AI/ML-based automation functions. It deals with data feature extraction and model training, the storage of trained models, and monitoring of the performance of deployed models. If the performance is becoming unacceptable, the Model Manager initiates an update of the model, possibly initiating retraining.

A set of data-driven methods and algorithms for network automation are described and evaluated. For each of the methods presented, a comprehensive overview is provided, including details about the methodology employed, techniques utilized, results achieved, and the analysis and conclusions obtained from the results, to understand its effectiveness and applications. The methods are grouped into three classes as follows.

(i) Methods optimising the deployment of distributed AI: These address various aspects of placement and migration of services at the edge, including optimization of application placement to avoid resource fragmentation by minimizing the number of active nodes, automating edge computing node allocation to minimize human intervention, placement of AIFs making use of federated learning and hardware acceleration, and a general method for split inference and compression of ML models to reduce computational costs and processing time.

(ii) Methods for predicting/forecasting future performance: One method examines how predictive radio awareness can be achieved through advanced machine learning techniques, thereby enhancing system performance. Another method focusses on how machine learning can predict key performance indicators and user mobility to optimize service migration and radio handoffs in a mobile edge computing context.

(iii) Methods for monitoring and preparing data: A monitoring method explore a framework for anomaly detection tailored specifically to the challenges posed by highly distributed 5G&B and edge computing environments. To address these complex requirements, a Federated Learning based framework for anomaly detection approach is devised, ideally suited for processing data dispersed across multiple devices and locations within the network. Another method makes use of simulation to create datasets for training machine learning-based algorithms. These datasets have been thoroughly constructed to act as benchmark tools for the evaluation and comparison of data-driven algorithms.

Prototype implementations of two NSAP components, the Intelligent Orchestrator Component and the Data Collection Pipeline are described. The components, together forming a functional NSAP, are integrated and evaluated in the context of the project testbed implementing the connect-compute platform.

The data-driven methods and algorithms do not work without the appropriate data sources. Relevant identified data sources include container-level data, physical server-level data, eNodeB and UE level

data, WiFi AP-level data, application server-level data and network performance real-time monitoring data. An update is provided compared to the previous deliverable.

1. Introduction

This deliverable is the final report on the systems and methods developed in the project for the automation of the AI@EDGE connect-compute platform. It covers the conceptual and technical design of the project's architecture for closed-loop network automation, with scalable collection and distribution of platform data, and support for AI model life-cycle management. It furthermore contains the description and evaluation of automation methods using data-driven AI/ML models for performance prediction, resource management, and orchestration of application components.

There are several scalability aspects in this context relating to using data-driven AI/ML methods for network automation and providing support for distributed applications making use of AI/ML, for example, involving federated learning. Data-driven methods for network automation make use of status and performance data from the network platform, some of which are high-volume real-time data. The project develops the concept of a data collection pipeline processing framework to support such data sources in a scalable fashion, including the deployment of pipeline processing close to the source and sharing the pipeline output between several consumers. The life-cycle management (LCM) aspects are also important from a scalability perspective, where training and retraining ML models potentially can consume a lot of resources, both in time and energy.

The methods and algorithms using data-driven AI/ML models are here grouped into three classes, as follows.

i. **Methods for optimising the deployment of distributed AI**: These are data-driven algorithms for placement and migration of services at the edge. Placement algorithms based on distributed and deep reinforcement learning, a method for distribution of worker nodes for federated learning, and a method for distributing deep neural networks over an edge computing infrastructure are presented. (Section 4.)

ii. **Methods for predicting or forecasting future performance**: These are methods that from comprehensive monitoring of network and compute platform performance, including radio-level metrics, predict or forecast performance in the near future. These methods form a basis for proactive data-driven network management services, including service deployment and migration. (Section 5.)

iii. **Methods for monitoring and preparing data**: These are methods that deal with monitoring of the network and compute platform, and methods for creating and managing data for data-driven algorithms. A federated learning-based method for anomaly detection, and a method for generating data sets that enable the training of ML-based algorithms are presented. (Section 6.)

The achievements reported in this deliverable are largely corresponding to the project's overall Objective 3 on designing and implementing a general-purpose network automation framework. This objective is detailed in the four work-package objectives: (i) defining the technical requirements specific to the methods and systems mechanisms planned for development relative to the overall AI@EDGE concepts and architecture; (ii) defining the systems-oriented methods for developing reusable data models, scalable data propagation and information-exchange, container deployment strategies and processes, as well as adaptive security approaches implementing data protection and service isolation; (iii) designing the learning methods for secure and resilient infrastructure management and performance prediction, largely based on AI/ML for the purpose of supporting automated and adaptive deployment processes and resource allocation in line with application specific requirements; and, (iv) realizing the tools facilitating development and operation for stakeholder users (application developers, service providers/network operators) at the application level.

Deliverable 3.1 [D3.1], presented work on four automation methods that are not further covered in this deliverable. For completeness, they are briefly summarised below.

Optimising resource scheduling with machine learning (D3.1, Section 4.1.4) [PerezRamirez23]: A common task for network management is to schedule a set of resources over time considering network specific constraints. In this work, a machine-learning approach that leverages graph neural networks (GNNs) and Transformers is used to solve the combinatorial optimization problem of generating interrogation schedules for IoT networks with battery-free sensors. The proposed scheduler is trained on small networks using the optimal schedules obtained from a constraint optimizer. Without the need to be retrained, it can be deployed to compute schedules for previously unseen networks, while outperforming a carefully crafted heuristic by up to 50%.

Predicting general and application specific user performance from the mobile edge perspective (D3.1, Section 4.1.3) [Behravesh22]: Many applications can benefit from prediction of future performance to be able to adapt its service. In this work, a performance prediction algorithm using a deep neural network is developed that enable a DASH video streaming application to predict the bitrate of the next video segment, enabling pre-fetching to reduce access delay, and thus overall service latency.

Data augmentation to increase the robustness of learning (D3.1, Section 4.2.5): In many cases, there is a lack of enough data for training ML-based methods representing a breadth of scenarios or cases. In this work, data augmentation is explored as a strategy to not only expand but also enrich an existing dataset, thereby enhancing its suitability for training machine learning algorithms. The aim is to increase the robustness of machine learning models and mitigate issues of overfitting and underfitting, and thereby enhancing generalization performance across a broader range of scenarios or cases.

Autonomous systems operations and continuous learning processes (D3.1, Section 4.2.3): In the context of the project with data-driven methods for network automation in access networks with edge computing support, there is a need for continuous training of the methods to cope with dynamically changing conditions. In this work, an approach for enabling autonomous system operations with continuous learning based on federated learning is described.

The rest of the document is organised as follows. An update of the project's architecture for network automation, the Network and Service Automation Platform (NSAP), is described in Section 2. The NSAP is a framework for closed-loop automated network management that provides an environment for data-driven, intelligent, methods that support decision making. This section contributes to objectives (i) and (ii). Section 3 describes the data collection pipeline and ML model manager, two important functions of the NSAP, contributing to objectives (ii) and (iv). The next three sections describe and evaluate methods for network automation, mainly contributing to objective (iii). First, in Section 4, methods for optimising the deployment of distributed AI are covered. These are ML-based algorithms for placement of services at the edge. Section 5 contains methods for predicting or forecasting future performance, in particular for radio-level performance metrics. Then, Section 6 deals with methods for monitoring and preparing data. Section 7 describes the prototype implementation of the NSAP platform and mainly contributes to objective (iv). Section 8 provides an update of the platform data sources described in Deliverable 3.1 [D3.1] contributing to objective (ii). Finally, conclusions are provided in Section 9.

2. AI@EDGE Network and Service Automation Platform

In this section, an updated architecture for the AI@EDGE Network and Service Automation Platform (NSAP) and its relation to the AI@EDGE system architecture is presented. This provides data-driven intelligent automated network management and serves as a basis for the rest of the work in this deliverable. The architecture is based on an initial version presented in D3.1 [D3.1] and shown here in Figure 1, with updates based on further work in the project shown in Figure 2, including additional components such as the data collection pipeline, and interfaces.

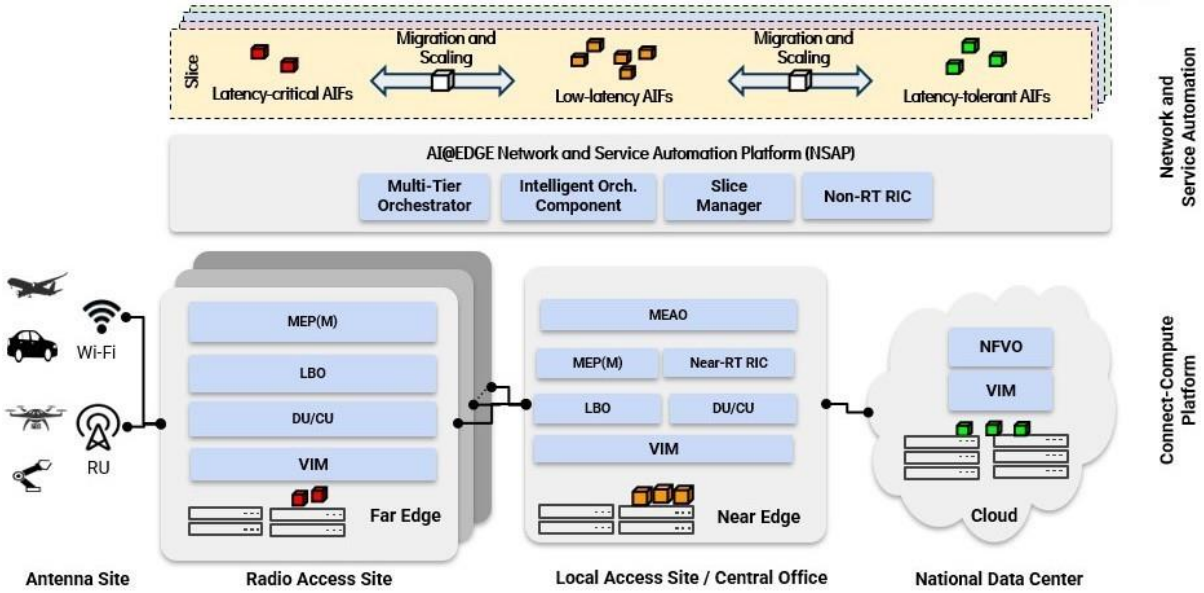


Figure 1 Initial AI@EDGE architecture

The NSAP consists of the components located at the Cloud that provide the means to properly control and optimize the performance of the MEC and 5G Systems deployed at the Near and Far Edges and also components for collecting and processing data at the near and far edge in order to do data driven network automation. The components of the NSAP and its detailed architecture are shown in Figure 2 and explained in Section 2.1. The details of the Near/Far Edge and Cloud systems are mainly in the scope of the AI@EDGE Connect Compute Platform defined in WP4. In this section, the main focus will be on describing the NSAP architecture.

The rest of this section is structured as follows: In Section 2.1, the NSAP architecture is described in more detail, with architecture components described in Sections 2.1.1, 2.1.2, and 2.1.32.1.3.

2.1 Architecture overview

Figure 2 shows NSAP architecture and its relevant components. The key components of the architecture that are central to the NSAP are:

- **Multi-Tier Orchestrator (MTO)** - handles coordination of the different orchestrators to orchestrate and deploy AI Functions (AIFs) at different levels of the MEC system.
- **Intelligent Orchestration Component (IOC)** - provides support to the MTO to make intelligent data driven decision on the orchestration of AIFs.
- The **Non-RT RIC** is the key element of the O-RAN's Service Management and Orchestration (SMO) component to enable control-loop automations at the 5G RAN. This also includes the **Operation and Maintenance (OAM)** components.

- Finally, the **Slice Manager** intelligently manages MEC and 5G resources to create multi-tier slices.

The MTO, IOC, and Non-RT RIC and their relation to the NSAP architecture are described further in this section. The Slice manager can create and deploy multi-tier network slices involving MEC and 5G resources with defined SLAs. It is further described in project Deliverable 2.3, Section 3.1.3 [D2.3]. Intelligent management of multi-RAT slices by the non-RT RIC and the rApps are experimentally validated in project Deliverable 4.2, Section 4.6 [D4.2], according to defined SLAs. In an integrated scenario, the Slice Manager would be responsible for managing the slices and associated SLAs, and announcing them to the rApps.

In addition to these components, the Data Collection Pipeline (DCP) system that is required to enable scalable and trustworthy information exchange across computing overlays. The DCP is used by the Model Manager, Data Processor, and Data Collector components to maintain and update machine learning models for the AI automation tasks and provides support for collecting and processing data for further use by AIFs. In particular, the data collector and the processor components may be running in the near and far edge in order to collect platform data from the MEC and RAN systems. Data from the DCP can be fed directly into AIFs for real-time use or can be stored at the Data Saver for long-term storage. The task of the data collector in NSAP is to provide data for network automation; while application data may also be used by AIFs, the collection of this is not the task of the NSAP but is the responsibility of the applications, with an interface towards the storage and processing of the DCP being provided. The DCP is described in more detail in Section 3.1.

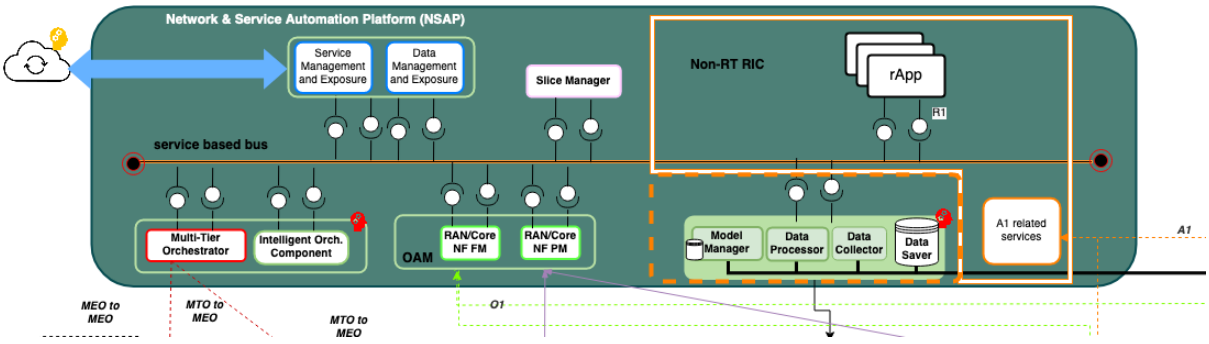


Figure 2 NSAP - Network & Service Automation Platform

A design decision was made to define the NSAP as a service-based architecture with a common service-based bus enabling the production and consumption of different network services. This type of architecture provides an efficient way assembling applications from reusable services and allows developers to build applications more quickly.

The following architectural aspects of the AI@EDGE platform are to be noted:

- **Service-based architecture (SBA)** is the approach for interfacing components themselves and may be used by the MTO to orchestrate and manage the AIF deployment and management.
- **Data handling reference points** are defined between logical components in a way that any impact to the live network is localized to the source of data, so the extensions of the existing protocols may be used to minimize the impact to the live networks.
- **Domain federation** - the deployment of AIFs may span different domains, e.g., distribute the components across different domains (e.g., Core, RAN and Edge).

- **Third-party integration** - enables the third-party solution providers to integrate with AI@EDGE platform providing AIF application, the complete pipeline or some of the components of the pipelines, such as Model component.

The data driven components of the NSAP like the AI/ML model manager, Data processor and Data collector may or not be implemented as part of the Non-RT RIC. This means that the AI@EDGE architecture fits well for integration with O-RAN systems but can also be implemented independently. The relationship between the AI@EDGE and O-RAN architectures is explored further in Section 2.1.32.1.3.

2.1.1 Multi-Tier Orchestrator

The Multi-Tier Orchestrator (MTO) is the Network and Service Automation Platform (NSAP) entry point for the operations related to the onboarding and instantiation of MEC AIFs/apps. The MTO enables communication with different types of orchestrators across multiple southbound clients, such as: (i) the MEC Orchestrators (MEOs) located at the Near Edge of each MEC System; and (ii) the cloud based NFV Orchestrators. Depending on the incoming request, the MTO can issue different orchestration operations for the applications lifecycle (e.g., deployment, migration, termination, etc.) to nodes located on the near edge or far edge of the network.

The MTO represents an optional module of the architecture when it comes to deployment. In other words, the modular design of both software artifacts enables the deployment of a single MEO if required. This could be useful, for instance, in isolated environments where a single orchestrator manages the whole networked infrastructure.

The operation of the MTO can be observed in Figure 3. When the MTO receives a deployment, or migration request from the NSAP, the requirements of the specific applications are shared with the Intelligent Orchestration Component (IOC) to seek the most suitable solution to the orchestration request received. Notice that this procedure is possible thanks to the metric aggregation performed at the NSAP, which congregates platform and infrastructure telemetry data coming from various MEOs at the NSAP. The infrastructure telemetry data is a sub-functionality of the DCP related to infrastructure data and provides an overall view of the status of the various resources available at the various MEC systems. The decision of selecting the best candidate location for allocating and deploying the AIF/app is computed by the IOC, which belongs to the decision-making module of the network automation closed loop stated in D3.1 [D3.1], and it is described in more detail in the next subsection. Once the decision is provided by the IOC, the MTO first checks if the descriptor is already on the AIF database, and otherwise, it onboards it accordingly. In the case that the values in the descriptor have varied, they are also updated in the corresponding descriptor at the database. Then, the MTO proceeds with the deployment (or migration) process by communicating with the selected MEC Orchestrator required. This operation is performed through the message broker implementing the MTO-to-MEO interface. Finally, if the request received corresponds to a deletion process, then the MTO directly communicates through the MTO-MEO interfaces with the MEC system where the application is instantiated to proceed to kill it an erase it from the system.

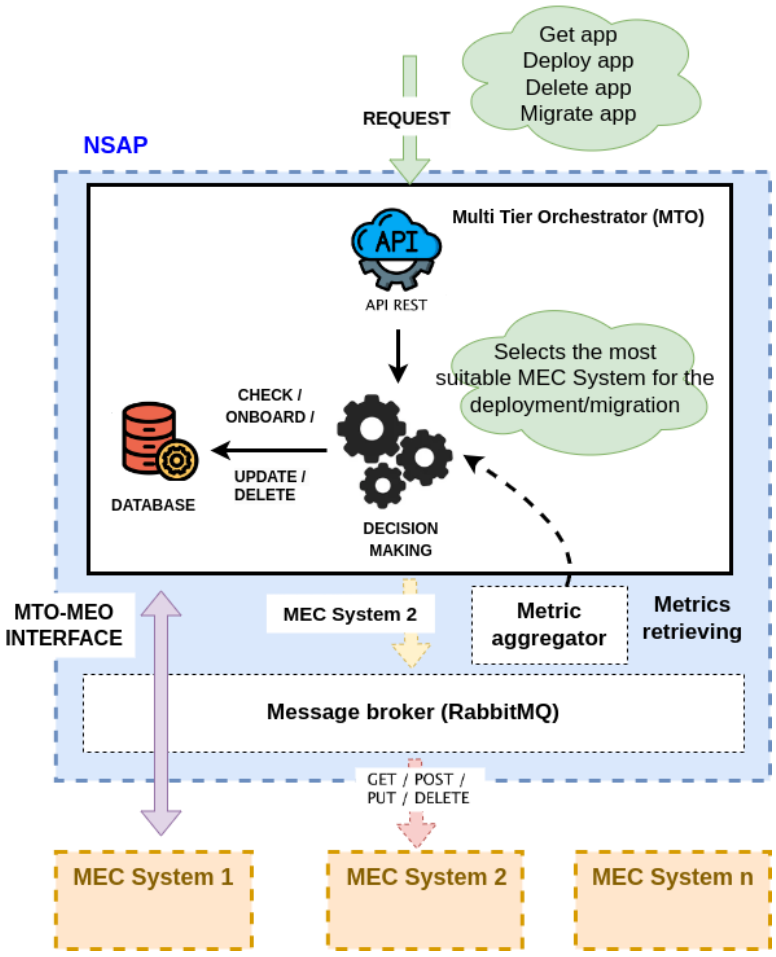


Figure 3 Operational structure of the Multi-Tier Orchestrator

2.1.2 Intelligent Orchestration Component

The Intelligent Orchestration Component (IOC) provides decision support for the deployment and migration of AIFs during runtime. It leverages fault, security, and resource management functionalities and is situated at the NSAP level, where it interacts with the MTO, facilitating intelligent decisions for optimal AIF placement and migrations.

The IOC obtains platform and infrastructure telemetry data from the Connect Compute Platform (CCP) through the Data Collection Pipeline (DCP). This data, including specific metrics from AIFs and MEC system-level metrics for resource utilization (CPU, RAM, HW acceleration availability), serves as the basis for intelligent, data-driven orchestration decisions. The telemetry may also encompass historical data from long-term storage, enabling the IOC to predict resource availability using internal AI/ML models.

When the IOC receives a request from the MTO, it incorporates critical information from the AIF descriptor. This includes computational requirements (RAM, CPU, Disk, HW acceleration, among others), AIF-specific metrics (throughput, latency, etc.), and potentially information on data locality.

All collected metrics are aggregated to a MEC system level, providing a global view of each MEC System, allowing the IOC to make optimal MEC system deployment decisions, while the task of determining the node for AIF placement is delegated to the IARM at the CCP level. In this sense, the IARM offers a more granular and real-time view of each node's availability.

The IOC continually monitors different MEC systems for anomalies or faults in specific AIFs and nodes. Upon detection, the IOC suggests migrating certain AIFs to the MTO to balance the load between nodes, ensuring compliance with the minimum requirements specified in the descriptor.

The internal intelligence of the IOC manifests in various forms, including AI/ML models developed in the project to address placement and runtime issues in AIFs. Examples of such models are anomaly detection, federated learning models, and AIF placement models, which are elaborated upon in Section 4.

2.1.3 NSAP Relation to O-RAN Non-Real-Time RAN Intelligent Controller

The NSAP in AI@EDGE implements a subset of functionalities of O-RAN's SMO layer [OranArch]. In particular, as shown in Figure 2, the Non-Real-Time RIC is the key element to implement intelligent closed-loop automations related to the 5G System at the NSAP level, and to manage the 5G System automations present at the Connect-Compute Platform level.

At the NSAP/SMO level, intelligent network automation is done by the rAPPs, which have access to non-RT RIC functionalities (e.g., A1 interface, internal data…) and SMO functionalities (e.g., O1, O2, external data…) through the R1 interface. The R1 interface, which is still under definition in O-RAN [OranR1], is a service-based interface with Service Management and Exposure (SME) and Data Management and Exposure (DME) capabilities in order to facilitate interconnection among rAPPs (as service consumers and/or producers). O-RAN is also considering R1 interface as a way to expose rAPPs/non-RT RIC services to external entities (e.g., RAN analytics to 3GPP NFs like NWDAF). This is aligned with the proposed NSAP architecture shown in Figure 2, where NSAP services can be exposed to non-RT RIC and rAPPs through the service-based bus, and vice versa.

The Non-Real-Time RIC implements the A1 interface termination, which provides the necessary methods to manage the closed-loop automations performed at the available Near-RT RICs and xAPPs. These methods or functions comprehend the management of policies (A1-P interface), the exposure of enrichment information available at the NSAP level (A1-EI interface) and the management of machine learning workflows (A1-ML interface). For instance, using these three interfaces, rAPPs can fine-tunning the operations of the xAPPs through the defined policies, expose data from external components to them (e.g., from core network NFs or application servers) or exchange AI/ML model parameters in federated learning architectures, respectively.

O-RAN has started the definition of the AI/ML support [OranAIML], where topics like how to provide AI/ML workflow services (e.g., model management, model training, model inference, data preparation…) at the non-RT RIC are still under discussion (e.g, rApps only, platform only, or hybrid rApp/platform approaches). In any case, the support for multiple types of machine learning as supervised, unsupervised, reinforcement, and federated learning is assured and the architecture is flexible enough to enable diverse deployment scenarios: e.g., offline/online training at the SMO or the non-RT RIC (rAPPs), inference at the non-RT RIC (rAPPs), online training and inference at near-RT RIC (xAPPs). In addition, the R1 interface enables the chaining of modular models, since rAPPS can consume and produce data and prediction services. In conclusion, AI@EDGE is well aligned with O-RAN's view on AI/ML, where the AIF concept could be applied to rAPPs/xAPPs and the proposed data pipeline (i.e., model manager, data collector, data processor) could implement some of the AI/ML workflows services proposed by O-RAN. In addition, it would facilitate the data and analytics sharing between 5G and MEC systems.

2.2 Artificial Intelligence Functions - AIF Conceptual Model

Artificial Intelligence Functions, AIFs, is a central concept of the AI@EDGE project. The project has defined a conceptual model in project Deliverable 2.3 [D2.3, Section 2.3.2]. For completeness of this deliverable the definition is summarised here.

The purpose of AIFs is to encapsulate an application's AI logic so that it can be dynamically distributed and provisioned for execution at the edge in the project's MEC-based connect-compute platform. The AIF is thus a kind of cloud native function (CNF) with additional requirements on the infrastructure to execute its AI model. The conceptual AIF model with its interfaces is illustrated in Figure 4.

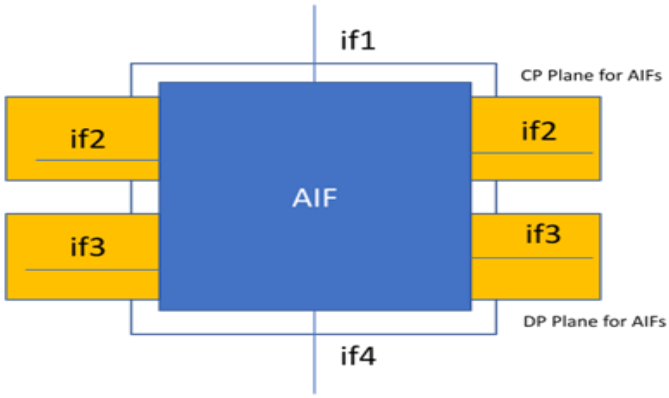


Figure 4 The AIF conceptual model

The AIF's interfaces are the following [D2.3]:

- if1, is the northbound interface used for (re)configuring the AIFs. Its semantics is defined by the specific component the AIFs is implementing. For example, this interface could be used to set threshold for any of the observed values (CPU usage, signal strength, number-of-connected-users, etc) below which a management event should be triggered.
- if2, is the ML control plane interface used to exchange model parameters and support distributed and/or federated learning scenarios.
- if3, is the ML data plane interface used to exchange the data on which the ML model is applied. For example, in the case of a load balancing application this could be the stream of radio channel quality (e.g., RSRP/RSRQ) measurement originating from the RAN.
- if4, is the ML southbound interface used to (re)configure another entity. This could be for example an external SD-RAN controller. The format of the interface is the one exposed by the external entity.

3. Data Collection Pipeline and Model Manager

This chapter details the architectural aspects of the Data Collection Pipeline (DCP) and the new component proposed in the AI@EDGE called Model Manager. We start by describing the components of the proposed DCP system and then we introduce some Model Managers workflows model update scenario and how it uses the DCP components to help the Model Manager to update the models efficiently. In this Section, we also present views on how the AI@EDGE architecture can support different types of learning. In the next section, a discussion about LCM management using the Model Manager is proposed. We conclude this chapter by describing the Model Manager experiments on the machine learning models updates. As the AI@EDGE architecture proposes different choices for models' updates (e.g., continuously update, periodic update, and drift detection update), it is the role of Model Manager to detect what it is the best way to update the model based on the model's performance and the network resource consumptions.

3.1 Data Collection Pipeline Conceptual Interfaces and Implementation

3.1.1 Conceptual Data Collection Pipeline Interfaces

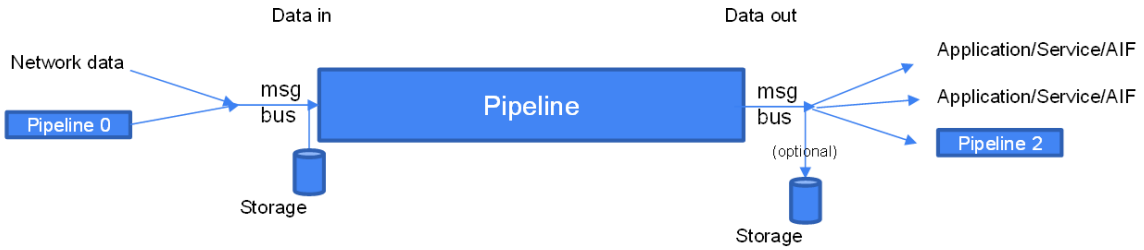


Figure 5 Data Collection Pipeline Interface

In addition to the updated DCP architecture described in the previous section, new DCP interfaces have also been defined, as shown in Figure 5. While the DCP was part of the original NSAP and AI@EDGE architecture, it was not clear how it interacted with all other components, so a clearer definition on its components and how they are structured, in particular with focus on data reusability, is needed. The figure shows a pipeline and its input and output interfaces. Data can enter the pipeline through the following possible ways:

- *Network data.* This is data that comes from different network services in the NSAP and can be provided either through the message bus interface or directly as raw data.
- *Pipeline 0*. As multiple pipelines can be connected to enable more complex services and data processing to be created, input to the pipeline can also be received through the message bus interface from another DCP.
- *Storage*. Finally, data can be entered into the DCP from longer term storage where it has been stored by other DCPs or AIFs for future reuse.

After internal processing of incoming data in the pipeline, it is sent onto the output message bus interface. Here it can be consumed by one or more consumers:

- One or more application services/AIFs may consume the output data to complete their tasks. The pub/sub nature of the message bus allows multiple consumers to reuse the same data if desired.
- *Pipeline 2*. As for the input interfaces, the output of the DCP can also be fed into one or more additional pipelines.

- *Storage*. If the DCP produces data that need to be stored for future use by other AIFs or DCPs, the output data can also optionally be stored in longer term storage.

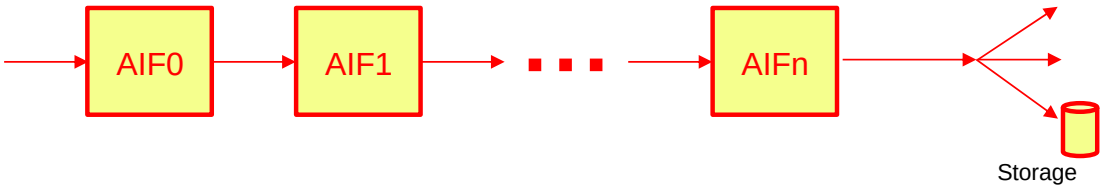


Figure 6 Internal structure of data pipeline, with AIF interfaces

In Figure 6, the internal structure of the DCPs is shown. There we can see that each DCP consists of one or more AIF (potentially also other non-AI cloud native functions) that perform the data processing of the pipeline. The if3 interface in the AIF is used for data input and output.

3.2 Model Manager Pipeline

The Proposed Model Manager Pipeline is illustrated in Figure 7. The blocks that represent the proposed architecture are as follows: the data collector, the data processor, the model manager, the data saver, and the model saver. Each of these blocks and their relationship are described in the following figure.

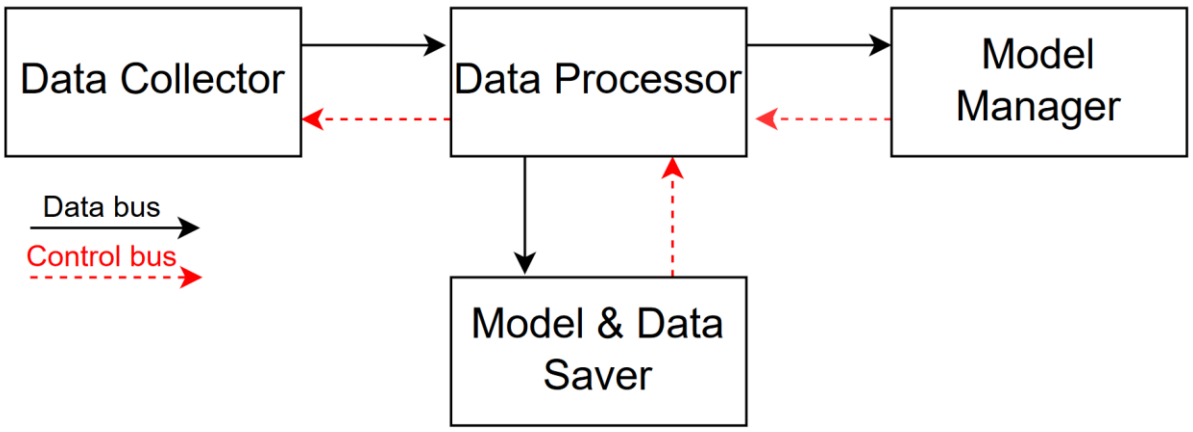


Figure 7 Proposed Data Pipeline Architecture.

Data Collector: it is the component responsible for the collection of the data where the data is extracted from the source (e.g., UE). The Data Collector can also be represented in the *"data in"* description of the Figure 5 where the data will be collected.

Data Processor: This component is responsible for processing the data and extracting the necessary features to deliver it to the correct applications. It is responsible for the "Map to Model" step where it maps the current data to the model that is requiring that data. "Clean" when all unnecessary or corrupted data that may remain are removed. The "Randomize" step is where the data is randomized to improve security. The "Anonymize" step where it adds another level of security to that data by anonymizing it. "Extract the feature" where all the necessary features of the data are extracted. "Metadata enhance" where we improve the metadata of the data to improve the management of that data.

Data and Model saver: This step is where the data is going to be saved. If the data is in the "raw" format, then the data should be stored in the "raw data" repository. However, if the data is processed and the features extracted, then this kind of data should be stored in another repository called the

"Insight & data processed" repository. The "Data saver" is composed of four tasks: "Authenticate" and "Metadata refine" which is the same type of processing as described in the "Data Processor" component and this is done just to improve the security and the quality of the data that will be stored.

After the security improvement, the data is compressed at the "Compress" step, and it is stored at the "Store" step. It is important to highlight that the "Data Saver" and the "Model Processor" can be done at the same time in parallel. Furthermore, the Data and Model Save can store machine learning models and, before saving it, the step called "Process" is done where it is to make sure the metadata is valid. This is done in the "Metadata lookup". Then, the next step is to compress the model (done in the "compress" step) and save the model (done in the "save" step). Concluding, the Data and Model Saver can make use of the data storage component illustrated in Figure 4 of Section 3.1.

Model Manager: This component is responsible for the evaluations of the performance of the model. After the deployment of a model in the AI@EDGE networks, it is necessary to monitor its performance to check if it is necessary to update the model. This monitoring is important as the characteristics of the data can change over time and this will result in a decrease in the performance of the model. It is the role of the model manager to detect this decrease in performance and to suggest the necessary actions to be taken.

3.2.1 AI@EDGE Architecture support for different types of ML AI

The AI@EDGE architecture supports multiple types of ML AI that include supervised, semi-supervised, unsupervised, reinforcement, and federated learning, respectively.

Supervised learning, where there is access to a dataset and all data are annotated, is the most straightforward learning type supported by the AI@EDGE architecture. The NSAP, Near Edge, and Far Edge domains can deploy models based on supervised learning. For the NSAP case, the structured and labelled data are stored in the Data Processor's database, that contains data from all other domains. Similarly, for the Near Edge, the labelled data are stored in the Data Processor's database. However, unlike the NSAP database, the Near Edge database only considers data from the domains controlled by the MEC orchestrator. Having different databases, it is possible to instantiate AIFs that consider the supervised cases in different domains. This means that it is possible to have an AIF in the NSAP that may oversees the end-to-end services. This approach also supports both semi-supervised and unsupervised learning since they consume data from the repositories, even though the data is not fully labelled. For the Far Edge, it can deploy models even considering the restricted network resources. However, a decision of how to split the processing and data collector should be manager as Far Edge contains limited resources consumptions.

3.2.1.1 Reinforcement learning

Reinforcement learning, where there is a feedback mechanism on the actions taken to improve model performance, data could be readily available from the Data Producer and the Data and Model database. Here we consider three domains capable of deploying AIFs with reinforcement learning: NSAP, Near Edge, and Far Edge. For the NSAP and Near Edge two modes of reinforcement learning are possible, namely live and mirror. For the live case, the data that comes from the environment is fed directly to the Agent to make decisions of what actions to take. This enables ultra-low latency services. For the mirrored case, where a model can be trained with live data but without taking actions that impact the network, data can be augmented with the Data Processor's database, ensuring that a model could be replaced in case the performance of the original model downgrades. For the Far Edge, since resources are scarce, only the live reinforcement method is supported with data coming from the Model Consumer. The model on Far Edge could be changed using the Data and Model pipeline. Like with supervised learning, data needs to be stored in the Data Processor's database. However, there are cases where privacy and security are a must, where federated learning is required.

3.2.1.2 Federated learning

For federated learning, where data are local and only parts of the model are sent to a centralized location, we consider distributed store too. This means that both the NSAP and the Near Edge domains support federated learning. For traditional federated learning, where the model updates are sent to a central repository, the Model Processor in the NSAP receives the updates from all Model Processors in the Far Edges. This ensures data remains decentralized, usually stored in the Near Edge's Data Processor's database. It is also possible to deploy federated learning AIFs without the NSAP. In this case, the Near Edge Model Processor is responsible for aggregating the model updates coming from other Edges. This ensures that data under the AI@EDGE architecture remains local and private. Furthermore, other learning types such as assisted learning, where data and models remain local, are also supported.

3.3 LCM workflow illustration

Until now, the discussion about data pipeline should not only be able to handle the aspects regarding the data such as how to bring the data to the correct application, how to preprocess the data and extract the necessary features, and how to store the data efficiently. Going one step further, this section will describe how the proposed Model Manager pipeline will handle aspects of the machine learning life cycle management such as monitoring a machine learning model that is running to be sure that its performance is not under the expected and, if this is not the case, how to re-train the model and how to do it efficiently.

It is known that future mobile networks will be capable of handling and storing dozens, hundreds or even thousands of machine learning models. Based on this, the idea of a repository to store all these models becomes important. AI@EDGE aims to use this concept and bring innovations and answer questions such as *"how to select the best model for deployment?"*.

To contextualize this question, consider an example of a latency prediction model that was trained offline using a dataset with a distribution probability and it was deployed and running. In this scenario, there is an entity called a "model manager" that is responsible for monitoring the model performance to make sure that the model is capable of correcting and predicting future latency. However, at some time step, the statistics of the current data change in such a way that the model is not able to predict the latency correctly. This change in the probability distribution will result in a decrease in the model performance. If the decrease in performance is critical, then it is necessary to re-train the model to match with the new probability distribution. Based on this scenario, the question "how can we effectively re-train our model or replace our model?". Considering the approach of our Model Manager pipeline, three different scenarios or different ways that we can re-train our models are available:

- Scenario 1: we can collect the data and re-train a model from scratch.
- Scenario 2: we can reuse the data that we collected before and train our model.
- Scenario 3: we can replace our model with a compatible one.

Doing an analysis of the time of each scenario, Scenario 1 is the slowest scenario because the time that is necessary to collect all the necessary data to compose a training dataset can be prohibitive and, in addition to this, the training time from scratch can vary from minutes and hours until days depending on the application and on the dataset size. Scenario 2 is an update of Scenario 1 because there is no need to collect the data on runtime because a database that can represent the statistics of the current data in an acceptable way can be reused and, therefore, the only time that is necessary is the training time. Concluding, Scenario 3 is the fastest one because instead of collecting the data and training the model from scratch, AI@EDGE uses a model repository and stores a model that was used in the past and when a re-training is needed, the AI@EDGE Model Manager pipeline just replaces the current model by a compatible one.

In summary, Table 1 illustrates all the necessary times for each scenario. All the Scenarios are described with more details in the following subsections.

Table 1 Necessary time to re-train a model for each Scenario

Scenario ID	Collecting data	Training the model
Scenario 1	Included	Included
Scenario 2	Not Included	Included
Scenario 3	Not included	Not Included

3.3.1 Scenario 1: Collecting the data and training the model

This scenario is the simplest one and it is described in Figure 8. In this scenario, an Inference AIF is running a model, and this model is being monitored by the Model Manager and this is done when the Model Manager receives the PREDICTION, PERFORMANCE, CONFIDENCE, and the key to access the model metadata. After receiving this information, the Model Manager compares the CONFIDENCE, and the PREDICTION with the information described in the Model Metadata (the model metadata is accessed using the *modelRef* key). After this comparison, the Model Manager realize it needs to update the model. The first step before updating a model is to check in the Model database if there is a compatible model that can be used for model replacement. As, in this scenario, this is not the case, then the Model Database returns *"noCompatibleModel()"* as output. As there is no compatible model, the current model needs to be trained again. However, before starting the training, it is worthy to check in the database if there is a previously stored database that is similar enough with the current data. However, in this scenario there is no database previously stored. Then, it is necessary to collect enough data to be used as a training dataset. All this is done by the training AIF which is deployed by the MEO / MTO. The Model Manager pipeline in this scenario is responsible for collect the data, extract its features, and deliver to the training AIF for the training to be executed. Also, the new trained data should be stored in the dataset to check if it will be used in the future. This is one of the ways that the Model Manager pipeline provides data reusability (instead of collecting again the data, the Model Manager pipeline should be smart enough to check previously in the database if there a good dataset to be used in the training).

After the model training step, the new updated model is stored in the model database for future check. For example, if in the future the statistics of the data changes back to the one that was collected now, instead of collecting the data again, the Model Manager pipeline just uses the data that was collected now.

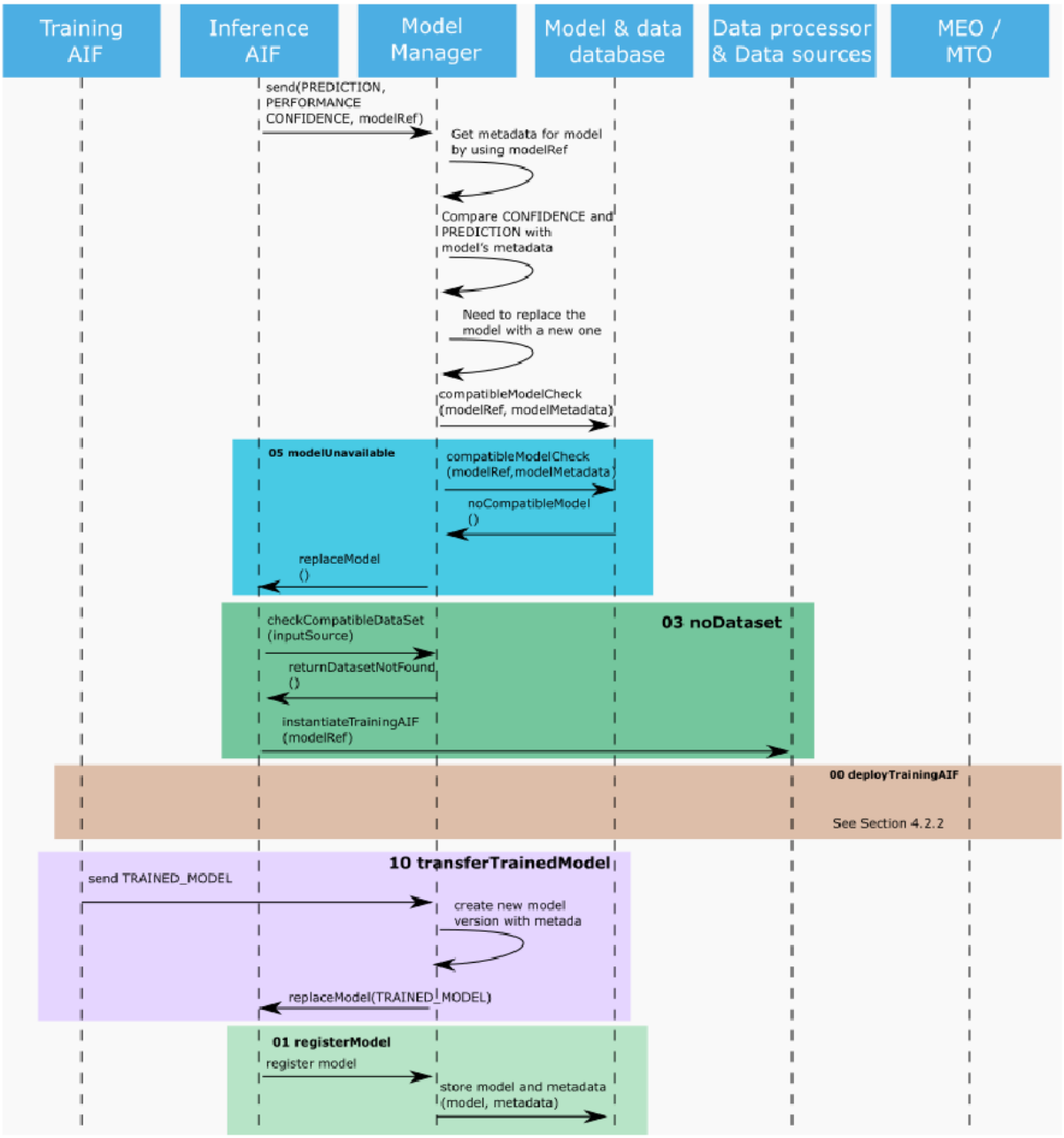


Figure 8 Scenario 1 workflow: compatible model not found

3.3.2 Scenario 2: There is compatible data but no compatible model

As mentioned in the previous scenario, the new data was collected and stored in the database. Now, consider a future timestamp where completely different data is being consumed by the network. If the data statistics change to the same or similar statistics to the data that was collected in the previous scenario, the Model Manager pipeline can take advantage and use this dataset instead of collecting all the data again. This scenario is illustrated in Figure 9. The workflow starts in a similar way with the Model Manager checking the model's performance and comparing it with the information stored in the model metadata. Then, the Model Manager realizes that the model needs to be updated. As usual, the first step that the Model Manager takes is to check in the Model database to see if there is a compatible model for replacement. As, in this example, there is no compatible model, then the Model Manager

checks in the database if there is a compatible database that has similar characteristics with the current data that is being consumed by the model. As this data was previously stored (consider, for illustration purposes, that the data was collected in the previous scenario). Then the Model Manager asks for a training dataset to be deployed and requires the training data to train a new model using the database that was collected. This is an improvement on the previous scenario as no duplicated data was collected. After the training, the new model is registered and stored in the Model database. Here, the Model Manager pipeline was acting directly to improve the MLOps performance as the non-duplicated data was used. However, the Model Manager can go one step further and take advantage of all these models that are being stored over time. This advantage is described in the next scenario.

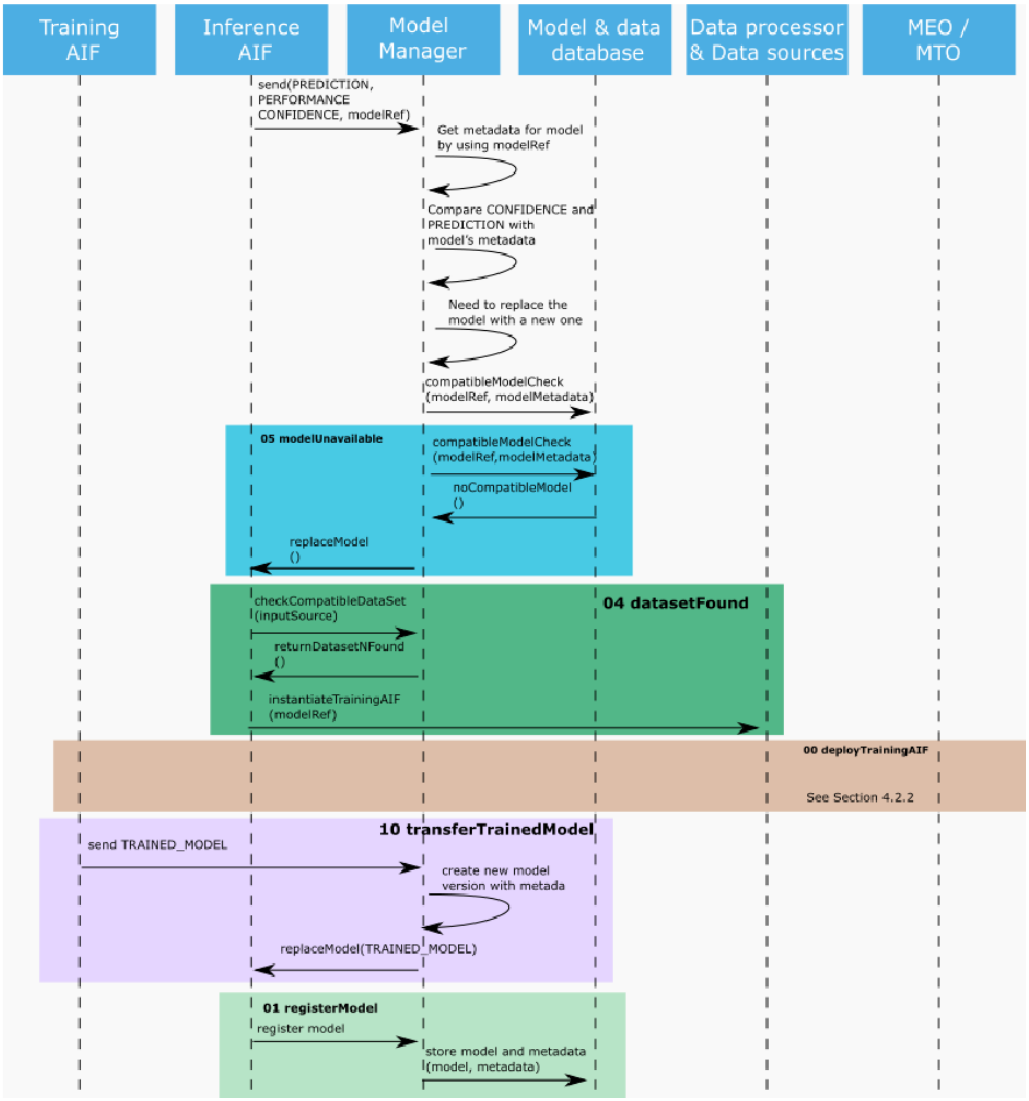


Figure 9 Scenario 2 compatible model

3.3.3 Scenario 3: There is a compatible model

This scenario is an improvement over the scenario 2. The idea is that, as many model are being stored over time, we could end up with a considerable amount of model and the Model Manager can take advantage of all these previously stored model by applying the model reusability feature that is described in this example which starts in the same way as the previous one by the Model Manager collecting all the necessary information of the monitored model to check if the model performance is

acceptable. If this is not the case, then the model needs to be updated. The first step here to update the model is to check in the Model Database if there is a compatible model that can be reused again. In this scenario, a compatible model is found. Then, the role of the Model Manager is just to ask to the inference AIF to update the model.

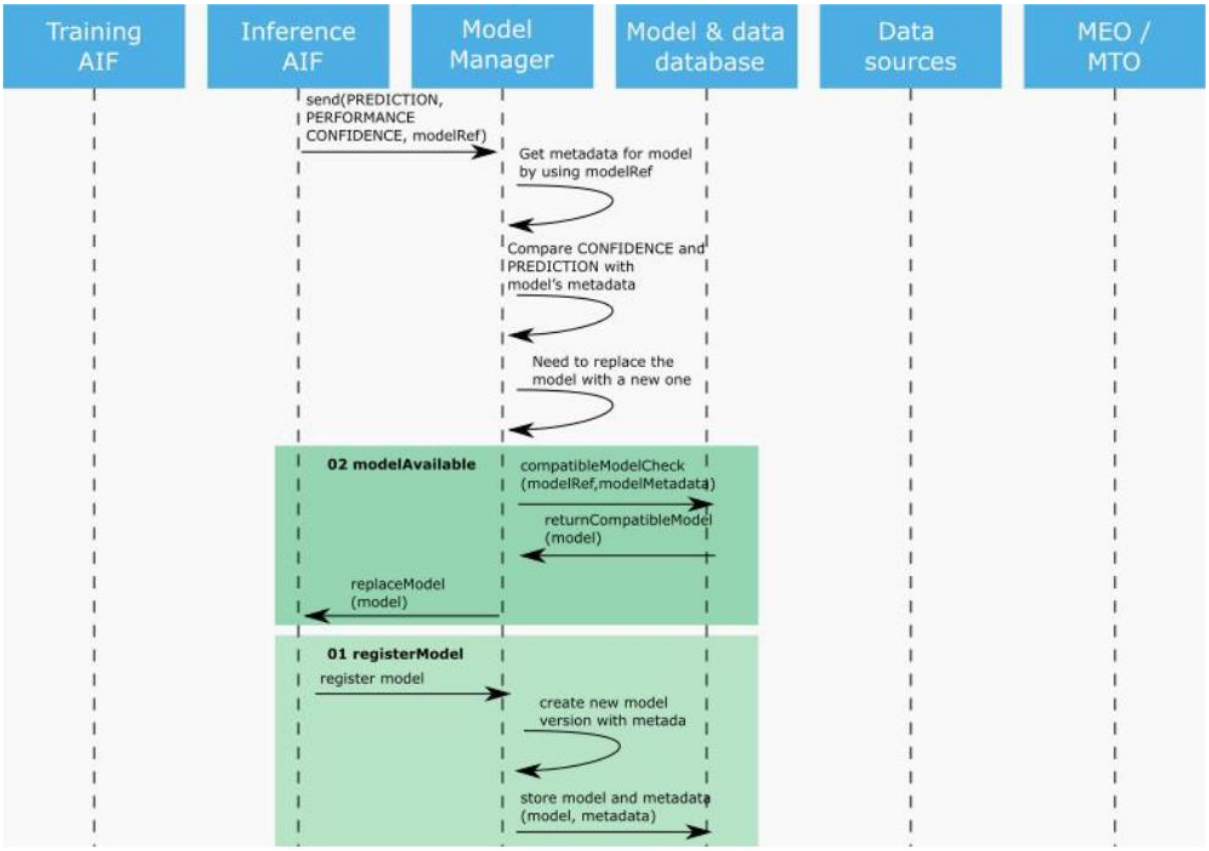


Figure 10 Scenario 3 workflow

Verifying which compatible model for model replacement requires a smart way to organize the model database for fast model selection. How to organize the models in the model database is one of the contributions of the Model Manager paradigm. Furthermore, these three ways to update a model provides an overview of how the Model Manager pipeline can be used to improve the Model MLOps by giving faster time response using the data reusability and model reusability feature. However, there are other possible ways in which a model can be updated (e.g., update a model continuously, periodic or based on event). All these workflows can be found in D2.3 [D2.3].

3.4 Model Manager experiments

In this Section, we describe how the Model Manager updates the model based on the available updating method (continuously, periodic, or based on drift). We are not considering the model replacement as discussed before, just how the Model Manager evaluates its performance in updating the models continuously, periodic or based on drift detection. We would like to start by highlighting the Model Manager logical view. The Model Manager can manage models that are in different physical domains such as the cloud, near edge, and the far edge. Each physical domain has different network resources availability, and the Model Manager should take this into consideration when deciding how to proceed.

As the Model Manager will work based on the Machine Learning models' performance and the available network resources, it is not a trivial problem to check if the Model Manager is doing his role

properly. As an example, imagine the following scenario: a stakeholder has deployed a model $\mathbb{A}$ with accuracy performance of 90%. Then, the Model Manager checks that it is possible to retrain the model to increase the accuracy to 95%, however, to achieve that new accuracy, it is necessary to train the model for three days using four different GPUs. Now, the question is: is it worthy to spend all these network resources to increase the model accuracy by 5%? To start answering this question, a first step is to present all the metrics that the Model Manager considers, and this dialogue is presented in the next subsection.

3.4.1 Model Manager Metrics and Performance Evaluation Designer experiments

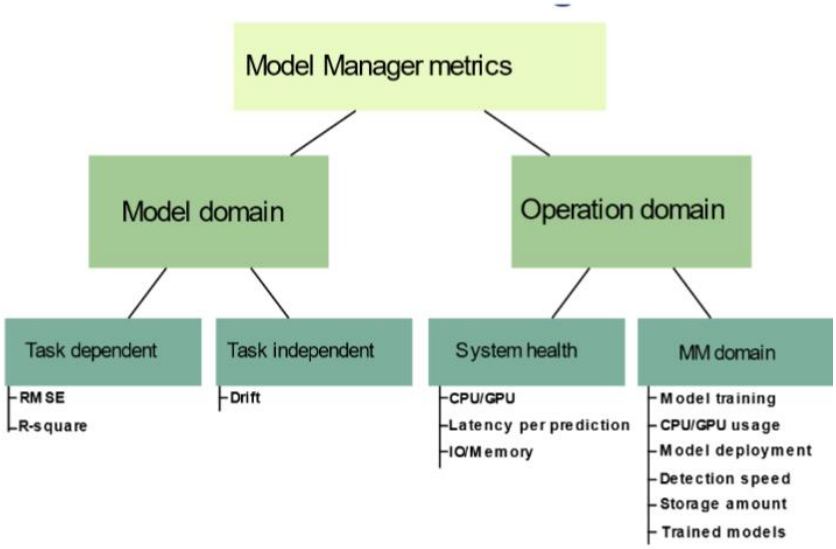


Figure 11 Model Manager metrics

It is important to understand how to measure the Model Manager performance to check if its role is being achieved. However, this is not an easy task as different variables need to be considered when evaluating the Model Manager performance. As an example, Figure 11 described all the variable considered to evaluate the Model Manager performance. It is possible to see that the variables are from different domains (e.g., Root Square Mean Error (RSME) and CPU/GPU usage). The scope of this section is to have a discussion regarding how to use these different variables to compute the Model Manager performance.

First, if we would like to evaluate the Model Manager performance, it is easier if this performance can be measured in a quantitatively way. If we can assign a single number $x \in \mathbb{R}$. We can call this number x as **score**. The idea is that a higher score value represents a better Model Manager performance, and a small score would represent a poor Model Manager performance. After defining the score value, we need to understand how we can use all the different metrics presented in Figure 11. For this paper, we have considered some metrics such as task dependence and independence metrics, system health metrics and model manager domain metrics. Each of them will be described in the next subsection.

3.4.1.1 Task Dependent Metrics

These metrics are related to the specific task that the model is working on. For example, for object detection the metrics can be mean Average Precision (mAP), for classification can be accuracy. As in this paper we are considering the user case of mean end-to-end latency prediction, the Root Square Mean Error (RSME) metrics is chosen.

To define the RSME, we need to consider the model that will be deployed in the network to predict the future latency. This model will receive an input vector of N samples $X \in \mathbb{R}^N$ and outputs the vector $\hat{Y} \in \mathbb{R}^M$ containing the next M latencies predictions. To use the RSME metrics, we need to compare the $\hat{Y} \in \mathbb{R}^M$ predictions with the correct latencies $Y \in \mathbb{R}^M$ values. We can define the RSME used in this paper as

$$RSME = \frac{1}{m}\sqrt{\sum_{i=0}^{m-1}(\hat{y}_i - y_i)^2}$$

3.4.1.2 Task independent metrics

All the metrics that are not directly related to the specific task are classified as Task independent metrics. In this work, we are only considering the drift metric. As defined in [Lu19], drift (or concept drift) is defined when the statistics properties of a domain changes over time. More precisely, given a time $[0, t]$, the input vector X and a label vector Y. A concept drift occurs at time $t+1$ when $F_{0,\hat{t}}(X,Y) \neq F_{t+1,\infty}(X,Y)$ where $F_{0,\hat{t}}(X,Y)$ is the probability distribution of the input vector X and output vector Y starting from time $t = 0$ until $t = \hat{t}$ and is the same probability distribution but from time $t = \hat{t} + 1$ and ∞. To detect this drift change, one possible approach is to use the Kullback-Leibler (KL) divergence [Cover91]. This metric is defined as

$$D_{KL}(P \,||\, Q) = \sum_{x=0}^{n-1} P(x) log\,(\frac{P(x)}{Q(x)})$$

where $D_{KL}(P \,||\, Q)$ is the KL divergence between the P and Q distributions and both distributions have x samples. The $D_{KL}(P \,||\, Q)$ value is proportional to the divergence between the P and Q distributions. To define if a drift is detected, we need to define a threshold λ. To detect the concept drift, we need to compare KL divergence between the current distribution P and the distribution used to train the model which is defined as Q. After computing the KL divergence, we just compare this value with the threshold λ. If this value is greater than λ, then the concept drift is detected. Otherwise, the concept drift is not detected. As the concept drift depends on the λ value, a high value for λ will result in a non-sensible concept drift algorithm in which the concept drift will only be detected even there is a huge different between the distributions. Otherwise, a small value for λ will result in a sensible concept drift detection where a small change in the distributions will result in a trigger to inform the Model Manager that the concept drift is detected.

3.4.1.3 System Health Metrics

System Health Metrics are the ones related to the system resources consumption. In this work, we selected the GPU/CPU usage which is defined as the $0 \leq x \leq 1 \; \forall \; x \in \mathbb{R}$ where 1 means that all the CPU/GPU resources are being used. Otherwise, 0 means that no CPU/GPU resources are being used. Latency per prediction is the time that is necessary to predict the future latency $\hat{Y}$, this value is given in milliseconds. The last System Health Metrics is the IO/Memory which is defined as how much the RAM memory is used and this value is described in the same way as the GPU/CPU usage.

3.4.1.4 Model Manager domain Metrics

The most abstract measures are the ones classified as Model Manager domain metrics. These metrics are defined as the one directly related to the Model Manager functionality. The following metrics are described as follows:

- Model Training: all the resources used to train a model. Includes the CPU/GPU usage, storage resources, memory usage.

- CPU/GPU usage: how many CPU/GPU is necessary to run the Model Manager instance.
- Model deployment: how many resources are being used to deploy a specific model. Can include CPU/GPU usage, storage resources, memory usage.
- Detection speed: how fast the Model Manager detects that it is needed to update the model.
- Storage Amount: how many storages is being used by the Model Manager. The storage includes all the databases with the collected data and the Model repository including all the model managed by the Model Manager.
- Trained Models: how many models are being trained by the Model Manager.

3.4.2 Model Manager Score Definition

After defining all the necessary metrics for the Model Manager, it is time to develop an equation that represents how good the Model Manager performance is. The goal is to include all the metrics described in the previous sections and compute a single value $S_{mm} \in \mathbb{R}$ that describes the Model Manager performance. This is not a trivial question as there are metrics from different domains with different ranges.

The Model Manager Score S_{mm} is defined as

$$S_{mm} = \frac{\sum_{i=0}^{k-1} \alpha_i M_i}{\sum_{j=0}^{l-1} \beta_j Op_j}$$

where M_i is the performance of the i_{th} model (RSME, Accuracy). In this equation, there are k models available; Op_j is the j_{th} operation metric (e.g CPU/GPU Usage, RAM usage) and each metric can be weighted by the β_j. In this equation, there are j different metrics to be computed. Furthermore, α_i and β_i are both hyperparameter and are defined by the user.

The equation above shows that the model performance is proportional to the Model Manager Score value. However, as in our experiments we are using the latency prediction use case and the chosen model metric is RSME, we need update to adapt the equation above for the following reasons. First, we decided that a high Model Manager Score means a good performance. However, a high RSME means a bad model performance. As we need to minimize the RSME as much as possible, we need to find a way to map a low SME value to a high Model Manager Score. The chosen way is to update the equation above by putting the model performance in the denominator as

$$\frac{1}{\sum_{i=0}^{k-1} \alpha_i M_i \times \sum_{j=0}^{l-1} \beta_j Op_j}$$

For simplification purposes we are not considering the effects of α and β in our experiments. Then, we set them to 1 resulting in

$$\frac{1}{\sum_{i=0}^{k-1} M_i \times \sum_{j=0}^{l-1} Op_j}$$

Another aspect we should consider in our experiment is that we are using a single model. Therefore, $k = 1$. This will be resulting in the final equation to compute S_{mm} used in this paper.

$$S_{mm} = \frac{1}{M_0 \sum_{j=0}^{l-1} Op_j}$$

3.4.3 Model Manager: experiments description

In this section, all the details regarding the experiments design are described. We have divided this section in two parts: Models Updates Parameters and Dataset Generation. The latter will explain all the details regarding when a model will be updated, and the latter will explain how the dataset used in this work was created.

3.4.3.1 Models Updates Parameters

For the experiment design to evaluate the Model Manager performance, we choose three types of model updates: Continuous, Periodic, and Drift (please see Figure 12). Continuous update means that the model will be updated after every prediction with x different samples, periodic means that we will update the model at each x sections and by drift update we mean that we will update the model only when a drift is detected.

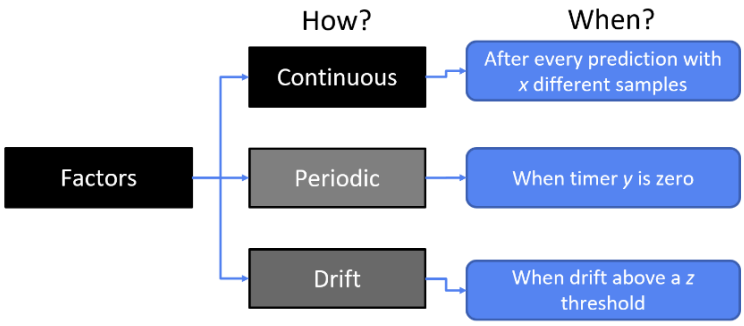


Figure 12 Model updates description

As described in Figure 12 by continuous training we update that the model at each 1, 10, 100, 1000, and 100000 samples. Updating model at each 1 sample gives more fast time response with the drawback of the computational cost as we need to constantly train the model every time that a new sample appears. On the other extreme, updating the model only after 10000 samples requires less computational cost time but the model update latency (i.e., the necessary time to update the model) is higher. In our experiment, to compute the Model Manager score S_{mm} we take into consideration both elements: the model accuracy and the computational cost necessary to update the model.

Updating a model after drift means that constantly we will check if a data drift was occurred. To check the data drift, we use the Kullback-Leibert difference and, if the divergence is higher than the threshold λ, then the drift is triggered, and the model is updated.

The periodic update means that, at every x samples, the model is updated. As described in Table 2 in the periodic wise approach, we update the model at each 60, 30, and 600 seconds. This update is independent of drift or the model performance metrics which means that the model will be updated even if the model accuracy remains the same.

Table 2 Experiments Parameters

Factor	Levels				
Number of samples for continuous	1	10	100	1000	10000
Drift Threshold	0.125	0.25	0.5	1.0	
Periodicity (sec)	60s	300s		600s	

Time horizon	12 hours	24 hours
Data drift	10%	30%

3.4.4 Dataset generation

To generate all the datasets used in this experiment, we used the pareto distribution. This probability distribution will tell if a UE will be active or idle. We can manipulate the UE by changing the pareto distribution parameters. We have used 30 different Ues $UE = [ue_1, ue_2, ue_3, I, ue_{30}]^T$. At each time stamp, each UE_i will be active if a random value x generated by the pareto distribution is higher than $\beta = 0$. When the UE is active, it starts to send messages of XX Kbytes to the cloud. Each UE_i has a corresponding latency l_i between the UEs and the cloud. Let us define the latency vector $L = [l_1, l_2, I \ldots, l_{30}] \in \mathbb{R}^{15}$ where each latency l_i corresponds to the latency between the ue_i and the cloud. Finally, to generate the dataset, we define the average latency L_{avg} as

$$L_{avg} = \frac{1}{30}\sum_{i=1}^{30} l_i$$

Figure 13 shows all the datasets generated in this work. Starting by the training dataset, each value is the average latency. The test dataset has an average drift of 10% (e.g, the average is 10% higher than the average of the training dataset), and the second test dataset has an average drift of 30% (e.g., the average is 30% higher than the average of the training dataset)

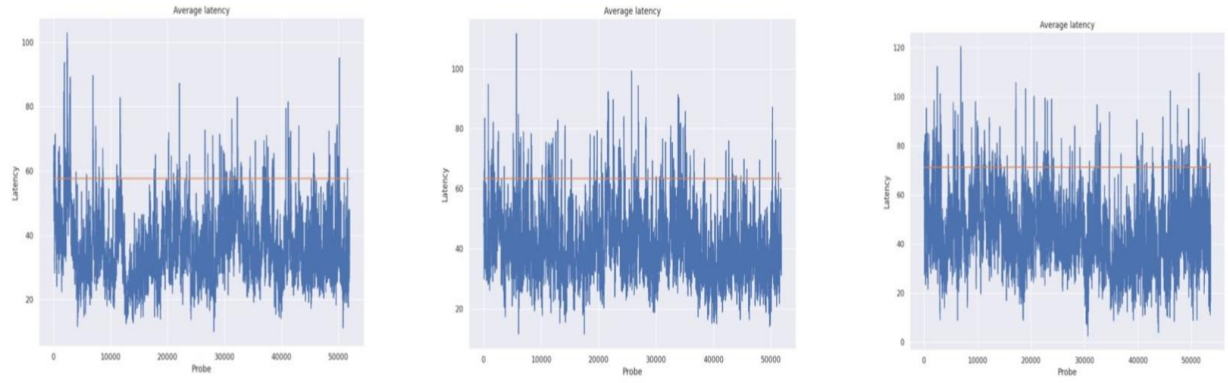


Figure 13 Training dataset (left), test dataset with 10% of drift (middle) and test dataset with 30% of drift (right)

3.4.5 Model Description

The model used in this work is a prediction model. Figure 14 describes the model and, as it is possible to see, it is composed of six different layers: the input layer which received the last 15 latency measures $X = [x_{t-15}, x_{t-14}, x_{tI}, \ldots, x_t]^T \in \mathbb{R}^{15}$ where t denotes the timestamp value containing the 15 last latency measures. The next layer is the LSTM layer that will contains 20 units per layer. To improve the model generation performance, a dropout layer with a dropout rate of 0.2 is added between each LSTM layer. Furthermore, a fully connected dense layer with X neurons is added after the second LSTM layer. The output of the model outputs the next 15 latencies measures and it is a vector $\hat{Y} = [\hat{y}_{t+1}, \hat{y}_{t+2}, I, \ldots, \hat{y}_{t+15}]^T \in \mathbb{R}^{15}$. The learning rate of the training is 0.001 and we have used the Adam optimizer. To compute the model performance, we need to collect the real measures values $Y = [y_{t+1}, y_{t+2} I, \ldots, y_{t+15}]^T \in \mathbb{R}^{15}$ and we compare with the predicted latency $\hat{Y}$ using the RSME equation**:**

$$RSME = \frac{1}{15}\sqrt{\sum_{t=1}^{15}(\hat{y}_t - y_t)^2}$$

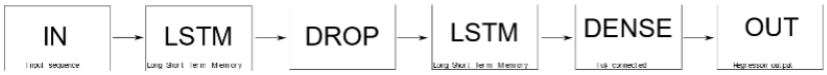


Figure 14 Prediction model description

3.4.6 Model Manager results

The results presented in this Section can be divided into two types: 12h results and 24h results. 12h results means that the data was generated and collected during an interval of 12h. The same applies to 24h results. In the following, the results for 12h is described.

3.4.6.1 12h results

Figure 15 (a) shows the Model Manager score S_{mm} for each type of model updating. Higher values mean a better score and implies a better Model Manager performance. The x-axis value represents a type of model updating and it is composed of three fields: the first one it is the update name, the second it is how many samples we are considering and the last one it is about the drift value in percentage. As an example, the first element of the x-axis is called drift_1000_30 which is the result when the model updates by drift detection, and the dataset has a drift of 30%.

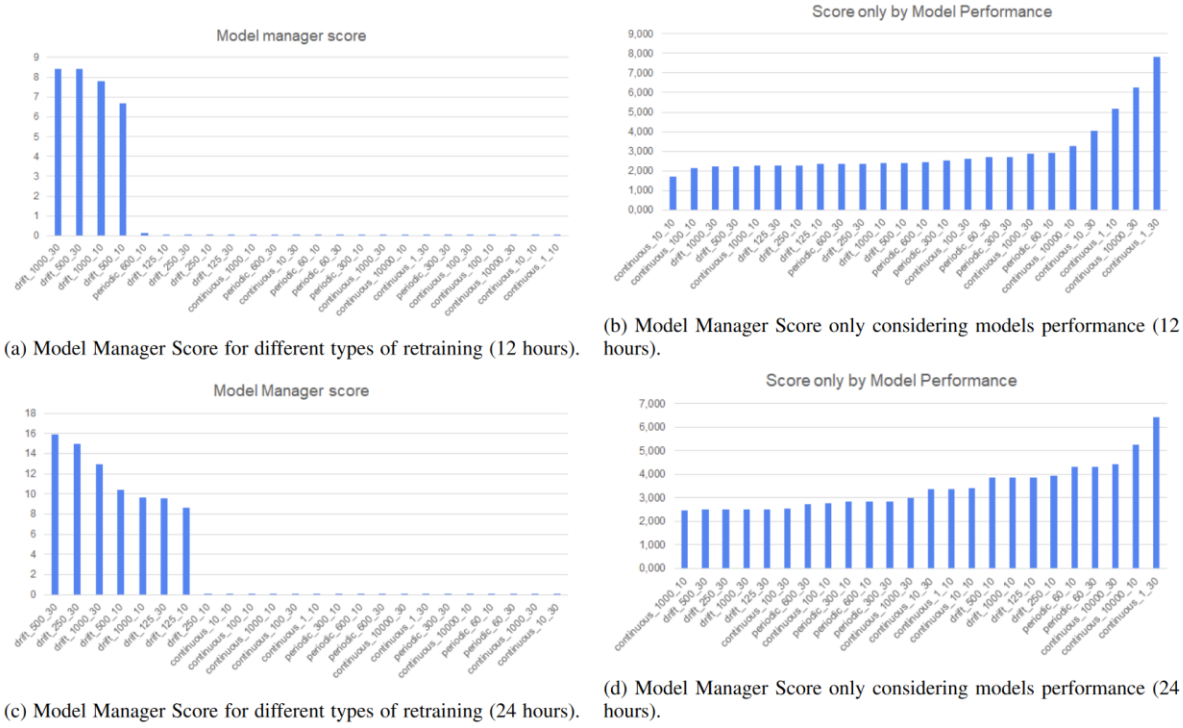
As all the results are ordered, therefore, the most left value represents the better one. In this experiment, the best way to update the Model considering the model performance maximization and network resource consumption minimization it by drift when the drift of the test data it is 30%.

To continue, the most important results come from some deeper analysis. The first one it is that all the drift updating methods present a very high Model Manager Score S_{mm} if compared with the remaining ones. This fact leads to the conclusion that, in this experiment, the best way to update the model is by data drift detection. It does not matter if the drift is by 10%, 30% or 50%, the best choice to update the model is to apply the drift detection algorithm. However, this does not mean that the drift leads to a better model performance but, when we consider the model performance maximization and the network resource minimization, updating the model by drift it is the best choice. This discussion becomes clear when we check only the model performance evaluation as described upper right-side plot of Figure 15 (b). In this picture, the continuous learning gives the best Model Manager performance. This means that, using continuous approach, the model will lead to better results. The drawback is that the computational cost necessary to implement the continuous approach it is too high resulting in a poor overall performance.

3.4.6.2 24h results

The last part of the results deals with the data that was generated and collected during a time interval of 24h. The purposes to increase the duration of the experiment it is to have a better generalization view of the results. As discussed in the 12h results subsection, ***Error! Unknown switch argument.*** (c) represent the same Model Manager Score S_{mm} for different ways to update the model. The results are like the 12h experiment scenario with the drift methods having the best Score S_{mm} when compared to other models. The main difference it is the order of the drift positions. Furthermore, Figure 15 (d) represents the Model Manager results when only the Model Performance it is considered, and the results are like the 12h experiments with the continuous approach giving the best model performance results. The drawback of this approach is that the computational resources that it is needed it is too high.

The results presented in both experiments highlights that it is not a trivial question to decide which update model method we should use in our network. Some of them can give better results such as the continuous learning approach but depending on the network resources availability, maybe it is not going to be the best decision. For example, update the model by drift detection can represents a good trade-off between the model performance and the network resource consumption. The Model Manager score S_{mm} metrics it is effort to quantify the model performance and the network resource consumptions in a single number to easily inform the user how good or how bad the Model Manager is doing its job.



(a) Model Manager Score for different types of retraining (12 hours).

(b) Model Manager Score only considering models performance (12 hours).

(c) Model Manager Score for different types of retraining (24 hours).

(d) Model Manager Score only considering models performance (24 hours).

Figure 15 Model Manager results

3.4.7 Discussion and conclusion

This work presented a component called Model Manager which has the role of monitor the model's performance and decide the best approach to update the model. Furthermore, to quantify the Model Manager performance, a score S_{mm} metrics was proposed. This score considers the model performance and network resource consumption, and it is a single number $x \in \mathbb{R}$ that quantifies how good the Model Manager is doing its duties.

The first step is to check if the Model Manager Score metric is different for each scenario. This is important as the Model Manager decision will be driven by this metric. This can be checked by looking into the results with deeper details. To test the Model Manager performance, a set of experiments was designed. First, the experiment was divided into two classes: 12h and 24h which represents the time interval in with the data was generated and collected. For each experiment, we tested different ways to update the model (e.g., by drift detection, use continuous learning, and periodically). Furthermore, we used three different test datasets: one with 10% of drift, the second with 30% of drift and the last one with 50% of drift. We compute the Score S_{mm} for each different scenario and the results shows that updating the model using drift detection approach gives better performance than the other ones. However, this does not mean that the drift gives the best model performance results. If we consider only the model performance results, the continuous approach it is the best choice to update the model. However, the network consumption to implement this method it is too high, so the overall performance

is smaller than the drift which presents a best trade-off between model performance maximization and network resource consumption minimization.

Concluding, the main impact of this work is that the proposed Score metric is different for each type of model update (e.g., continuous, drift-based, periodic) which means that the Model Manager can make a different in how to decide what is the best approach to update the model. If, for some reason, the Score was the same for every updating method, then the Model Manager would not be able to manage the network resources and models performance as, for the Model Manager perspective, every model would give the same performance and it will, for example, choose the updating method with provides the best accuracy. Usually, the continuous approach is the best performance-wise choice. However, as it is known, the resource consumption is considerable specifically if we are dealing with the domains with restricted resources such as the Edge or Far Edge domain.

For future works, we propose to continue the Model Manager evaluation in different scenarios, to test considering a higher number of models being deployed at the same time and investigate how the Model Manager should handle different learnings methods such as Federated Learning and Reinforcement Learning. Furthermore, how the Model Manager decisions will affect the network performance and accuracy.

4. Methods for optimizing the deployment of distributed AI

MEC plays a vital role in beyond 5G networks as it is expected to accommodate different services and applications in geographically distributed scenarios. These networks face the challenge of deploying services with varying service patterns, traffic demands and quality requirements. Inadequate accommodation of these can result in resource bottlenecks, failures in meeting Service Level Agreements (SLA), unnecessary energy consumption, costly infrastructure expenses and inefficient resource utilization. This inefficiency can result in underutilization, resulting in wasted capacity, while overutilization puts excessive strain on hardware components, leading to decreased performance and potential failures. Therefore, optimizing service placement in these networks is crucial to achieve efficient resource utilization, cost-effectiveness, and improved user experience.

On the other hand, ML models are deployed at the edge for inference, because it enables immediate and optimised real-time processing. However, low-capacity edge devices can lead to limitations in the operation of high size ML models. Hence, continuous attention is needed to optimize the allocation of ML models at the edge nodes is important alongside service deployment.

The subsequent subsections present proposals to mitigate the effects of these scenarios. These address various aspects, including optimization of application placement to avoid resource fragmentation by minimizing the number of active nodes, automating edge computing node allocation to minimize human intervention, and split inference and compression of ML model to reduce computational costs and processing time. For each of the methods presented, a comprehensive overview is provided, including details about the methodology employed, techniques utilized, results achieved, and the analysis and conclusions obtained from the results, to understand its effectiveness and applications.

The proposed algorithms have been designed for seamless integration into the AI@EDGE architecture in the form of AIFs or multiple AIFs in the case where the model is split into multiple parts. These come equipped with interfaces to facilitate the acquisition of data and the output of the model results. This integration serves the purpose of overseeing the orchestration of services and the optimization of edge infrastructures. These methods are used to place applications, services, intelligent AIF placement and the compression of activation signals.

4.1 Application placement and active node minimization using Distributed Reinforcement Learning

MEC networks present dynamic conditions, workload fluctuations and changes in device availability. These scenarios become more complex in the case of multiple MEC Systems managed by several orchestrators in geographically distributed topologies. Multiple orchestrators can cater to different types of services or applications within the MEC system network, and they are not required to operate simultaneously. For instance, one orchestrator may focus on latency-critical applications, while another may prioritize bandwidth-intensive services**.**

Thus, service management strategies in these configurations must consider available resources at each device of the different MEC Systems and the demands of different users. Therefore, it is necessary to implement service management algorithm to assure SLA and maximize revenue for providers and users. Specifically, the problem of application placement implies the decision of where to deploy and execute the applications in the network infrastructure. Various considerations arise from this problem: (i) to ensure efficient resource utilization; (ii) to ensure the desired QoS; (iii) to minimize the network congestion; and (iv) to minimize energy consumption.

Within this scope, its crucial to assure minimum energy consumption, due to the presentation of high-power demanding devices and the need of training Artificial Intelligence algorithms in the edge of the network. Unnecessary power consumption can arise when network nodes are needlessly kept active, rather than placing applications on alternative nodes to prevent resource fragmentation.

Consequently, this section presents an approach to efficiently handle the application placement problem in distributed MEC Systems networks managed by different orchestrators, to reduce the energy consumption minimizing the number of actives nodes.

The objective of the ML method is the design of a computing algorithm based on distributed learning that fulfils the application requirements of storage, CPU and RAM optimally, seeking to have as few active nodes as possible. The ML model will ensure resource availability in MEC systems, achieving a trade-off between the economic operational costs and the LA while minimizing the number of active nodes. This optimization objective avoids unnecessary resource fragmentation. On the other hand, in the case of applications with high requirements, it assures whenever possible the applications hosting.

Based on the telemetry information of the network, the model would suggest the most efficient placement for the incoming applications that guarantees said application's requirements. The distributed method aims to solve the problem of application placement, while training a more accurate model by sharing updates between each of them. The model is addressed as a black box that gets as input the application data and nodes availability, and then provides as output the action to be taken.

4.1.1 Implementation

The main objective of this method is to leave as few active nodes as possible, with the aim of minimizing the operational energy costs, while at the same time fulfilling the application requirements. Being able to maximize the number of nodes on standby is primordial in the reduction of the energy consumption, especially over rented infrastructure. As such, the infrastructure responsible would pay less and only for the equipment that are being fully used.

We consider using the Reinforcement Learning method, because meta-heuristic and linear programming methods are high consuming in highly distributed scenarios, as in our cases of several MEC Systems with multiple orchestrators. In the DRL method an agent is exposed to an environment with a pool of actions and consequences generated for taking those actions. The agent is then trained using a method that involves rewarding it when taking the desired action or punish it otherwise. The reward function in this particular case is an exponential function. If the agent chooses to place an application in a domain that completely fulfils the requirements, it receives a high reward value. On the other hand, if the agent chooses a domain that does not fulfil the application requirements, it is penalized with a negative reward. If the choice corresponds to a node with full availability of the requirements, and other nodes are available with occupied resources to host the application, the reward will be 0. The reward function shows the instant value that the model obtains by selecting a specific action to a given state. If we continue generating reward values for each pair of action-states, we obtain a long-term value for each action-state named Q (Hence the name Q-Learning).

Q-Learning method seeks to find a stable Q value after several repetitions of the previously described method of obtaining a reward based on a state-action pair. However, we can take advantage of the prediction qualities of the Neural Networks (NNs) and, instead of repeating the state-action reward method to update the Q value (which is a process computationally exhaustive), we can use the NN to predict an estimated Q value. The learning process starts at the same point, obtaining different Q values from the state-action reward method. After several training cycles are performed, the state-action, reward and future Q values can be re-arranged and used to train the NN as a predictor, this dataset is known as a Replay Buffer. This alternative allows us to use the NN as predictors in a way in which is faster to obtain the Q value and also allows us to predict Q values from unknown state-action pairs. By using a Neural Network as an estimator, the RL method is called Deep Q-Learning. The training process of the predictor generates weights that are related to each of the neurons in the Neural Network. It is possible to separate this process in different, independent models (named Actors), each with different learning weights. The main idea is that the actors can share their own learning weights with the other actors of the network, improving the learning process in a distributed way.

Regarding the Distributed Deep Reinforcement Learning (DDRL) approach, the ApeX [Horgan18] architecture has been selected. ApeX distributes Deep Reinforcement Learning model agents, to perform their learning functions faster and enables them to learn for various environments at the same time. The agents are distributed into the MEC Orchestrators, to learn the different availability characteristics of MEC systems. The main components of the ApeX algorithm are the agents, also called workers or actors in literature, the learner, and the prioritized replay buffer. Each actor generates experiences by interacting with its own environment and sends them to the buffer jointly with the initial priorities of the data. The learner takes samples from the buffer and updates the network parameters and priorities with the information obtained from all the DRL workers acting in the MEC Systems. Subsequently, the workers are updated with the latest network parameters [Horgan18]. This process is repeated until the model reaches a steady state accuracy. This distributed approach includes the concept of distributed prioritized experience replay. It aims to reduce variance and accelerate convergence and benefits from importance sampling and prioritized experience replay. Figure 16 presents ApeX architecture described before.

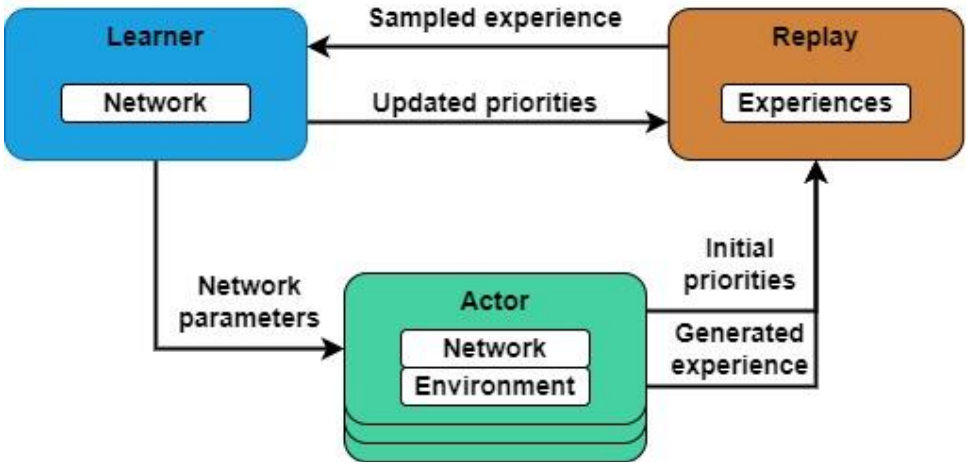


Figure 16 Apex architecture [Horgan18]

The proposed implementation of the distributed model into the current AI@EDGE architecture can be seen in Figure 17, where an DRL actor is running into each of the MEO's of the MEC System and are able to exchange learning weights to the rest of them. Since the proposed ML method is envisioned at MEC system level, the training process assumes that the specific MEC system to which an incoming application request is forwarded, has been already selected at the NSAP. The application request arrives at the NSAP, where the MTO forwards to a specific MEC system the requirements and the availability of the MEC nodes. Each of the models located at MEC orchestrator level receive the aforementioned parameters in terms of CPU, Storage and RAM. The DRL algorithm then selects the most suitable node to perform the placement in each MEC System.

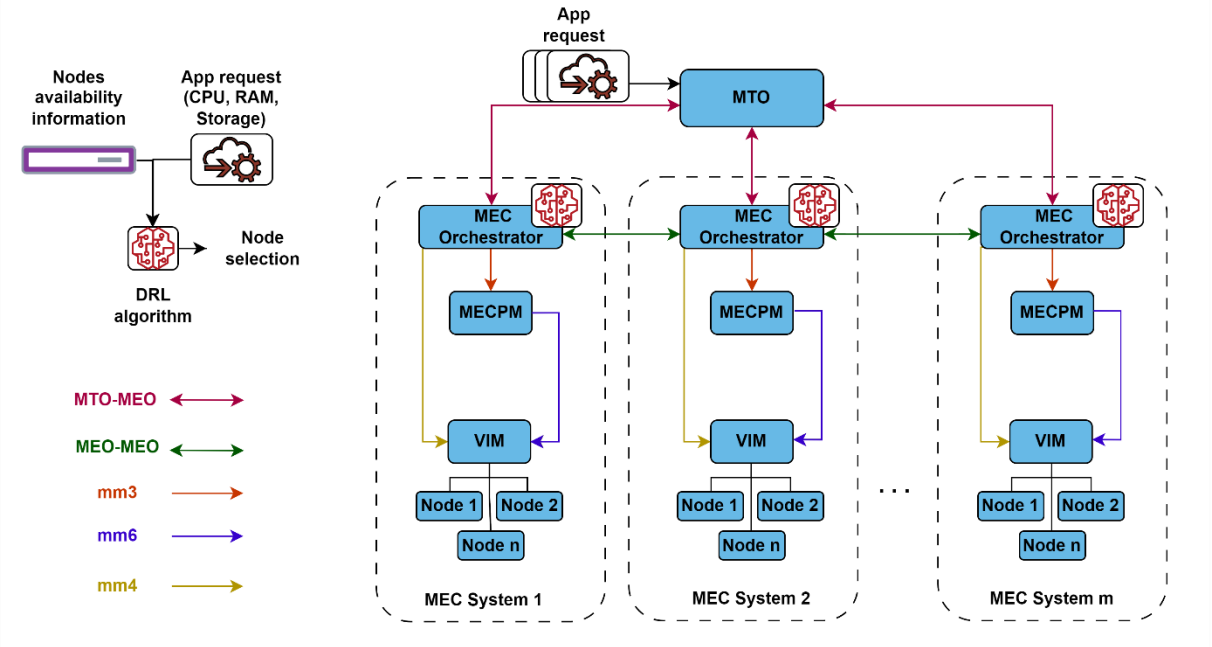


Figure 17 Distributed RL Model integration into the MEC's architecture

4.1.2 Performance Results

We have successfully trained a DDRL model following the previously described process. The DDRL algorithm was trained in a scenario composed of 2 MEC systems including 50 applications arriving in each case in the configurations of 4, 8, 12 and 16 nodes per MEC System. Each application has random values between a selected range of RAM [512MB - 4GB], Storage [512MB - 32GB] and CPU [1vCPU - 4vCPU]. Regarding the nodes availability, all nodes in the same MEC System present equal initial capacities, but they differ between MEC System #1 and MEC System #2. With these values, the DRL algorithm decides over which of the nodes to instantiate the application.

The machine learning model is tested using a simulation scenario developed in Simu5G [Simu5G] [Nardini20]. Simu5G can simulate a 5G data plane network, having as a base the OMNET++ simulator. With this base, a 5G MEC architecture will be used to test the behavior of the ML model. The scenario consists in a cluster with several nodes configurations composing a MEC System, with a UE generating applications continuously and asking the model where to instantiate the applications within a pool of connected domains. The requirements of the applications will be selected randomly within a pool of values that resemble typical and extreme network characteristics. This configuration is replicated with a second MEC system with the same characteristics. As described before, each MEC system can exchange the training weights with the other ones as needed. The complete simulation scenario for the configuration of 2 MEC Systems and 4 nodes per MEC System is shown in detail in Figure 18.

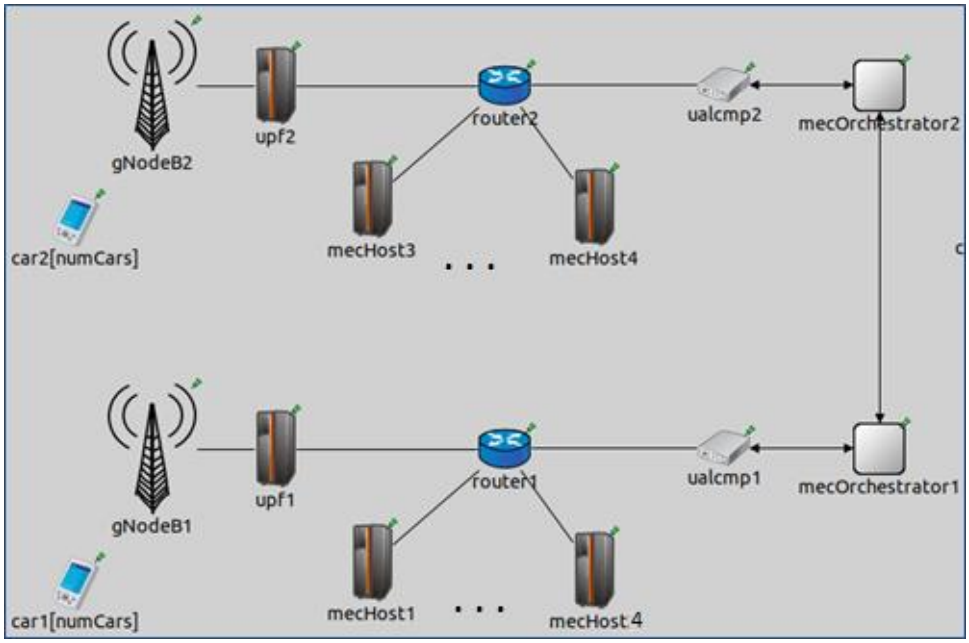


Figure 18 Simulation scenario for distributed placement for load balancing

In order to test the accuracy during the inference process inside a MEC System, we receive 50 applications in each episode and perform 100 episodes. An accuracy value of 1 is given to the decisions in which the hosting corresponds to the node with the maximum value of the reward function of all the possible nodes, otherwise it set to 0. The test results for the configuration of 4, 8, 12 and 16 nodes per MEC system are presented in Figure 19. The mean and average accuracy are maintained above 97% in all the cases. However, we appreciate slightly decrease of accuracy in form of outliers. When an application's resource demand exceeds the availability of the current placement node, is relocated to another node for hosting. However, there are instances where a different application arrives that could have been hosted on the previous node. In such cases, if the new application continues to be placed on the node where the relocation was taking place, and that node is not the least occupied, the accuracy becomes 0. Therefore, there exist iterations with less availability.

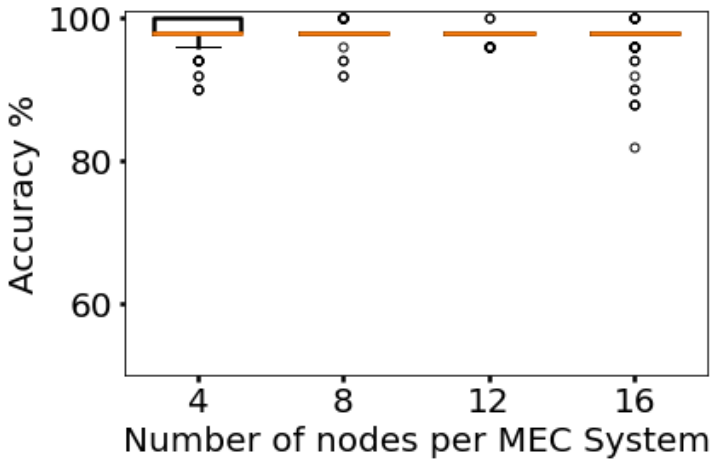
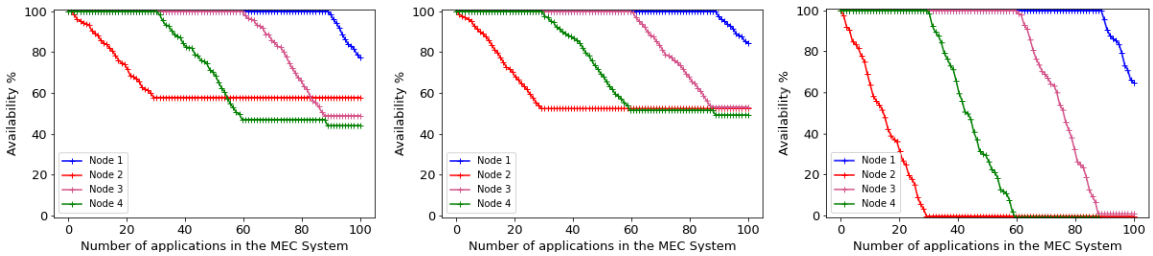


Figure 19 Accuracy for different number of nodes per MEC System

An additional crucial factor to assess is the achievement of the primary goal of minimizing the number of nodes. To evaluate this, we conducted an experiment with a scenario consisting of four nodes and

100 applications. Our aim is to ensure that a node would not be activated until another node reached its maximum capacity, thus maximizing the number of available nodes. Figure 20 presents the availability for storage, RAM and CPU. As more applications are being received, the ML method forces the use of a new node to maintain as many inactive nodes as possible. As the episodes continue, the model guarantees the instantiation of the application. Whenever this scenario is not feasible (because the application requirements are higher than the current node free capacity), the model is forced to activate a new node but then it tries to use the maximum capacity of that node before activating a new one, minimizing the active nodes at any single time. On the other hand, we compare our performed method with an Integer Linear Programming algorithm for the configurations with the different number of nodes, in this case, the obtained performance accuracy is 100% in all the scenarios presented before, However the inference time is at least 5 times longer that in the case of DDRL.



a: Storage availability *b: CPU availability* *c: RAM availability*

Figure 20 Availability for 2 MEC Systems and 4 Nodes per MEC System with 100 entry applications

4.1.3 Conclusions

This work proposes a DDRL model considering the distributed ApeX architecture to solve the application placement problem minimizing the number of active nodes and assuring applications' SLA. We present an incorporation of the algorithm to AI@EDGE architecture. The results obtained provide an accuracy above 97% in the cases of 4, 8, 12 and 16 Nodes, demonstrating model's adaptability to environments with variable resource availability in scenarios with 2 MEC Systems for the training phase and the fulfilment of the main objective. We utilize Simu5G for the performance evaluation of the DDRL model.

4.2 Intelligent Placement of services at the edge

In this section, we will introduce, analyse, and evaluate a set of algorithms and procedures to automate the optimal location of computing nodes (far edges/near edges) in the MEC. It will also propose a set of tools to automate the dynamic placement of AIF based on the AIF descriptors. These AIFs cover the request of resources (computation, storage, and communication) demanded by the external applications rApp in the NSAP.

These algorithms have been developed considering the architecture, elements, and functionalities of the MEC (modules, interfaces, etc.) designed in this project, as well as the technologies proposed for its implementation.

This section is divided into five subsections. The first subsection 4.2.1, proposes an ML mechanism in the NSAP for the optimal and dynamic placement of the MECs and AIFs. In the first step, a solution based on centralized DQN is proposed, analysed, and evaluated to subsequently apply federated mechanisms [Sutton18]. In subsection 4.2.2, the problem model is formulated, and in subsection 4.2.3 we propose a deep Q-network (DQN)-based intelligent controller (DQNIC) located at NSAP, whose function is to implement the automation of the AIF request between several MECs.

In subsection 4.2.4 the MEO is automated, proposing a mechanism based on deep Q learning that obtains the optimal nodes where the AIF can be located, considering the instantaneous state of the system. Subsection 4.2.5 proposes an optimal solution for AIF placement in EC-enabled B5G/6G heterogeneous networks for CAVs, leveraging Deep Reinforcement Learning (DRL) to allocate computing resources based on demand and network conditions intelligently. To evaluate the performance of our approach, we employed a comprehensive testbed between value-based, policy-based, and a classical iterative Integer Linear Programming (ILP) optimization algorithm.

4.2.1 Intelligent Controller at NSAP for multiple MECs

To capture the environment characteristics mentioned above and formally define the problems we investigate, we consider the reference architecture depicted in Figure 21. Under this architecture, a global SDN controller, located at the NSAP, is deployed to manage a region compounded by several MEC systems. Thus, we denote k as a set of MECs, k ∈ N. Each MEC system k is formed by a group of physical nodes n where AIF f can be scheduled. The cluster nodes are considered energy-constrained devices, representing edge nodes with a fixed amount of computational resources.

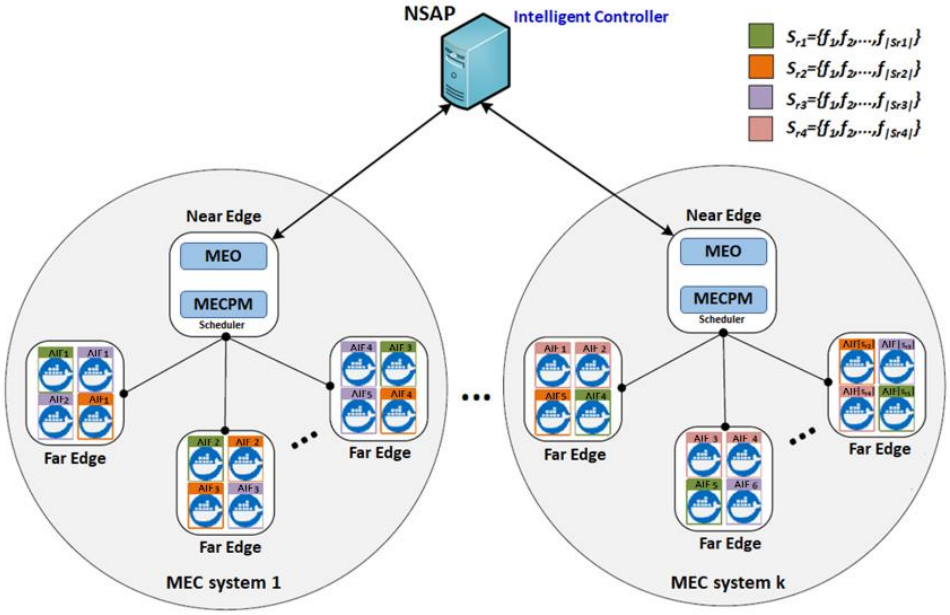


Figure 21 Reference architecture formed by several MEC systems.

The AIFs are placed in a set of deployable units within computing nodes P (far edges). Each virtual node p ∈ P is identified with an ID. We include a global parameter p_i^{kn} to indicate that virtual node p_i has been placed in physical node n of cluster k, where n ∈ N and k ∈ K.

Each physical node n in MEC system k has resource and power capacity. The former contains the computing resources (i.e., CPU and memory). The latter implies the power source that maintains the device working. Thus, we denote the resource capacity of each node by $C_{kn} = (C_{kn}^{CPU}, C_{kn}^{MEM})$, representing the available resources in terms of CPU and memory. Moreover, we indicate the available energy capacity of each node as E_{kn}, which is expressed as the state of charge (SOC). Additionally, each node has a total output bandwidth represented as W_{kn}.

The system can process services composed by a set of VNF demanded by the rAPP that generates a sequence of AIFs, $F = \{f_1, f_2, \ldots, f_{|F|}\}$.

Each service instance f ∈ for CPU and memory denoted by $D_f = (D_f^{CPU}, D_f^{Mem})$. We denote S_R as a set of service requests arriving at the controller node (NSAP). In this vein, each request $s_r \in S_R$ must be directed through the AIFs, compounding the request by considering its requirements. The service-related s_r is denoted as follows:

$$s_r = \{f_1, f_2, \ldots, f_{|sr|}\}, f_i \in F, i = 1, 2, \ldots, |s_r|.$$

Each service request s_r has specific QoS demands, such as the bandwidth requirement W_r and deadline d_r for processing the given request. Additionally, each AIF f in the request has a running time parameter (t_r) to denote the time that must pass before considering it complete. When $t_r > 0$, the event runs during the specified time. Otherwise, the event will be executed during the system's lifetime when this parameter is zero or not set.

Finally, for each request s_r ∈ S_R, we use a binary variable $x^i_{sr,kn}$ to indicate the placement decision of each AIF f_i belonging to the requested service. Specifically, $x^i_{sr,kn} = 1$ when AIFf_i is successfully placed on node n ∈ N of MEC k ∈ K; otherwise, $x^i_{sr,kn} = 0$. Table 3 provides a list of notations related to the system model.

Table 3 System model notation

Notation	Description
K	Set of MECs
N	Set of physical nodes where events can be placed
P	Set of deployable units of virtual computing nodes in the far edges
S_R	Set of network service requests arriving to NSAP
F	Sequence of AIFs compounding a network service request
k	Each MEC formed by a set of physical nodes (N)
n	Each physical node where virtual nodes are created
p	Each virtual node created on the physical node to run the events
$p_i^{k_n}$	Indicates the virtual node p_i is placed in the far edge n of MEC k
s_r	Each network service request formed by a sequence of AIFs
f_i	Each AIF forming part of a network service
C_{k_n}	Available resource capacity of node $n \in N$ of MEC $k \in K$ in terms of CPU and memory
E_{k_n}	Available energy capacity of node $n \in N$ of MEC $k \in K$ in terms of SOC
D_f	Resource demand of service instance $f \in F$ in terms of CPU and memory
W_{k_n}	Total output bandwidth of node $n \in N$ of MEC $k \in K$
W_r	Bandwidth requirement of service request $s_r \in S_R$
d_r	Deadline for processing a given request
t_r	Running time of a given request before considering it complete
t_a	Arrival time of a given request to NSAP
t_s	Starting time when the selected nodes process a given request
$x^i_{s_r,k_n}$	1 if the AIF f_i is successfully deployed on node $n \in N$ of MEC $k \in K$, 0 otherwise
$a_{s_r,\tau}$	1 if service request $s_r \in S_R$ is active in time slot $[t_s, t_s + t_r]$, 0 otherwise
$\eta^f_{k_n,\tau}$	Number of AIF instance $f \in F$ that are placed on node $n \in N$ of MEC $k \in K$ in time slot $[t_s, t_s + t_r]$
$\nu_{k_n,\tau}$	1 if any AIF instance is placed on node $n \in N$ of MEC $k \in K$ in time slot $[t_s, t_s + t_r]$, 0 otherwise
$z^{s_r}_{k_n,\tau}$	1 if any AIF of request $s_r \in S_R$ is placed on node $n \in N$ of MEC $k \in K$ in time slot $[t_s, t_s + t_r]$, 0 otherwise
y_{s_r}	1 if request $s_r \in S_R$ is deployed, 0 otherwise

4.2.2 Problem modelling

To face real-time network variations due to receiving requests with an unknown arrival time, we use the concept of time slot τ. At each time slot τ, the Intelligent Controller feed by MTO/IOC verifies the

status of nodes and links. Additionally, it can receive service requests, make deployment decisions, and update the network's states. In this regard, we define $C_{n,\tau}$ as the available capacity of node n at time slot τ. Likewise, $W_{n,\tau}$ represents the available bandwidth of node n at time slot τ.

Additionally, we define $S_{R,\tau} \in S_R$ to cope with the possibility of receiving several requests at time slot τ. Thus, simultaneous service requests will be treated as they arrive by considering the event's ranking in the priority queue. The event's ranking is calculated based on delay(f) and $wait_{queue}(f)$. The former represents the time that the virtual function's execution f can be delayed without missing its deadline (1). The latter represents the waiting time of the VNF f in the queue (2). In (2), t_a denotes the arrival time of the SFC request.

$$delay(f) = f_{d_r} - t_{now} - f_{t_r} \quad (1)$$

$$wait_{queue}(f) = t_{now} - f_{t_a} \quad (2)$$

Considering the previous definitions, we calculate the ranking score for the AIF frank as follows:

$$f_{rank} = \beta_1 \cdot delay(f) - (1 - \beta_1) \cdot wait_{queue}(f) \quad (3)$$

To represent whether request $s_r \in S_R$ is still in service at time slot τ, we use the binary variable $a_{sr,\tau}$ as follows:

$$a_{s_r,\tau} = \begin{cases} 1 \ , t_s \leq \tau < (t_s + t_r) \\ 0 \ , \text{otherwise} \end{cases} \quad (4)$$

where t_s represents the starting time when the selected nodes start processing the service request. Since we consider multiple instance deployment of VNFs in the same node, we must know the number of VNFs $f \in F$ placed on node $n \in N$ of MEC $k \in K$ at time slot τ. Thus, we utilize the variable $\eta^f_{kn,\tau}$ to reflect this value (see (5)).

$$\eta^f_{k_n,\tau} = \sum_{\forall s_r \in S_R} \sum_{1 \leq i \leq |s_r|}^{f} x^i_{s_r,k_n} \cdot a_{s_r,\tau} \quad (5)$$

Similarly, we indicate when any AIF instances are placed, at time slot τ, on node $n \in N$ of MEC $k \in K$ through the binary variable $\nu_{kn,\tau}$ as follows:

$$\nu_{k_n,\tau} = \begin{cases} 1 \ , \sum_{\forall f \in F} \eta^f_{k_n,\tau} > 0 \\ 0 \ , \sum_{\forall f \in F} \eta^f_{k_n,\tau} = 0 \end{cases} \quad (6)$$

After the previous specifications, we formally present the model defined as <S, A, R, γ>, where S represents the set of discrete states, A the set of discrete actions, R the reward function, and $\gamma \in [0, 1]$ a discount factor for future rewards. Thus, the core elements used in our model are defined as follows.

State Definition: The state $s \in S$ at time t (s^t) is represented as a vector consisting of each node's remaining resources and bandwidth and the requirements of the current AIF to be placed. Thus, state s^t is defined as follows:

$$s^t = (C^t, E^t, W^t, R^t),$$

where C^t defines the remaining resources of each node, therefore $C^t = (C_1^t, C_2^t, \ldots, C_{|N|}^t)$. In addition, the remaining state of charge of each node is described as $E^t = (E_1^t, E_2^t, \ldots, E_{|N|}^t)$. The remaining bandwidth of each node is represented by $W^t = (W_1^t, W_2^t, \ldots, W_{|N|}^t)$. Finally, the requirements of the AIFs to be scheduled is defined as $R^t = (D_i, W_{ri}, t_{ri}, d_{ri}, PF_{ri})$, where D_i is the resource demand on node by the AIF, W_{ri} represents the bandwidth demand, the running time of the service request is established

by t_{ri}, d_{ri} defines the deadline for processing the given request and PF_{ri} is the amount of AIFs in $s_r \in S_R$ waiting to be deployed.

Action Definition: We denote action $a \in A$ as a binary vector. More specifically, each position in the vector corresponds to a possible action. When the first position is 1, it indicates that no node will be selected (i.e., $a^t = (1, 0, 0, 0, ..., 0)$ do nothing). The value of 1 in one of the remaining positions infers the selected node (i.e., $a^t = (0, 0, 1, 0, ..., 0)$).

Reward Function: We define the reward function (7) as a weighted sum of objectives since we want to achieve them jointly.

$$\mathcal{R}_{s^t} = \sum_{i=0}^{\infty} \gamma^i \cdot r^t(s^{t+1}, a^t), \qquad (7)$$

where $\gamma \in [0, 1]$ is the discount factor for future reward.

State Transition: The state transition is defined as (s^t, a^t, r^t, s^{t+1}), where s^t is the current network state, a^t is the action taken (i.e., do nothing or place AIF) and s^{t+1} is the new network state after receiving the reward r^t.

4.2.3 DQN-Based Intelligent Controller at NSAP solution

In this section, we propose our deep Q-network (DQN)-based intelligent controller (DQNIC) located at NSAP to deploy virtual functions of a service request among several MECs.

After receiving a multi-deployed service request from one of the assigned MECs, the proposed solution selects the best nodes among all the MECs by considering remaining battery estimations and CPU usage. Please notice that in Figure 22 each MEC has a scheduler located at MECPM in charge of deploying local service requests using the SOC and capacity-based scheduler (SOCCS) [Llorens21].

Since our proposal needs available information (e.g., CPU, memory, and SOC) of the local schedulers to deploy services by considering a multi-MEC placement, we incorporate a mechanism in the local schedulers to guarantee their communication with the intelligent controller located at NSAP. This mechanism is also used for the proposed solution, and it is based on a data distribution service (DDS). Thus, our intelligent controller (MTO) is formed by five main elements: the global scheduler, the global monitor, the data writer, the data reader and the DDS monitor. These modules have been grouped into two categories according to their functions: DQN-based scheduler and global DDS.

Figure 22 depicts the relation among the constituent modules of the proposed solution. More specifically, the orange line represents the DDS communication between global and local entities their relationship with the schedulers' blocks. Additionally, the blue line reflects the connection between the schedulers' elements and how the local schedulers communicate with the orchestrator, MEO. Finally, the green line shows the interaction between the orchestrator and the MEC nodes.

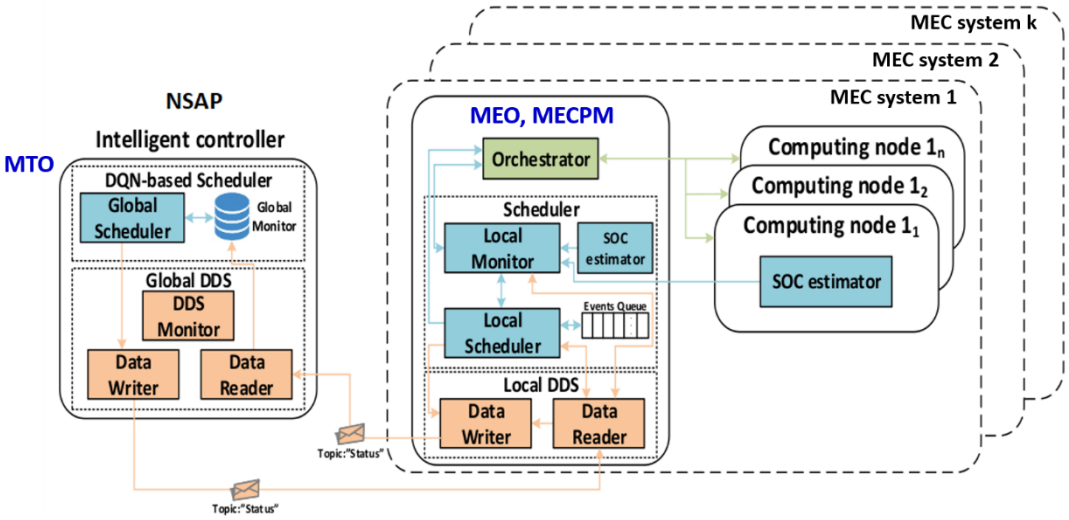


Figure 22 Intelligent controller solution design at NSAP and the relationships among its constituent modules

4.2.4 MEO, the Global and Intelligent Scheduler

This element represents the most crucial component in DQNIC. It determines the best node where a particular AIF can be placed according to predictions that consider the current state of the system. Through these predictions, the global scheduler chooses what it considers the best action. This element is built by a DQN that improves its decisions by considering past experiences and obtaining rewards from the system after each action is taken.

The architecture is shown in Figure 23. The input layer represents the state vector, and the output layer is the actions' probability distribution.

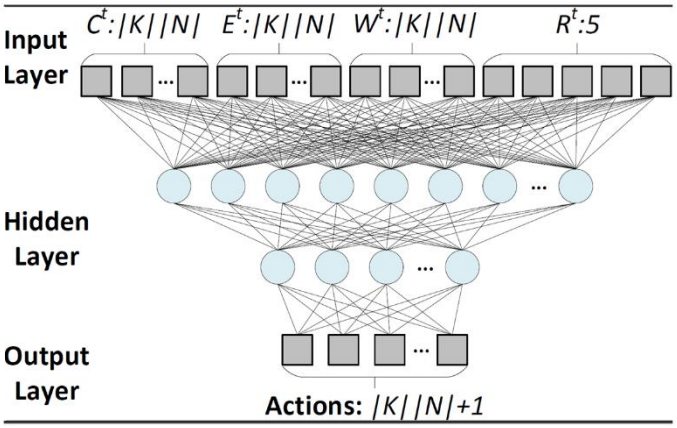


Figure 23 Neural network design to estimate the Q-value function

By considering a DQN model to estimate the possible actions that DQNIC can take, we implement the main process of global scheduler as shown in Algorithm 4.2.1.

Algorithm 4.2.1: DQNS training process.

Create DQN model according to the number of inputs and outputs or load a pre-trained model
Initialize action-value function Q with random weights Θ
Initialize target action-value function $\hat{Q}$ with weights $\Theta^{-} = \Theta$
Initialize action space ($|K||N|+1$), RL-agent and replay buffer (D) with a determined size
while $trainSteps < exploreSteps$ **do**
  Create input state (nodes' status and AIF request) using the stored information in *global monitor*
  if *RL-agent's strategy is exploration* **then**
    RL-agent selects a random action a^t from action space with probability ϵ
  else
    RL-agent selects action $a^t = max_a Q(s^t, a; \Theta)$
  Trigger *global datawriter* to indicate AIF's deployment to the selected node in action a^t
  Wait for acknowledgment of AIF's deployment from selected node
  Calculate reward r^t using (14)
  Get next state (s^{t+1}) and store transition (s^t, a^t, r^t, s^{t+1}) in D
  if $envSteps > observedSteps$ **then**
    Sample random mini-batch of transitions (s^j, a^j, r^j, s^{j+1}) from D
    forall *Transitions in mini-batch* **do**
      if *Episode terminates at step* $j+1$ **then**
        Set $y^j = r^j$
      else
        Set $y^j = r^j + \gamma * max_{a'} \hat{Q}(s^{j+1}, a'; \Theta^{-})$
      Perform gradient decent step on $(y^j - Q(s^j, a^j; \Theta))^2$ with respect to the network parameters Θ
      Every X steps reset $\hat{Q} = Q$

This algorithm runs simultaneously to the communication process managed by the global DDS. Thus, we avoid delays in the algorithm's execution due to the performance of other modules.

The main function of Algorithm 4.2.1 is to select the best node to place an AIF request by considering an input state. To this aim, we create a DQN network according to the number of inputs (e.g., node's status and AIF request) and outputs (e.g., a set of MEC nodes) in the system's model. For the computing resource inputs (e.g., CPU and SOC) in state s^t, we divide the remaining capacities of each node by a maximum $C_{max} = max(C_{kn})$, $E_{max} = max(E_{kn})$, $\forall k \in K, n \in N$. In the same manner, we also compress the bandwidth-related inputs and the rest of the AIF requirements. Thus, the normalized input state is:

$$\left(\frac{C^t_{1_1}}{C_{max}}, \ldots, \frac{C^t_{1_{|n|}}}{C_{max}}, \ldots, \frac{C^t_{|k|_{|n|}}}{C_{max}}, \right.$$
$$\frac{E^t_{1_1}}{E_{max}}, \ldots, \frac{E^t_{1_{|n|}}}{E_{max}}, \ldots, \frac{E^t_{|k|_{|n|}}}{E_{max}},$$
$$\frac{W^t_{1_1}}{W_{max}}, \ldots, \frac{W^t_{1_{|n|}}}{W_{max}}, \ldots, \frac{W^t_{|k|_{|n|}}}{W_{max}},$$
$$\left. \frac{D_i}{C_{max}}, \frac{W_{r_i}}{W_{max}}, \frac{t_{r_i}}{100}, \frac{d_{r_i}}{100}, \frac{PF_{r_i}}{10} \right)$$

4.2.5 Optimal Resource Placement in 5G/6G MEC for Connected Autonomous Vehicles Routes Powered by Deep Reinforcement Learning

This subsection presents an approach for optimizing the placement of AIFs in B5G/6G networks for CAV, using Advanced methods such as DRL. The approach considers minimum route cost, available resources, and network KPIs for every EC node along the vehicle's path. Firstly, the problem of optimal VNF placement in EC is formulated as a Markov Decision Process. Secondly, a comprehensive benchmark test is conducted to compare three approaches: value-based and actor-critic DRL algorithms and a classic optimization method [Mnih16] [Schulman18] [Schulman17].

4.2.5.1 System model and problem statement

This study considers that each Mobile Network Operator (MNO) cell comes equipped with a Next Generation Node B (gNB) base station (BS) and a Multi-Access Edge Computing (MEC) system. We deploy a set of Far Edge Computing (FEC) nodes $K = \{FEC_1,\ldots, FEC_K\}$ in each MEC system. Each FEC_k is equipped with 2048 GPU cores, 32 GB of RAM, and 10 Gbps of BW upon initialization. In general, $FEC_k = [FEC^{GPU}{}_k , FEC^{RAM}{}_k, FEC^{BW}{}_k]$ describes the status of each FEC_k, $k \in \{1,\ldots,K\}$. For each CAV request, an instance of an AIF_v ($v \in V$) is required. We assume that AIF requests arrive at the system one by one, and time is discretized into steps $t = \{1,\ldots,T\}$. An AIF_v request is defined as [s, d, g, r, b] where s, d ∈ N denote the source and destination CAV's path nodes, respectively.

The parameters g and r represent the cloud-aware demands for GPU cores and RAM on each FEC_k along the CAV route. Additionally, b is the network and service KPI that must be fulfilled regarding BW. For instance, [1, 45, 512, 25, 1] is a VNF request that needs from node 1 to 45, 512 GPU cores, 25GB of RAM, and 1 Gbps of BW.

We establish a wired connection between each FEC_k node and its corresponding MEC system within the cell, which allows the node to establish wireless connectivity with the CAV. Furthermore, we assign each FEC_k node to provide coverage for a specific subset of the streets set, which we illustrate in Figure 24 using different colours. These colours provide a clear visual representation of the coverage area for each FEC_k, making it easier to interpret the model. In this setup, every node falls within the coverage area of two different FEC_k nodes, which allows a CAV to request an instance of VNF_v from two different FEC_k nodes.

The proposed system can generate a customizable number of background vehicles (BV) that adhere to the same principles as the instance's CAV, regarding VNF_v requests. However, unlike the CAV, the routes of the BVs are predetermined using the Dijkstra algorithm to identify the minimum route cost. In the event that a FEC_k denies resource allocation or the transport KPI cannot be met for a BV, it remains stationary at its current node until the required resources or transport KPI are available. This feature allows for the simulation of traffic congestion, which the CAV can detect as road events and avoid accordingly. All BVs move simultaneously from node to node within each time step ($t \in T$). To ensure that the desired number of BVs in the scenario is maintained, once a BV completes its assigned route, a new vehicle is generated with a new incoming AIF_v.

The network management system keeps track of the AIF_s at each FEC_k node and manages the handovers between cells and FEC_k nodes within a cell. Once the CAV completes its route, a new AIF_v arrives in the system, and this process continues until either a FEC_k rejects a request or the maximum number of allowed time steps is reached, or the vehicle completes the path.

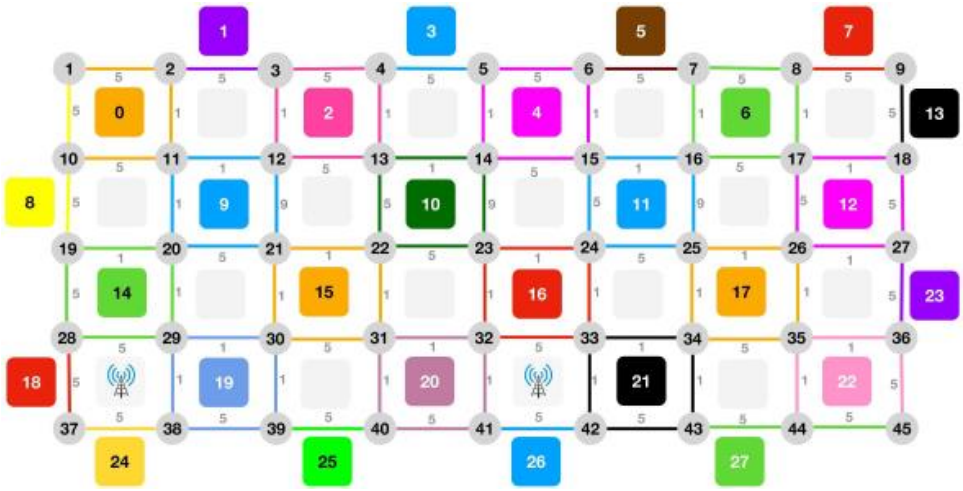


Figure 24 Scenario for the CAVs comprising two network cells totalling 27 FEC nodes

The CAV scenario is modeled by an undirected weighted graph G = (N,E), adopting a Manhattan grid layout, being N the set of nodes, and E the set of edges. The nodes represent the crossroads, and the links are the street of this Manhattan like city. Each link (i, j) ∈ E, being i, j ∈ N, is characterized by a cost c_{ij}, which allows us to evaluate and optimize the performance of CAVs under various scenarios and conditions. Moreover, each link has one FEC associated according to the corresponding coverage area, f_{ij}. Considering the previous notation, this FEC will have [f^{GPU}_{ij}, f^{RAM}_{ij}, f^{BW}_{ij}] resources. Each analysed CAV maintains information on the FEC at which it is connected along the path, f. Utilizing an Integer Linear Program (ILP), we determine the next hop of each CAV that allows it to reach the destination using the less-cost path and to have sufficient network resources to meet the requirements of its VNF request. Thus, the ILP is a one-hop AIF Placement and Routing Problem (One-Hop-AIFP-RP).

To formulate this problem, let's define x_{ij}, a binary variable indicating if the link (street) (i, j) ∈ E is part of the constrained path followed by the analysed CAV:

$$x_{ij} = \begin{cases} 1 \text{ , if street } (i,j) \text{ is used along the vehicle path} \\ 0 \text{ , otherwise.} \end{cases}$$

The optimal One-Hop-AIFP-RP aims to minimize the total cost of the vehicle path, with the restriction of having enough resources in the following path hop. In addition, flow conservation constraints are needed to ensure a correct path leading to the destination. Finally, a resource capacity constraint will be added to calculate the next hop for each vehicle, checking the availability of resources only in the first hop and if the corresponding FEC is different from the one currently connecting the vehicle, f. With all of these considerations, the ILP for the One-Hop- AIFP-RP is the following:

$$
\begin{aligned}
\min \quad & \sum_{\forall (i,j) \in E} c_{ij} \cdot x_{ij} \\
\text{s. t.:} \quad & \sum_{\forall (i,j) \in E} x_{ij} - \sum_{\forall (j,i) \in E} x_{ji} = 1 && \text{if } i = s, i \in N \\
& \sum_{\forall (u,v) \in E} x_{ij} - \sum_{(j,i) \in E} x_{ji} = 0 && \forall i \in N | u \neq \{s,t\} \\
& x_{ij} \cdot g \leq f_{ij}^{\text{GPU}} && \text{if } i = s, \text{ if f} \neq f_{ij} \\
& x_{ij} \cdot r \leq f_{ij}^{\text{RAM}} && \text{if } i = s, \text{ if f} \neq f_{ij} \\
& x_{ij} \cdot b \leq f_{ij}^{\text{BW}} && \text{if } i = s, \text{ if f} \neq f_{ij} \\
& x_{ij} \text{ binary} && \forall (i,j) \in E.
\end{aligned}
$$

Figure 25 shows the iterative algorithm to find the result of an episode using the One-Hop-AIFP-RP model. Step 1 creates the parsed car with its request and background cars. From there, the system evolves iteratively in Steps 2 and 3, finding each hop of all the vehicles and considering the network resources if it is necessary to change the FEC. A background car that does not find resources to advance supposes its elimination and the new creation of another vehicle. If the leading car is the one that does not locate resources, the episode ends without the results being valid. On the other hand, if it ends because it reaches the destination, the results are stored for later analysis. In the next simulation part, the graphs are shown, comparing them with one DRL method. Integer Linear Programs (ILP) to solve combinatorial optimization are well-known to be NP-hard problems in general.

Even though pure optimization-based methods are guaranteed to provide an optimal solution if enough time is given, their use in online applications remains a significant challenge due to their high computational time. Some machine learning techniques, such as DRL, have been proposed to alleviate their computational burdens in the literature.

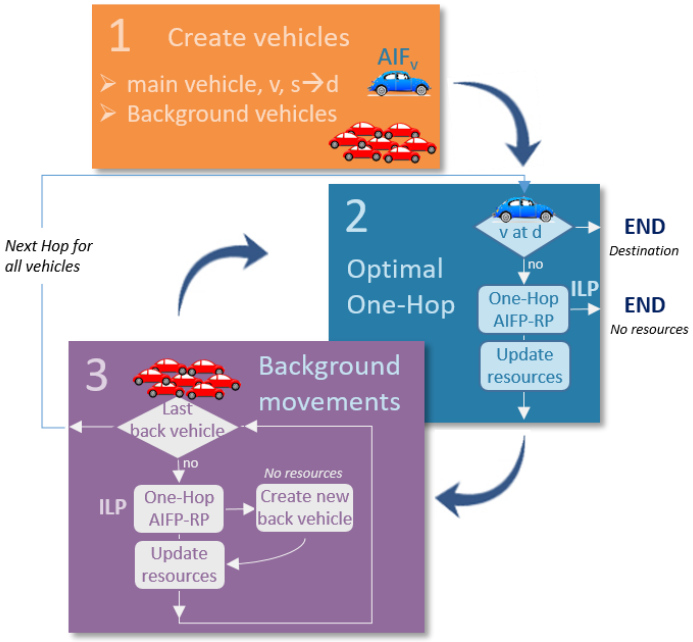


Figure 25 Iterative Optimization Model for one Episode

4.2.5.2 Approached solutions

In this subsection, first, we frame the system model as a Markov Decision Process (MDP). Then, we introduce the DRL algorithms used in our approach: namely, Deep Q-Network (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO) formulations.

According to the latter, we formulate our MDP as (S,A,R), where S is a vector s_t = [AIF_v,w_i, FEC^A_k , FEC^B_k]. The AIF_v is the incoming AIF descriptor. The w_i is a vector compounded of the current node indicating where the agent is located at every t, and the maximum time steps t'_{max} remaining t′ to be left for the agent to reach its destination before the episode ends. The FEC^A_k and FEC^B_k contain information about the two available FEC_k options from which the CAV can choose at any (i, j) ∈ E. The vectors provide detailed insight into the capacity and network status of the available FEC_k options, helping the CAV make an informed decision about which option to choose. To guide the CAV through the environment, the agent has a set of available actions A at every n ∈ N, which includes the directions of movement such as A = {north, south, east, west}. If the CAV is at a peripheral node and the agent's action is not within the set of available A, it will remain in its current node until a valid action is taken by the agent, allowing the CAV to move in the desired direction.

The reward function R(s, a, s′) is directly influenced by the actions taken by the agent. In this proposal, we have developed a reward function that accounts for a variety of factors, including invalid actions at a peripheral node, reaching t'_{max}, choosing a FEC_k with insufficient resources, successful CAV movements, reaching the final destination s_d, also reaching s_d with the minimum route cost, or visiting a given n ∈ N more than one time. By considering these factors in the reward function, we aim to incentivize the agent to make intelligent decisions and efficiently navigate the environment toward the goal.

- **Invalid action**: Let $\mathcal{A}$ be the set of available actions within $n \in N$ and $a \notin \mathcal{A}$ be the invalid action.

$$\mathcal{R}(s,a,s') = -100, \quad \text{if } a \notin \mathcal{A} \; \forall \; n \in \mathcal{N} \qquad (1)$$

- **Timestep limit**: Let the total number of timesteps required to complete a route be $\leq t'_{max}$.

$$\mathcal{R}(s,a,s') = -100, \quad \text{if } t'_{max} = True \qquad (2)$$

- **No resources**:

$$\mathcal{R}(s,a,s') = \begin{cases} -100, & \text{if} \quad g > FEC_k^{GPU} \\ -100, & \text{if} \quad r > FEC_k^{RAM} \\ -100, & \text{if} \quad b > FEC_k^{BW} \end{cases} \qquad (3)$$

- **Successful CAV movement**:

$$\mathcal{R}(s,a,s') = -c_{ij} \quad \forall \; (i,j) \in E \qquad (4)$$

- **Revisiting node**: Let S be a sequence of visited $n \in N$ during a CAV's route. Then the set of repeatedly visited nodes in a route can be defined as:

$$\mathcal{N}_S = \{n \in \mathcal{N} : count(n) > 1\} \qquad (5)$$

$$\mathcal{R}(s,a,s') = \mathcal{R}(s,a,s') \cdot count(n), \quad \text{if } count(n) > 1 \qquad (6)$$

- **Final destination**:

$$\mathcal{R}(s,a,s_d) = \mathcal{R}(s,a,s') + 50 \qquad (7)$$

- **Minimum route cost**: Let $\mathcal{PA}$ be the set of possible routes from n_s to n_d. Let a_r be the agent's completed route and $min()$ a function that returns the smallest element within the set.

$$\mathcal{R}(s,a,s_d) = \mathcal{R}(s,a,s_d) + 100, \quad \text{if } \min(\mathcal{PA}) = a_r \qquad (8)$$

4.2.5.3 *Simulations*

This subsection presents the simulation results comparing our selected DRL model against the ILP algorithm. Firstly, we describe the training methodology and hyperparameters tuning process for each DRL (DQN, A2C, and PPO). For each experiment, we ran the solutions 100 times (episodes), which enabled us to present simulation results with confidence intervals of 95%. The simulations were performed on a workstation with a 3.60 GHz Intel Core-i9 processor, 2 NVIDIA RTX2080 GPUs, and 64 GB of RAM.

To train the DRL models for the scenario described previously, we conducted a hyperparameter sweep grid search to find the optimal configuration for each model. Specifically, for the DQN model, we varied the learning rate (α), discount factor (γ), batch size (B_s), hidden layer configuration (Hl), and optimizer (Op). Across several routes, we explored α values between 0.0001 and 0.001, γ values of 0.5, 0.75, and 0.99, B_s values of 32, 64, and 128, and Hl values of [128, 64] and [256, 64]. We also tested two optimizers, Adam and RMSprop. This process yielded a total of 72 results, from which we determined the optimal configuration to be: $\alpha = 0.001$, $\gamma = 0.75$, Bs = 128, Hl = [256, 64], and Op = Adam. For the A2C model, we also used a grid search approach to explore the effects of different hyperparameters on performance. Across multiple routes, we varied the GAE parameter between 0.95 and 1, entropy coefficient (H) between 0.1 and 0.01, α between 0.0001 and 0.001, and Hl configuration of the policy network between [256, 64] and [128, 64]. We also tested two Op, Adam and RMSprop. After testing a total of 32 hyperparameter combinations, we determined that the optimal values were: GAE = 1, H = 0.01, $\alpha = 0.001$, Hl = [256, 64], and Op = Adam. Similarly, as in A2C, for the PPO model, we conducted a hyperparameter sweep and found the optimal configuration to be GAE = 1, H = 0.01, $\alpha = 0.0001$, Hl = [128, 64], and Op = RMSprop. Overall, our tests showed that these hyperparameters yielded good performance for all three DRL models. However, to achieve even more precise results, it may be necessary to fine-tune them further.

After configuring each model, we set five rounds of training, with each round comprising 3 million timesteps. At the end of each round, the model undergoes retraining for an additional round of 3 million timesteps, and this process continues until all five rounds have been completed. A fixed number of timesteps were used instead of episodes to account for the potential variability in route completion times caused by FEC congestion. This allowed for more precise training and evaluation of the agent's performance.

We must note that we have established a maximum limit of 25 timesteps for the agent's CAV to complete a given route. We trained the models on three routes: the route from s 1 to d 45, from s 41 to d 5, and from s 19 to d 27. To simulate realistic traffic scenarios, we defined a maximum of 25 background vehicles to share the network for each route. To ensure uniformity in traffic demands, we set the same requirements for incoming AIF_v for the agent's CAV and the background vehicles. The AIF_v for each vehicle is defined by a tuple of parameters, namely, [s, d, 512, 8, 1]. To address the exploration-exploitation dilemma, each model is equipped with specific techniques.

DQN, for instance, employs an ϵ-greedy policy to balance exploration and exploitation during training. Specifically, the agent selects the action with the highest Q-value with a probability of $1-\epsilon$, and selects a random action with a probability of ϵ. In this study, we set the value of ϵ to start at 1 and gradually decrease over the first 1.5 million timesteps until it reaches a value of 0. This approach allows the model to explore various actions during the early stages of training while gradually exploiting its accumulated knowledge after 1.5 million timesteps. On the other hand, A2C and PPO use a similar method to apply exploration and exploitation through an entropy term in the objective function. The entropy term encourages exploration by penalizing policies that are too certain or deterministic, thus encouraging the agent to try new actions and explore new states. An entropy coefficient controls the entropy bonus, which determines the trade-off between the policy's entropy and its expected reward. A higher entropy coefficient increases the importance of the entropy term and thus encourages exploration. In contrast, a lower entropy coefficient prioritizes exploitation and encourages the agent to focus on exploiting its

current knowledge. Taking into account the previously mentioned aspects, the training results for all three models on each of the three routes are illustrated in Figure 26, Figure 27, and Figure 28.

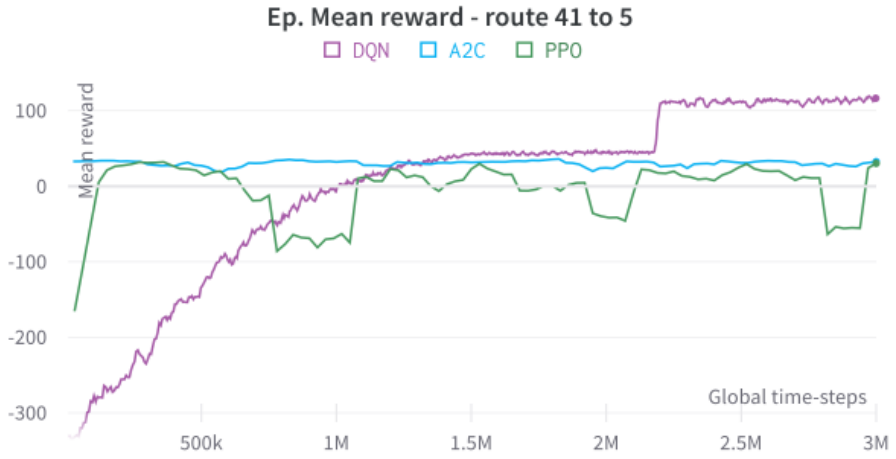


Figure 26 Episode mean reward during training for route s 41 to d 5

In Figure 26, DQN outperforms A2C and PPO, although it still fails to achieve an average reward over 100 per episode.

The complexity of the route and the presence of multiple FEC_k nodes along the path are two key factors that contribute to the actual results. Moreover, these FEC_k nodes are also responsible for serving a significant number of background vehicles, which can further exacerbate the resource contention issue. Despite the challenges mentioned earlier, the CAV agent completed the route with an average cost of 22. While this is higher than the optimal cost of 14, assuming no FEC_k rejections occur, it is still a promising result, given the complexity of the route and the uncertainties involved. The superior performance of DQN can be attributed to its ability to learn an optimal Q-value function that maps state-action pairs to their expected cumulative rewards.

By leveraging experience replay, DQN can effectively break the correlation between consecutive samples, facilitating more efficient and stable learning. In contrast, A2C and PPO directly optimize the policy function and typically require more samples to achieve comparable performance. Additionally, DQN's ϵ-greedy exploration strategy allows it to balance exploration and exploitation effectively, which can help avoid getting stuck in local optima.

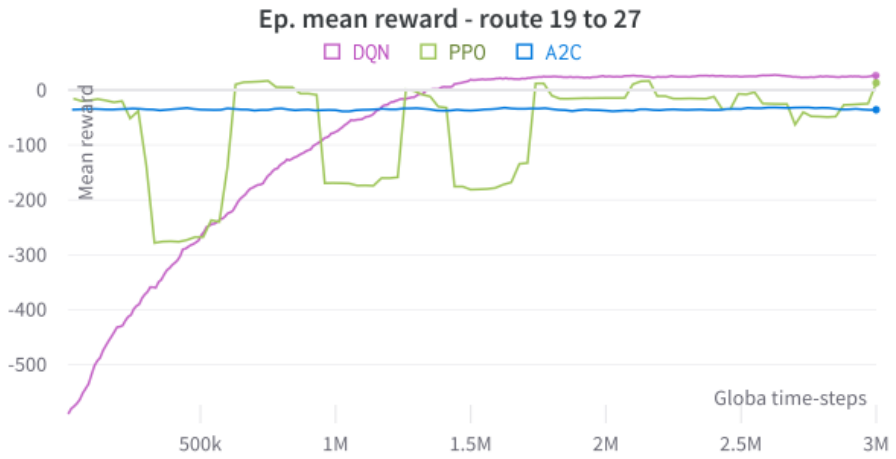


Figure 27 Episode mean reward during training for route s 19 to d 27

In Figure 28, similar results to those of Figure 27 can be observed in terms of the mean reward per episode. One interesting observation is the similar behaviour of A2C and PPO across all three training routes. It can be seen that from the first timestep, both algorithms apply the learned policy from all previous training rounds. However, while A2C does not show any margin of exploration, PPO does, but with poor results. Since both A2C and PPO are policy-based algorithms, once they learn the best policy out of the policy space so far, it seems difficult to change the strategy. This is in contrast to DQN, which is a value-based algorithm and thus has a clearer exploration-exploitation trade-off. As a result, DQN is better equipped to adapt to changes in the environment and learn better strategies.

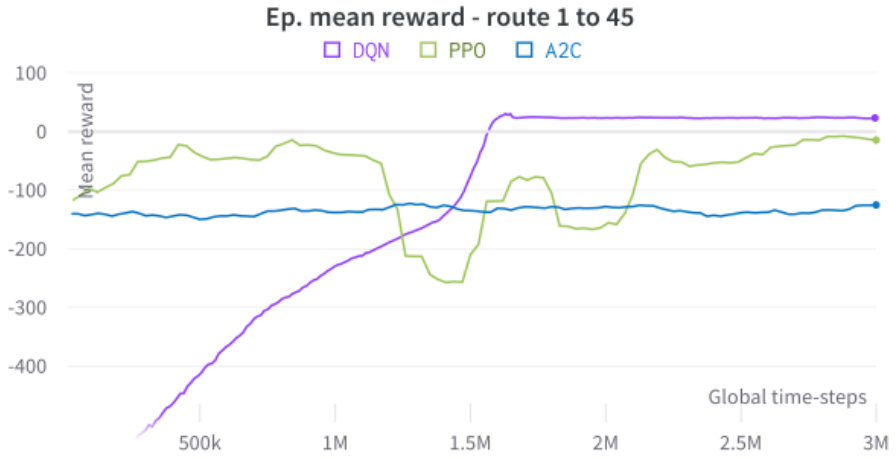


Figure 28 Episode mean reward during training for route s 1 to d 45

The purpose of the following simulations is to measure the performance difference between the DQN model and the proposed iterative ILP algorithm. To this end, the DQN algorithm will operate in production mode, meaning actions will be selected deterministically. Specifically, the agent will select the action with the highest Q-value for previously visited states as determined by its training process. The agent will approximate the optimal Q-value using its learned model for unvisited states. It is worth noting that no exploration-exploitation technique will be used during the simulations. Additionally, the background vehicles will be generated without a seed, following a random uniform distribution.

To evaluate the performance of the algorithms under study, we will measure several key metrics, including the route cost, number of route hops, and decision response time. To provide a visual representation of the performance metrics for both the DQN and iterative ILP algorithms, we have generated the following figures: Figure 29, Figure 30 and Figure 31. Each figure displays the evaluation of the respective key metric for all three routes under study, allowing for a direct comparison between the two algorithms. The simulation results show that the route from s 1 to d 45 exhibits the largest difference in performance between DQN and iterative ILP, in terms of cost, as depicted in Figure 30. In ideal conditions, the minimum cost for this route is 26, which requires the highest number of hops among all routes. As the number of background vehicles increases, DQN tends to increase the route cost while maintaining a similar average number of hops as iterative ILP (Figure 29). However, DQN outperforms ILP significantly in terms of decision time response (Figure 31), taking only a fraction of the time (up to 7 times faster) to complete the route compared to ILP. Despite the cost performance gap, the faster decision time response of DQN makes it a more appealing choice for real-time decisions. The performance of DQN is comparable to ILP in terms of cost and hops for the route s 41 to d 5.

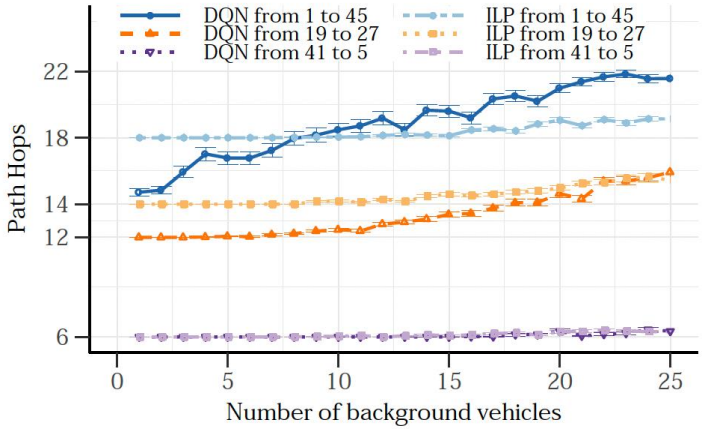


Figure 29 Number of hops comparison between DQN and iterative ILP

The average mean reward per episode confirms that the DQN model achieves route completeness through the minimum cost possible route. Both methods exhibit similar performance in terms of the number of hops required for the route. However, there is a significant difference in the decision response time between DQN and ILP.

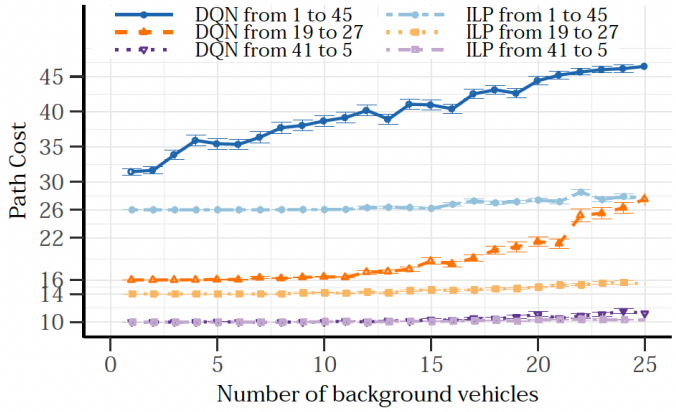


Figure 30 Route cost comparison between DQN and iterative ILP

On average, ILP takes up to 600 ms to complete the route, while DQN takes only 35 ms to initiate and up to 75 ms with 25 background vehicles. Overall, DQN provides much faster decision response time compared to ILP.

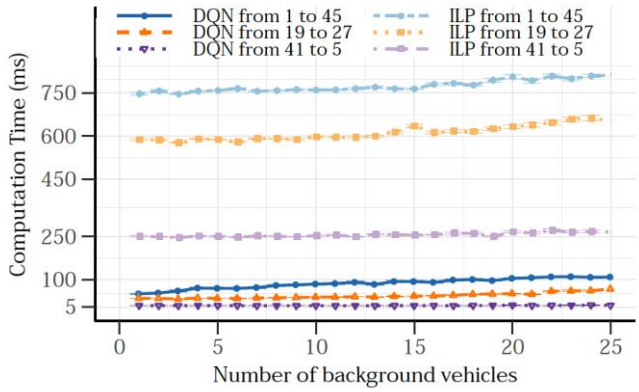


Figure 31 Response time comparison between DQN and iterative ILP

In summary, the performance difference between ILP and DQN varies for different routes regarding route cost, with the largest gap observed for longer routes. Increasing the number of iterations and fine-tuning hyperparameters for DQN could potentially improve its performance for longer routes. DQN significantly outperforms ILP in terms of decision response time, which could be crucial in real-time scenarios.

4.2.5.4 Conclusions

A proposal for an intelligent controller has been made in the NSAP based on ILP and DQN to automate dynamic placement requests of the AIF in the MECs based on their available instantaneous resources. We have also proposed a system model for CAVs and present a DRL approach to efficiently place AIFs in FEC nodes while minimizing route costs, considering capacity and network transport metrics. Our training phase for the three DRL proposals demonstrates that the value-based DQN algorithm outperforms the policy-based A2C and PPO algorithms.

4.3 AIF placement for FL-based AD framework

Edge intelligence combined with federated learning is considered to distributed learning and inference tasks in a scalable way, by analysing data close to or where it is generated, unlike traditional cloud computing where data is offloaded to remote servers. In this work, we address the placement of Artificial Intelligence Functions (AIF) making use of federated learning and hardware acceleration. We model the behaviour of federated learning and related inference point to guide the placement decision, taking into consideration the specific constraint and the empirical behaviour of a virtualized infrastructure anomaly detection use-case. Besides hardware acceleration (HWA), we consider the specific training time trend when distributing training over a network, by using empirical piece-wise linear distributions.

4.3.1 Optimal AIF placement

We define the AIF placement as the problem of finding the optimal number of AIFs and their location on a given network graph, considering the inter-AIF communication patterns, target learning loop time performance and the presence of hardware acceleration. We model the placement problem as a MILP (Mixed Integer Linear Programming) and we propose two variants of the problem.

Additionally, in Figure 32 we present an adapted architecture of the AIF. To support its operation, we identify five interfaces:

- if1: used by the orchestration platform for the communication with the AIF, including its configuration (e.g., for dynamic update of federated learning hyper-parameters)
- and the retrieval of inference results (e.g., inference running at the server AIF and/or edge AIFs);
- if2: used for AI model parameter exchange among AIFs (AIF control plane interface), e.g., the communication between edge AIFs and server AIF in federated learning;
- if3: used for data exchange among different AIFs (AIF data-plane interface) - which may be used for generic distributed learning, in the case of an AIF forwarding graph. if3 is not used in the case of FL;
- if4: hardware acceleration interface with components as GPU and Smart-NICs, for training and inference tasks;
- if5: for data collection and streaming, to interface with a data-pipe-lining system. In this work, data pipelining delay is negligible.

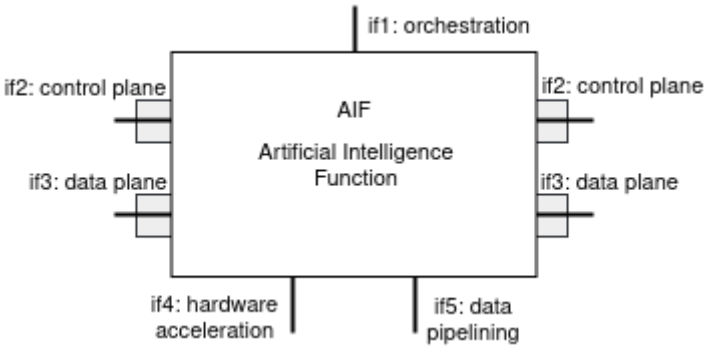


Figure 32 An adapted representation of the AIF

More precisely, with respect to the adapted representation of the reference AIF model presented in Section 2.3 that we show in Figure 32, in this work we consider that:

- Communication with the orchestration platform (if1) is possible at any node where AIF can be placed;
- AIFs interact using federated learning, hence using the (if2) interface for exchanging AI model parameters;
- Interface (if3) is not used, because in federated learning raw data is not exchanged among nodes;
- Interface (if4) is enabled only at a subset of the candidate AIF location for HWA;
- Data pipelining delay (if5) is negligible;
- The propagation delays over the link between candidate edge AIF locations and server AIF are not negligible with respect to the learning time;
- The servers hosting AIFs offer them the same computing capacity.

The optimization goal is the placement of the FL server and clients that are both deployed as AIFs.

4.3.2 Implementation

We consider a set N of physical servers such that an AIF can be hosted by any server. The communication latency between edge AIFs and the FL server AIF depends on the placement decision. Whereas the distributed training time t is a decreasing function of the number of AIFs, and does not depend on the AIF location, as we suppose the same computing capacity is delivered to AIFs independently of the location. The main goal is to minimize the number of active AIFs used for training, while guarantying that the overall training loop time (including distributed edge AIF training time and edge-server AIF delay) is at most T , or equivalently, that the time of a single FL round is no longer than T/R In fact, the time of a single round depends on the distributed training time t and the transmission latency.

In order to build a purpose-built model of the training time as a function of the number of AIFs that are used, we run the FL-based anomaly detection framework proposed in [BinRuba22] which consists of training LSTM (Long-Short Term Memory) autoencoders in a distributed manner, to learn and reconstruct the normal working conditions of a given system. To do so, we run a set of AIFs on a Kubernetes infrastructure where the placement is done automatically using a kubernetes scheduler. In fact, each AIF is an implementation of a LSTM model. The model is composed of a set of autoencoders that allow to detect anomalies at different system levels, i.e., physical level, virtual level, and access level, using different groups of metrics, e.g., CPU, memory and radio metrics.

We use the 5G3E dataset from [ChiPhung22], providing few dozens of feature time-series for each resource group, where groups are CPU, RAM, network link state, storage resources, and RAN and UE nodes. We train the ML model using data batches of 800 samples each, corresponding to 2 minutes of collected data for each data batch: this is the assumed retraining time of the system, which could vary in general depending on the sampling rate. The batch size is set to the data size, hence considering all the samples. The number of epochs is 10 and the model is trained for one round. The rest of the FL-based anomaly detection AIF hyper-parameters are the same as in [BinRuba22].

More details on [BinRuba22] can be found in Section 6.1.

In Figure 33, we present the corresponding training time distribution as a function of the number of active AIFs based on CPU resources (collected at the container level).

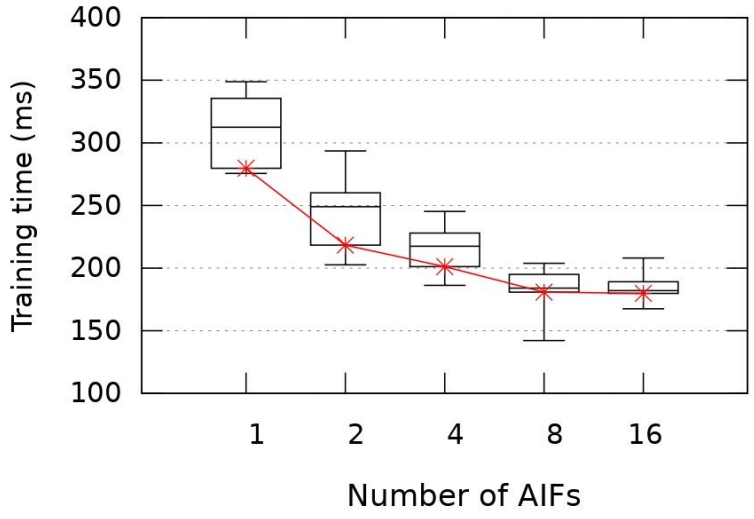


Figure 33 Training time vs number of AIFs. R = 1, E = 10

We can remark that the training time decreases with the number of AIFs up to a certain threshold value. A certain variance exists, due to CPU scheduling and storage system systemic variations at the operating

system level. We employ in the AIF placement model the piece-wise linear function obtained using the first quartile values, indicated in red in Figure 33.

We rely on the obtained results to build the AIF placement model. The mathematical model is available in [Yellas22].

4.3.3 Performance Evaluation

We solve the AIF placement problem using the following algorithms:

- Baseline: simplified case where no hardware acceleration is considered;
- AIF-P: the proposed placement MILP algorithm including hardware accelerators;
- AIF-P-var: MILP variant of the AIF placement model (more details can be found in [Yellas22]).

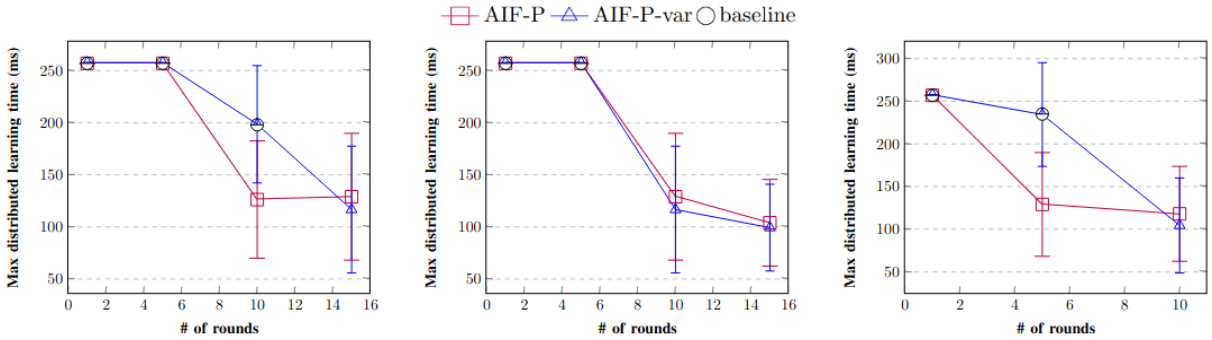
We propose to analyse the impact of the following parameters on the behaviour of the four aforementioned algorithms:

- Number of rounds: we test different rounds of FL training, i.e., 1, 5, 10 and 15, before the inference step. In fact, increasing the number of rounds helps increase the quality of the learning.
- Target learning time: we evaluate the proposed algorithm for different target times, i.e., 2 s, 1.6 s and 1.2 s. This parameter specifies the time during which we train the model before its exploitation for inference.

The rest of parameters are detailed in [Yellas22].

4.3.3.1 Maximum local training time

In Figure 34 we present the average of the maximum edge AIF distributed learning time (maximum among all the feasible solutions) for the different 15 hardware accelerator configurations, for different target times (a, b and c) and using different numbers of rounds. The error bars represent the standard deviation. We can notice that:



(a) Target learning time=2 s (b) Target learning time=1.6 s (c) Target learning time=1.2 s

Figure 34 Max learning time vs number of rounds

We can notice that:

- The distributed learning time decreases with the increase of the number of rounds for all target times. as already discussed; this can be explained by the fact that increasing the number of rounds requires higher number of active AIFs at each round since the increase of this latter decreases the training time.
- In Figure 34 (a), we can notice that with 15 rounds, and unlike all the other cases, AIF-P suffers a slight increase in the training time when compared to 10 rounds. This apparent contradiction is due to the fact that we consider only the maximum distributed processing time over all the

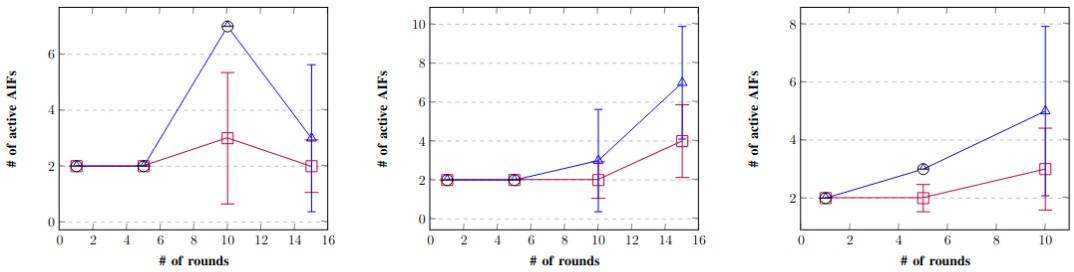
feasible solutions. Some instances that yield solutions with high learning times using 10 rounds are not accounted in the case of 15 rounds since no feasible solution exists for them.

- We can also observe that AIF-P and AIF-P-var have similar behaviours: they can produce a placement solution for all the proposed numbers of rounds for a target learning time equal to 2 s. On the other side, both models are not able to find any feasible solution for some stringent time constraint cases; these unfeasible solutions correspond to a number of rounds higher than 10 for a target learning time equal to 1.2 s. Moreover, AIF-P produces a lower distributed training time than AIF-P-var when R = 10 for T = 2s (R = 5 for T = 1.2 s respectively). However, this is not the case anymore when R increases. Differently than AIF-P-var, AIF-P tends to use hardware accelerators instead of increasing the number of deployed AIFs in order to respect the learning time threshold.

The baseline solution shows a worst performance when compared to the other methods: any feasible solution is found for a number of rounds that surpasses 10 and 5 when the target time is equal to 2 s (Figure 34 (a)), and 1.6 s and 1.2 s respectively (Figure 34 (b) and (c)).These results confirm the usefulness of hardware accelerators during the training phase. In fact, the reduction in the distributed training time is between 50% and 59% for the two approaches using hardware acceleration.

4.3.3.2 Number of active AIFs:

Figure 35, we present the average number of active AIFs using the different algorithms, while varying both the number of rounds and the target earning time. The length of the error bars represents the variation in the number of AIFs produced by the different 15 simulations for the same case, i.e, same number of rounds and the same target learning time.



(a) Target learning time=2 s (b) Target learning time=1.6 s (c) Target learning time=1.2 s

Figure 35 Number of active AIFs vs number of rounds

- As expected, the plots show an inverse relationship between the distributed processing time and the number of deployed AIFs (Figure 34 and Figure 35). All the strategies show an increase in the number of AIFs with the increase of the time constraints, except for the case of 15 rounds with a target time equal to 2 s using AIF-P and AIF-P-var, as shown in Figure 35 (a). As mentioned before, we consider the average of the obtained values only for the feasible solutions. For some settings, no feasible solution was found with 15 rounds. When T =2 s and T = 1.6 s (respectively T = 1.2 s) for a number of rounds equal to 10 (respectively 5), AIF-P deploys a lower number of AIFs than AIF-P-var, however the distributed training time is lower with AIF-P. This can be explained by the fact that this latter uses (more) hardware accelerators to reduce the distributed learning time instead of increasing the number of AIFs.
- Figure 35 (b) and (c) show that the baseline algorithm cannot find any feasible placement solution when the number of rounds exceed 5. This can be explained by the increased number of exchanges between the AIFs and the FL server which has a direct impact on the overall learning time. In this case, finding a feasible solution with respect to the imposed target times

is not possible. On the other hand, AIF-P and AIF-P-var show higher performance in placing the AIFs with stringent time constraints thanks to the use of hardware accelerators.

- It is worth mentioning that there exist some settings where AIF-P and AIF-P-var do not provide feasible solutions. This happens when we have strict time constraints. For instance, through the 15 instances, both algorithms provide 30% of unfeasible solution with T = 2 s and R = 15. This is related to the placement and the number of hardware accelerators on network nodes provided for each instance.

The possibility to use hardware accelerators reduces the per-round learning time, hence offering more flexibility in placing and activating AIFs and consequently training the model for a higher number of rounds. Also, the use of advanced formulations may have less benefit if the baseline strategy achieves a feasible solution. Their benefit can be shown when the constraints on target time are more stringent or the number of rounds is increased.

4.3.4 Conclusion

In this work, we have proposed a FL-AIF placement framework where we considered the use of HWA to reduce the local training time. The main objective was to reduce the maximum local training time while minimizing the number of active AIFs to reduce the deployment costs and increase the quality of learning as the AIFs can have more data to train and consequently a more complete view of the system. Currently, we are extending the AIF placement model to consider heterogeneous CPU resources that can be used by the AIFs. We redesign the objectives to both minimize the number of stragglers and the CPU costs. To do so, we proposed to include multiple components to represent the training time while proposing a stochastic variant for the placement strategy to ensure robustness against stochastic delays.

4.4 Compression of Activation Signals from Split Deep Neural Network

The rapid advancements in computation capabilities of edge devices such as mobile devices, drones, etc. have empowered many deep neural network (DNN) inference tasks to be migrated from the cloud to the edge device. However, low computation powers of edge devices relative to cloud servers can cause high inference latency. For example, considering a scenario with AlexNet that has more than 62 million parameters and an edge device responsible to collect the data, processing it, and reacting with responses, the time response and battery consumption would create strong limitations to the system.

Several approaches have been proposed to conduct DNN inferences on resource-constrained edge devices. Split learning and inference frameworks are one of the possible solutions which take advantage of the use of a server with large resources. In a split inference framework, a DNN model is split into edge-side and server-side networks assigned to the edge device and the server, respectively [Gupta18]. In [Eshratifar19], the authors propose an efficient, adaptive, and practical engine, called JointDNN, for collaborative computation between mobile devices and cloud for DNNs in both inference and training phase. In [Vepakomma18s], the authors introduce different configurations of split learning as part of a method called SplitNN to facilitate collaboration between health entities and eliminate the necessity of sharing sensitive raw data.

Another approach that has been used to produce small DNN models is the usage of compression methods. More specifically, techniques like model pruning and quantization have been used to remove parameters from the model and use fewer bits to represent them [Li20]. In [Jacob18], the authors empirically show that their quantization technique leads to an improved trade-off between inference time and accuracy on MobileNet for image classification tasks compared to the original, float-only MobileNet.

Our work is focused on a scenario that considers the use of the two techniques mentioned above: split inference and compression. However, unlike the work presented in [Eshratifar19] and [Vepakomma18],

we perform the split on already trained convolutional neural network (CNN) models, following the simplest split configuration. Also, the compression method proposed is performed on the output of the activation maps from the CNN, called activation signals (or scores), not in the network model as described in [Jacob18] and [Stock19].

In [Ko18] a DNN partitioning between the edge device and the host device is proposed in order to perform coordinated inference and improve the energy efficiency and transfer rate of the edge device. The middle layer output is compressed to reduce the amount of data transmitted to the host device through two techniques: JPEG encoding and RLE+Huffman. Results show that the authors achieved 15 times power reduction and 16.5 times bit rate improvement than just deploying the model entirely to the host device, plus a 2.3 power reduction and bit rate improvement of 2.5 times compared to deploying the model on the edge device.

In [Georgiadis19], the authors propose a new training method with the goal of improve the accuracy, reduce the model complexity through regularization and improve the sparsity of the scores. After the training stage, the authors propose the use of scalar quantization and entropy coding. The results show a compression of approximately 7 times on LeNet-5 and 6 times on MobileNet-V1.

Our original model M with K+L-1 layers is split into two parts. The first submodel M_1 contains the layers 0 to K-1 and the second submodel M_2 contains the layers K to K+L-1. The output scores of the layer K-1will be compressed. This configuration is useful for scenarios, as mentioned above, where the edge devices have restrictions regarding computation cost and the deployment of an entire model into them can lead to problems regarding processing time and battery consumption. Also, sending the original output directly to the cloud can lead to security problems. Instead, sending the scores compressed can add additional step security against attackers. While [Ko18] used a method based on JPEG (DCT+RLE+Huffman), our method has lower computational cost. Also, different from [Georgiadis19], in which the focus was to increase the inference speed and neural network compression, our compression method is focused on the optimization of the channel usage without facing a great accuracy loss.

4.4.1 Proposed Method Description

Figure 36 shows an overview of the proposed system. Initially, the original network model M, composed of K+L-1 layers, is split into two sub-models: the sub-model M_1 containing the layers 0 to K-1, and the sub-model $M_{\_2}$ containing the layers K to K+L-1. The sub-model M_1 is then allocated to the edge device, where it will receive the input data from the network and perform the extraction of initial features, as illustrated in step 1 of Figure 36.

The K-1 layer of the M_1 sub-model then generates the scores. These scores are collected and if the layer K-1 is not a 2D convolutional layer, the scores are rearranged to match a matrix with *m* lines and *n* columns, as shown in step 2.

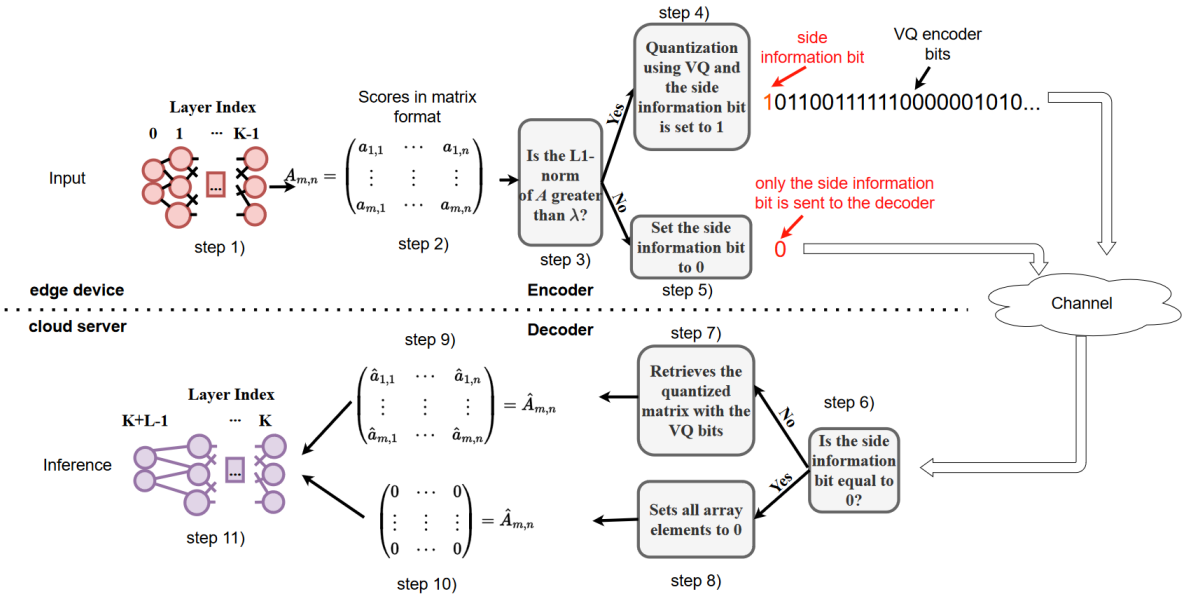


Figure 36 Split Compression Algorithm Big Picture

The scores arrays are then used as input to the encoding block, shown in step 3. The first step of the algorithm consists of checking the sum of the elements of each matrix using the L1-norm, defined by $||w||1 = \sum_{i=1}^{n} |wi|$, seeking to determine if the matrix is mostly composed of 0 values. The value of the sum obtained is then compared with an established threshold λ.

If the L1-norm is greater than the value of λ, the matrix of scores is then compressed using a two-dimensional vector quantization (VQ) with 16 clusters, thus requiring $\frac{\log2(16)}{2} = 2$ bits per scores for the quantization process. In addition to the bits used for vector quantization, a bit of side information set to 1 is used, responsible for informing the decoder that the matrix of scores has gone through the quantization process and that the next bits that will be read are the quantized bits. This process is illustrated in step 4.

On the other hand, if the L1-norm is less than or equal to the value of λ, it means that the elements contained in the array of scores have mostly values of 0 or close to 0. In this way, it is sent to the decoder only the bit of side information set to 0, as shown in step 5.

The compressed scores are then sent over a communication channel to the cloud server, where they are used as input to the decoding block. The side information bit is then checked, in order to provide the system with information on how the scores compression process took place, as shown in step 6.

If the bit value of side information is different from 0, the decoder block performs the dequantization process, collecting the next $2mn$ bits (as each matrix has *m* lines and *n* columns and were quantized using 2 bits per score), thus generating a recovered matrix of scores. If the bit value of side information is equal to 0, the decoder block generates an array of scores with elements of value 0. This process is illustrated in steps 7 to 10 of Figure 36.

The retrieved scores matrix is then used as input to the K-th layer of the M_2 sub-model, where the rest of the network inference process will take place and the final classification of the input signal will be generated.

To obtain the compression ratio of the proposed system, we need to define some variables. The first is the variable *b*, which represents the number of bits used in the VQ and which depends on the number

of clusters α and the dimension *dim* of the VQ. Both variables are related according to $b = {log2(\alpha)}/{dim}$.

For each scores matrix, one bit of side information is assigned and used by the encoder to determine if the matrix A will be quantized using VQ or if the entire matrix will be represented by the bit of side information 0. Thus, we need D bits of side information (which can be interpreted as the total number of matrices) for the system. However, if the bit of side information is equal to 1, the number of bits needed is given by *bRC*, where R represents the number of rows and C the number of columns in the matrix A.

The number of matrices that will be quantized depends on the number of times the side information bit is equal to 1, represented by the variable W. In this way, the total number of bits in the encoder output is given by the equation:

$$B = D + W\ b\ R\ C$$

and, assuming the original samples are represented with 32 bits per scores, then the compression ratio (CR) can be calculated as:

$$CR = \frac{D + W\ (b\ RC)}{32\ RC}.$$

The transmission rate of the proposed system (in bits per second) can be calculated as $\beta = BF$, considering the processing of F frames per second.

Results with MNIST dataset

This section describes the experiments and results obtained using the MNIST [LeCun98] dataset, a validated handwritten digit database widely used to train several image processing systems. The MNIST dataset was used to train a CNN model for classification of digits from 0 to 9. The network architecture used is represented in Figure 37. Two convolutional layers were used, configured with kernels of size 3 * 3 and stride 1, two ReLu layers and two pooling layers configured with the MAX function. 3 fully connected (or dense) layers were also defined.

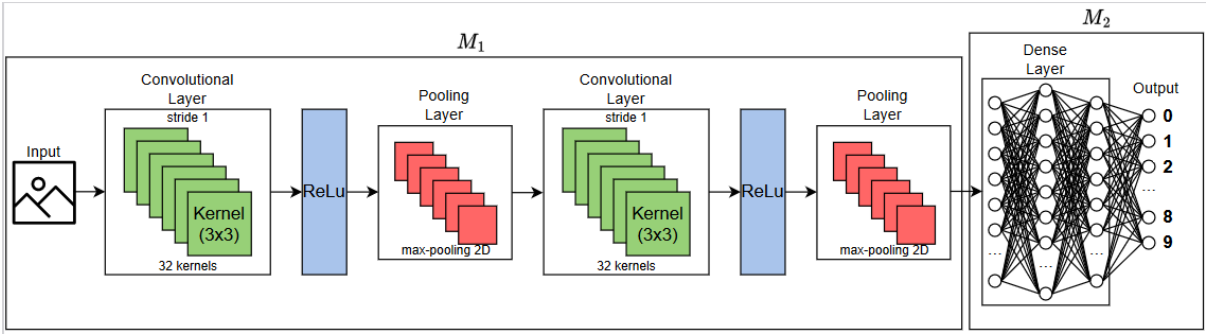


Figure 37 CNN model used in the experiments

The model M was partitioned so that the 2 blocks containing the convolutional layers, ReLu and max-pooling form the sub-mode M_1, and the dense layers form the sub-model M_2, as shows Figure 37. The output of the second layer of max-pooling (output of the sub-model M_1) is composed of 16 matrices of scores, for which it was checked if the L1 norm is greater than the value threshold λ.

4.4.2 Results

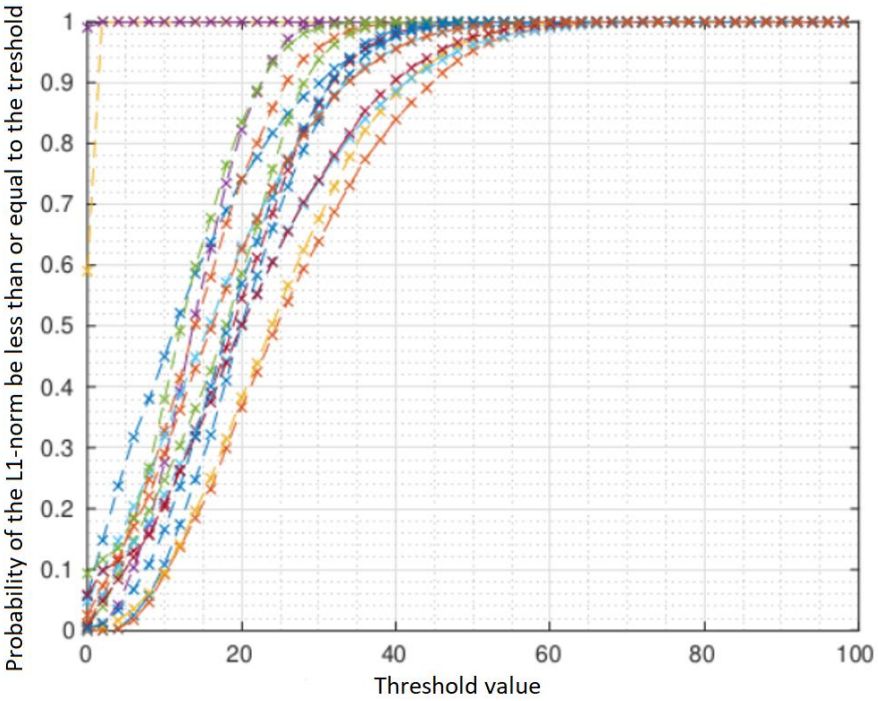


Figure 38 CDF of different layers of th neural network. Each line represents the probability of the L1-norm of the respective CNN kernel output be less than or equal to threshold

To understand how the variation of the threshold value λ impacts the number of matrices that will be zeroed, the Cumulative Distribution Function (CDF) [deisenroth202] of the scores is calculated, which evaluates the probability of the value of the L1 norm of each array being less than or equal to the threshold λ. The results obtained after calculating the CDF are shown in Figure 38, where each line represents a matrix of scores. It is possible to see that for values of λ greater than 60, the probability of an array being zeroed out is approximately equal to 100%. Although choosing λ values greater than 60 results in a high compression ratio, it should be noted that at the end of the model classification process on the cloud server, the system's accuracy can behave close to that of a random.

Thus, the experiments were performed by changing the threshold value λ from 0 to 60. The matrices were then forwarded to the second stage of the coding block where, according to the value of the L1 norm, they were quantized using a VQ of 16 clusters and therefore needing only 2 bits per sample, or they were zeroed.

The results are shown in Figure 39. The transmission rate in bits per sample is shown in blue and the model accuracy in percentage (%) is shown in black. The rate using the Sparse Exponential Golomb compression algorithm, proposed in [Georgiadis19], is also shown by the dashed blue line. As can be seen, for values of λ less than 10, the technique of zeroing the matrices reduces the rate without significantly affecting the model's accuracy.

However, for values of λ greater than 10, the loss of accuracy starts to be more noticeable, affecting the performance of the model. In this way, it is possible to infer that there are certain matrices of scores that can be zeroed without affecting the performance of the model. It is also possible to observe that, for values of the threshold λ close to 60, the performance of the model decreases considerably because, as mentioned previously in Figure 39, from this value almost 100% of the matrices are zeroed.

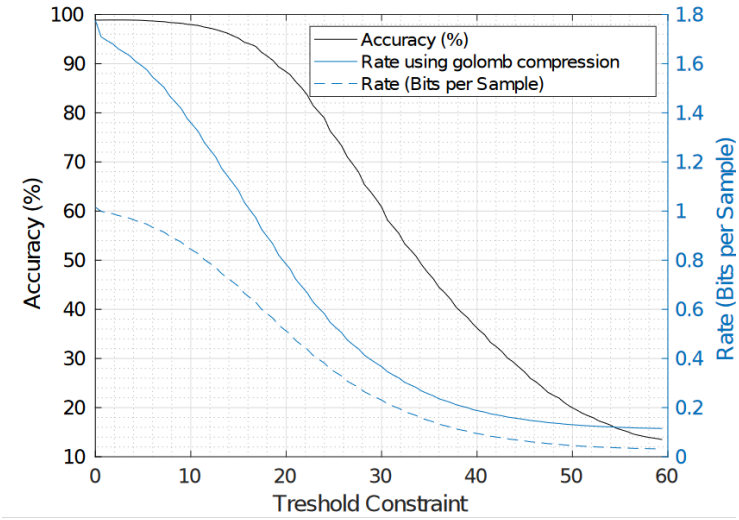


Figure 39 Accuracy (%) and Rate (bits per sample) results

The accuracy of the model presented in [Georgiadis19] was 97.93%. It is important to note that the main objective of the method proposed in [Georgiadis19accelerating] was to increase the classification speed by increasing the sparsity of the scores and reduce the model size through quantization and compression. Using a variation of the Exponential Golomb~ [teuhola1978compression] compression algorithm, the rate was reduced to 1.25 bits per sample. The model proposed in this work presented the same accuracy of 97.93%, but with a rate of only 0.82 bits per sample, thus reducing by 33.79% compared to the work in [Georgiadis19].

The following picture describes how the accuracy (%), the rate (kbps) and the operational cost (GIGA FLOPS) changes at each split configuration. For this scenario it is used the CIFAR-10 dataset. The neural network used in this experiment it was the ResNet-18. Sometimes if the network resources are limited, a split to save more resources in the Far EDGE can be used with the drawback of the rate increase. The best split decision depends on the available network scenario and how much resources can be used to run the desired split.

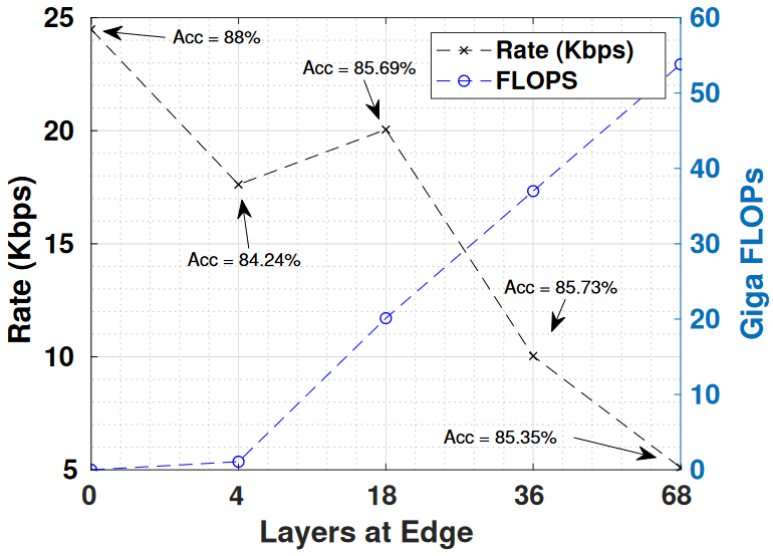


Figure 40 Rate (Kbps) and Computational cost for each different split

4.4.3 How the proposed method can be mapped to the AI@EDGE?

The proposed method can be deployed in the AI@EDGE architecture by deploying the models and sub-models into AIFs. For example, consider the model used in this Section described in Figure 37. One option is to deploy the entire model inside one AIF. This scenario is shown in Figure 41 a). However, we can split the model into two sub-models called here *"sub-model A"* which contains the first layers and the *"sub-model B"* containing the remaining layers. The split is done to minimize the rate (Kbps) between the Far EDGE and the Near EDGE while not affecting the accuracy in a prohibitive way. The Split scenario is described in in Figure 41 b) and in Figure 41 c) and we can use the interface if3 to send the data between the two AIFs.

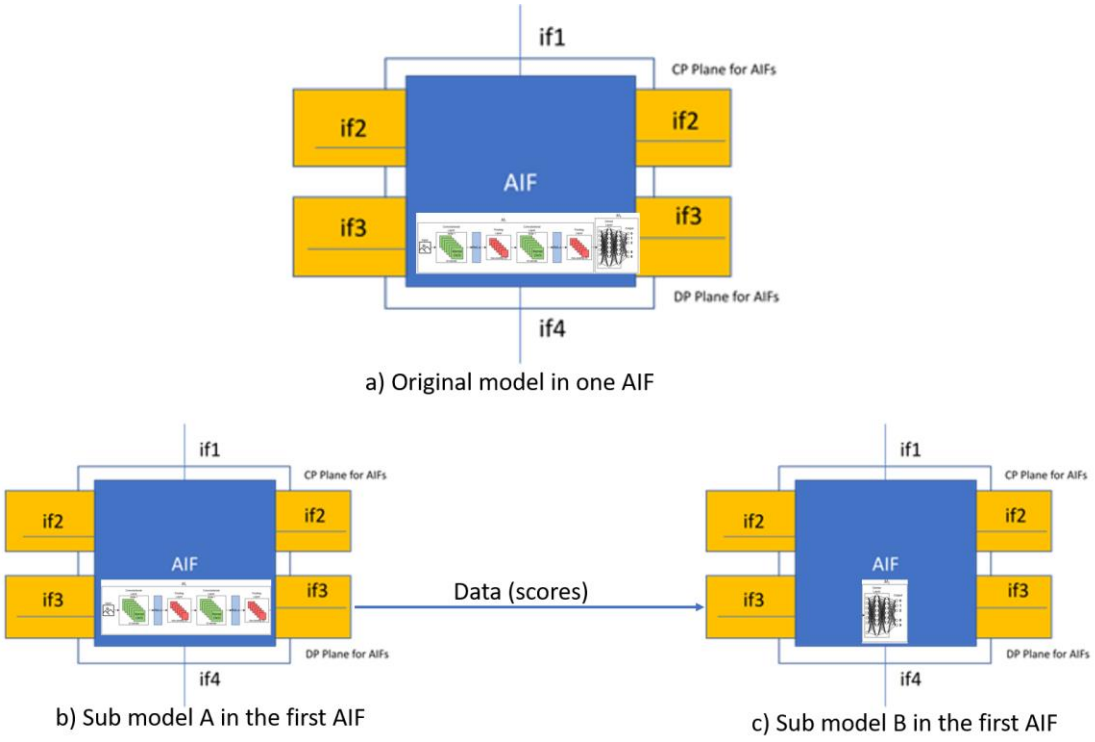


Figure 41 Model and sub-models deployed in AIFs

5. Methods for predicting/forecasting future performance

In an era where machine learning and wireless networks are increasingly intersecting, the demand for more efficient and intelligent systems is paramount. This section is segmented into two pivotal subsections, each addressing a unique yet interrelated facet of this interdisciplinary field. The first subsection delves into the integration of machine learning with wireless communications, specifically focusing on the optimization of distributed and federated learning systems in wireless environments. It examines how predictive radio awareness can be achieved through advanced machine learning techniques, thereby enhancing system performance. The second subsection shifts the lens to Mobile Edge Computing (MEC), a revolutionary paradigm for low-latency network services. Here, the focus is on how machine learning can predict key performance indicators and user mobility to optimize service migration and radio handoffs in a mobile edge computing context. Together, these sections offer a comprehensive view into the cutting-edge methodologies being developed to advance the synergies between machine learning and wireless networking technologies.

5.1 Machine learning approaches for predictive radio awareness for distributed and federated learning

Advanced machine-learning approaches, such as federated learning and distributed learning, need to exchange data and model update information frequently, including the distribution of an universal model to the participant nodes, the transmission of a partially trained model to other nodes, and essential control signalling and management messages. The origin of federated learning and distributed learning come from the Internet, where the default connectivity among the learning nodes is implemented with wired links such as optic fiber and local-area network (LAN). The properties of a wired link, including transmission rate, delay, jitter, packet error rate, and availability, are regarded as deterministic. When these learning methods migrate to wireless/mobile networks, the learning nodes suffer from a dynamic environment. The link performance of wireless transmission heavily depends on the quality of wireless channels. Since a wireless channel is highly dynamic and time-varying, a transmitter always prefers to adjust their transmission parameters for better performance. With the assistance of channel state information (CSI), the transmitter is able to adaptively choose its parameters such as the transmit power, constellation size, coding rate, transmit antenna, and MIMO precoding codeword to achieve great performance. If we can track and predict the CSI of all the links in a federated/distributed learning system, we can achieve better communication performance to guarantee the efficient operation of the intelligent system. Conversely, the system performance will be substantially degraded due to the lack of accurate CSI and then the wrong selection of transmission parameters.

Through modelling a wireless channel into a set of radio propagation parameters, two statistical prediction approaches - Auto-Regressive (AR) and Parametric Model (PM) - have been proposed in the literature [DuelHallen07]. However, the modelling is fossilized, leading to a gap between these models and real channels, and - in addition - the parameter estimation process relying on complex algorithms such as MUSIC and ESPRIT is tedious, harmed its applicability in practical systems. As a data-driven technique, neural networks can avoid parameter estimation thanks to their data-driven nature, and therefore attract interest from researchers in the field of channel prediction. Making use of its capability on time-series prediction, a number of CSI predictors mostly based on deep neural networks were proposed [Jiang19]. However, the current deep learning-based prediction approaches focus on the single-user mobile environment, which does not fit with the multiple learning terminals working simultaneously in a distributed/federated-learning system. In AI@EDGE project, therefore, we focus on multi-user prediction for the setup of federated learning and distributed learning, which can train and predict the CSI of multiple users simultaneously.

5.1.1 ML Algorithms Description

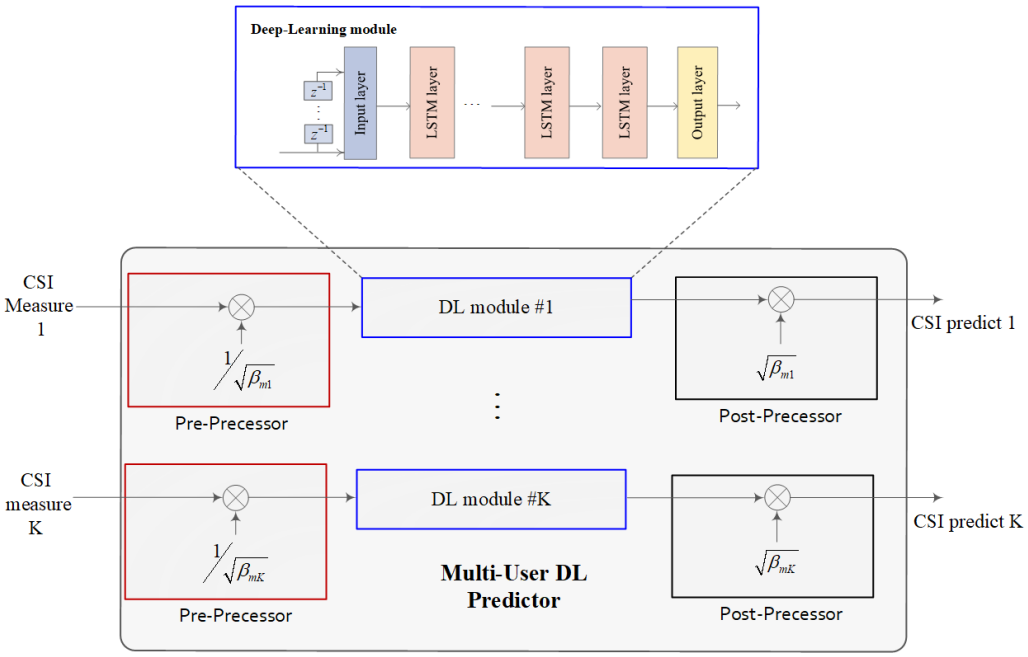


Figure 42 The structure of multi-user CSI predictor

The prior model-based and data-driven methods focus on single-user scenarios where only small-scale fading is considered with the assumption that channel gains follow standard complex Gaussian distribution with a normalized channel gain. However, a practical wireless system must accommodate many users and simultaneously serve multiple active users. Due to the near-far effect, distance-dependent large-scale fading among multiple users might differ by several orders of magnitude or tens of decibels (dB). Different links among ML nodes have different propagation distances, where previous single-user channel prediction cannot be directly applied [Jiang20]. We therefore design a multi-user predictor consisting of a bank of user-specific deep-learning predictive modules to deal with such power-gain difference. Figure 42 shows the structure of the multi-user predictor. The estimated CSI is first normalized by multiplexing a factor of $1/\sqrt{\beta_{mk}}$, which denotes the effect of large-scale fading, where m standards for the mth transmitter for the learning/computing node, while k means the kth users connecting to this learning/computing node. Each deep learning (DL) module generally consists of an input layer, multiple hidden layers, and an output layer, can perform single-user prediction independently. Feeding the CSI measure for a user into the input feed-forward layer to generates the response to the activation of the input layer and then go through the first hidden layer. The structure of a hidden layer built by long short-term memory (LSTM) or gated recurrent unit (GRU), where the detail structure of a LSTM hidden layer is illustrated in Figure 43. The activation goes through the network until the output layer gets a predicted value, which is multiplexed by a factor of $\sqrt{\beta_{mk}}$.

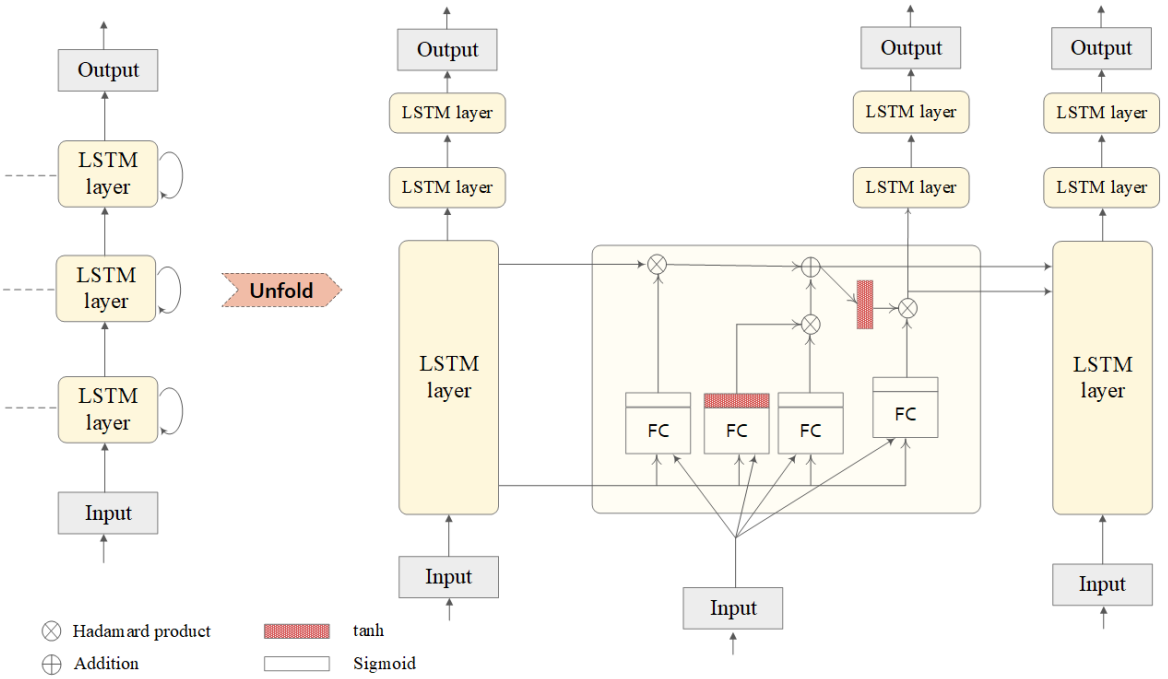


Figure 43 The structure of a deep-learning prediction model.

Differing from feedforward neural networks, recurrent neural network (RNN) has self-connections, feeding the activation from the previous time step back to the network as input for the current time step. As a class of classical neural networks, RNN is good at processing data sequences through storing indefinite historical information in its internal state, exhibiting great potential in time-series prediction. However, it suffers from the gradient exploding and vanishing problems with the gradient-based back-propagation through time training technique, where a back-propagated error signal is apt to be very large, leading to oscillating weights, or tends to zero that implies a prohibitively long training time or training does not work at all. To this end, Hochreiter and Schmidhuber [Hochreiter97] designed an elegant RNN structure - long short-term memory - in 1997. The key innovation of LSTM to deal with long-term dependency is the introduction of special units called memory cells in the recurrent hidden layer and multiplicative gates that regulate the information flow. In the original structure of LSTM, each memory block contains two gates: an input gate protecting the memory contents stored in the cell from perturbation by irrelevant interference, and an output gate that controls the extent to which the memory information applied to generate the output activation. To address a weakness of LSTM, namely the internal state grows indefinitely and eventually cause the network to break down when processing continual input streams that are not segmented into subsequence, a forget gate is added. It scales the internal state of the memory cell before cycling back through self-recurrent connections. Although its history is not long, LSTM has been applied successfully to sequence prediction and labelling tasks. It has already gotten the state-of-the-art technological results in many fields such as machine translation, speech recognition, and handwriting recognition, and has also achieved a great commercial success, justified by many unprecedented intelligent services such as Google Translate and Apple iPhone Siri.

Like a deep RNN consisting of multiple recurrent hidden layers, a deep LSTM network is built by stacking multiple LSTM layers. Without loss of generality, Figure 43 shows an example of a deep LSTM network that consists of an input layer, three hidden layers, and an output layer. At an arbitrary time step, as illustrated in the left part of this figure, a data vector goes through the input feedforward layer to get an outcome, which is the activation for memory cells in the first hidden layer. Along with the recurrent unit feeding back from the previous time step, a new outcome is generated and then forwarded to the second hidden layer. This recursive process continues until the output layer. Unrolling the network through time, as illustrated in the right part of this figure, the memory block at the first

hidden layer has two internal states at time step, i.e., the short-term state and the long-term state. Traversing the memory cells from the left to the right, it first throws away some old memories at the forget gate, integrates new information selected by the input gate, and then sends out as the current long-term state. The input vector and the previous short-term memory are fed into four different fully connected layer, generating the activation vectors of gates. Dropping some old memories at the forget gate, and adding some new information selected from current memory input, the previous long-term memory is thus transformed. Further, the signal passes through the activation function and then is filtered by the output gate to produce the current short-term memory, as well as the output for this memory block.

Up-to-date knowledge of channel properties can greatly enhance wireless communication by allowing sophisticated rate adaptation and resource allocation. Often, the adaptation is performed at the transmitter based on the measured CSI piggybacked from the receiver. In a dynamic environment, such as those where distributed/federated learning carried in vehicular units, the CSI may already be outdated by the time it reaches the sender. Therefore, we look deep into the opportunities for predicting CSI in a dynamic network setting with moving devices. On the terminal side, the predictor is only responsible for predicting the downlink CSI for this specific user, where the multi-user predictor can be simplified to a single-user predictor. Figure 44 shows the possible structure of a receiver integrated a DL-based channel predictor that mainly consists of an input layer, an output layer, and multiple hidden layers. The first hidden layer is opened to detail the internal structure of an LSTM memory block and its information flow. To remain historical channel information, a tapped-delay line is applied to form a series of consecutive CSI samples for the input layer. The predictor is inserted between the channel estimator and transmission parameter selector, transforming measured CSI to predicted CSI transparently without any other modifications for a learning system.

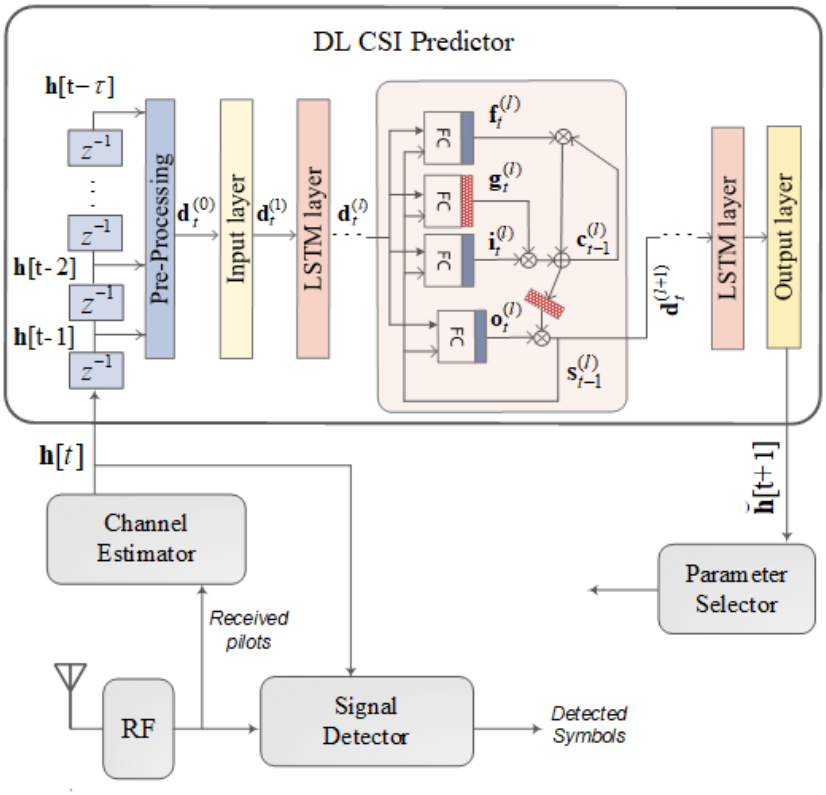


Figure 44 Illustration of the integration of a deep-learning predictor with the transceiver

5.1.2 Training Data Acquisition from Real Measurement

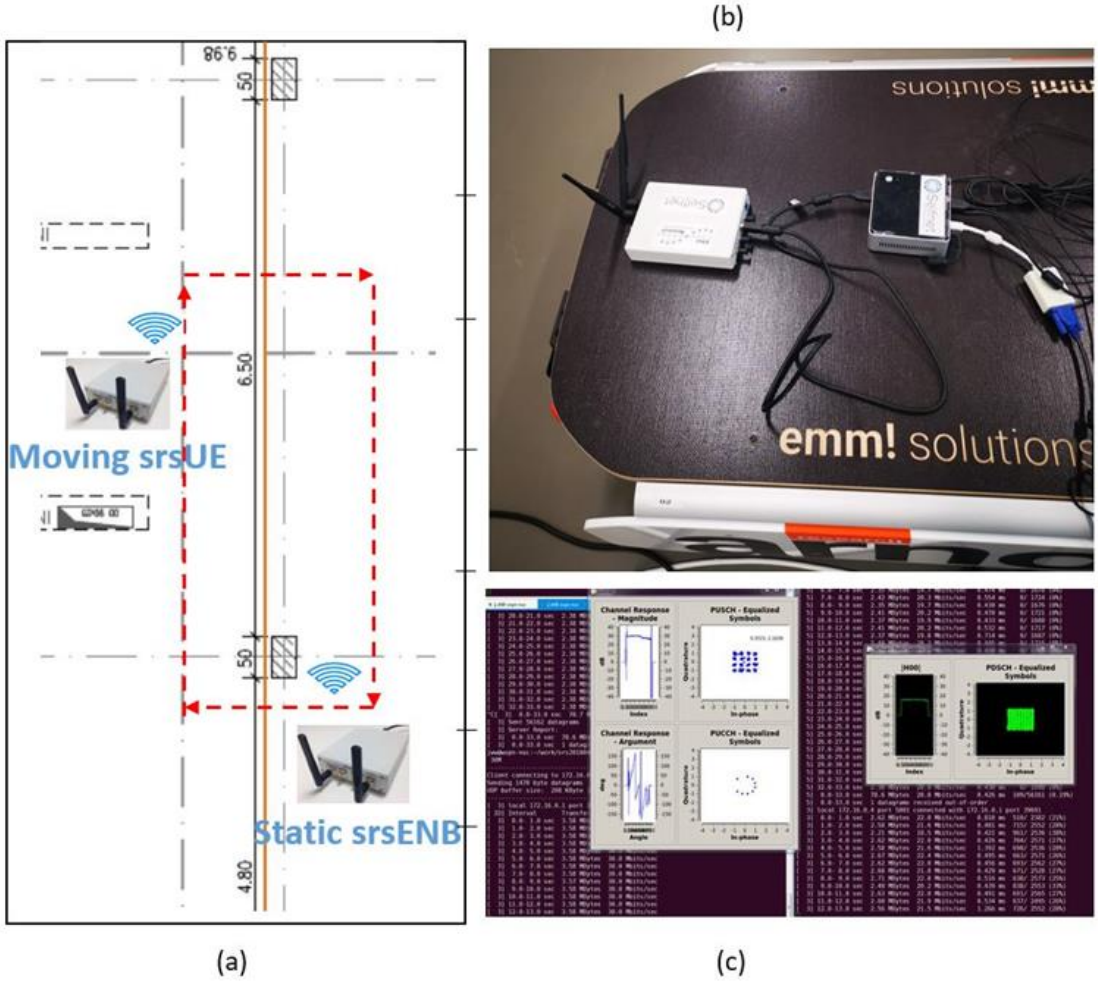


Figure 45 Illustration of channel measurement hardware setup

An overview of the scenario is shown in Figure 45. There are two computers running srsENB with srsEPC, and srsUE respectively, which are full implementation of software-defined radio applications of srsRAN. Both computers should have USB 3.0 ports for communication with Universal Software Radio Peripheral (USRP) B210. To observe a moving node in an indoor scenario, automatic guided vehicles (AGVs) with different speeds communicate with the BSs using different bands. The AGVs need to predict the rapid changes of CSI in real-time, perform mobile edge computing (MEC), and work with BSs for the adaptive transmission scheme to improve the efficiency and throughput of wireless communication systems. Specifically, we consider the SNR in the communication channel as a prediction target. Then the AGVs can use the predicted SNR to switch proper modulation modes to improve communication quality. The BSs can also adapt the transmission rate according to the predicted SNR. For an optimal simulation of AGV communication in indoor networks, we put our platform in a roomy demo laboratory (20mx10m), where there are also machines, pillars, tables, and walls with different materials, that can provide different types of reflections and create randomness as much as possible. Figure 45 (a) is a top view of the lab. There are two USRP devices, which connect to the srsUE and the srsENB machine respectively. We keep the srsENB device static in the corner of the lab. The srsUE device is then placed on an AGV, as shown in Figure 45 (b), which runs along the route as shown by the red dotted line in Figure 45 (a). So, we have created a relatively dynamic communication scenario. In srsRAN, on both sides of UE and eNB, a tracer for channel state information is implemented, so we

can easily monitor the data like reference signal received power (RSRP), received signal strength indicator (RSSI), SNR, bit rate, etc., as the system is running. Figure 45 (c) shows the window of running status and plots of some of the channel states.

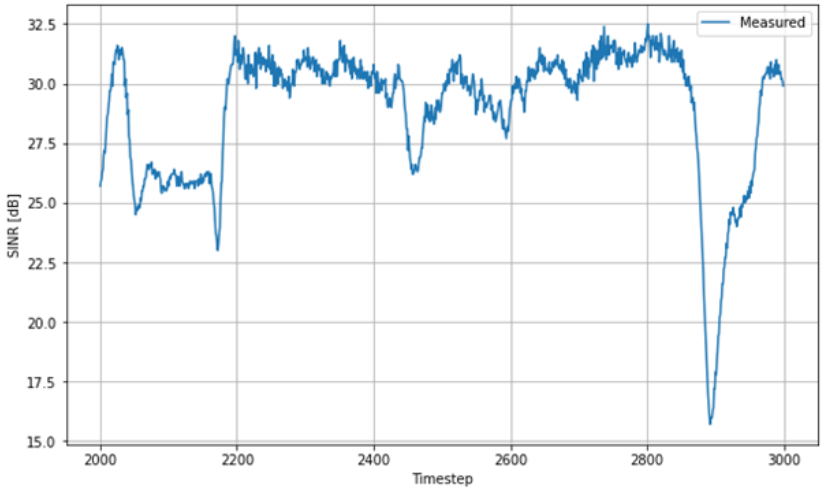


Figure 46 An exemplified plot of SNR data with timestep of every 10ms

5.1.3 Results and Analysis

We use the 12000 samples of the data for training and the remaining 2000 samples for testing, as exemplified in Figure 46. We have taken 32 filters each with 5 Kernels for CNN and 64 units for LSTM as an initial parameter configuration. Starting from an initial state with random values, the weights and biases are iteratively updated by Adam optimizer. The learning rate of 0.001 is derived from callbacks of the learning rate scheduler during training. We use the mean absolute error (MAE) to calculate the error between the predicted value and true value. As a baseline, we have taken 10-time steps of SNR observations to predict 1-time step SNR ahead. The same parameters are taken as above with a maximum epoch of 50 and 200 steps of fitting per epoch, we can achieve an MAE of 0.2661dB on the 2000 samples of testing data. Figure 47 depicts the true and predicted SNR of 500 samples of test data. It can be seen from this figure that the predicted SNR is almost the same as the true value, which shows that the proposed model is very effective for predicting dynamic changing channels.

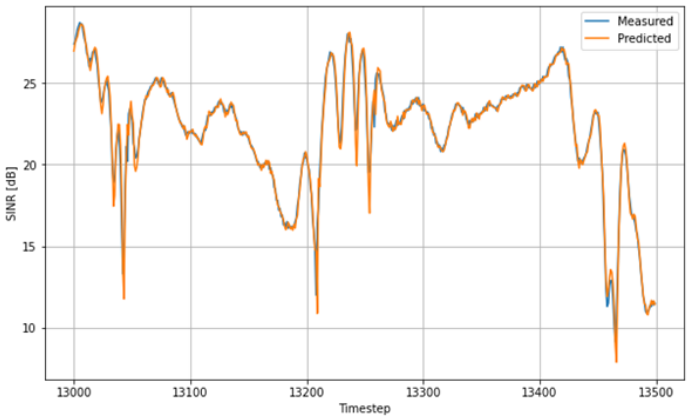


Figure 47 Plot of SNR prediction results of 500 samples from test data

We have taken the 5-time steps prediction from 10-time steps observations as a multi-steps prediction. By using the same structure and parameter of the algorithm, we can achieve MAE of 0.9973dB for the

fifth step of each multi-steps prediction on the 2000 samples of testing data. The result is shown in Figure 48. Compared with the result of single-step prediction, the multi-steps prediction shows worse prediction accuracy. But the error is still under 1dB, which is acceptable for system operations. We have also tried to change the hidden units and layers of the algorithms, Figure 49, where we change the number of units in LSTM layer. By increasing the LSTM units, only a slight improvement of accuracy can be seen, even though two layers of LSTM are used. Therefore, the number of hidden units has a minimal effect on the prediction performance when the training data is large enough. As a result, choosing a small number of hidden units and historical observations is possible and should be used for channel prediction to reduce the computation complexity needed for neural network training. Besides, if we compare the MAE of the predictor using one layer CNN with 32 filters and one layer LSTM with 32 units, and another predictor using two layers of LSTM, both with 32 units. The former gets the MAE of 1.0309dB, as shown in Figure 49, while the latter achieves the MAE of 1.0388dB, which has verified that the LSTM predictor with CNN layer for data prepossessing outperforms the pure LSTM predictor.

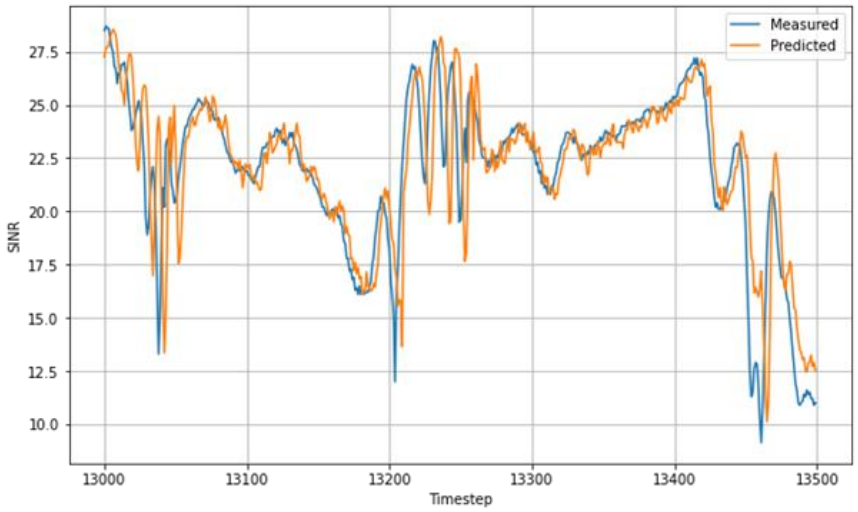


Figure 48 Plot of the results of each fifth step of 5-steps SNR prediction of 500 samples from test data

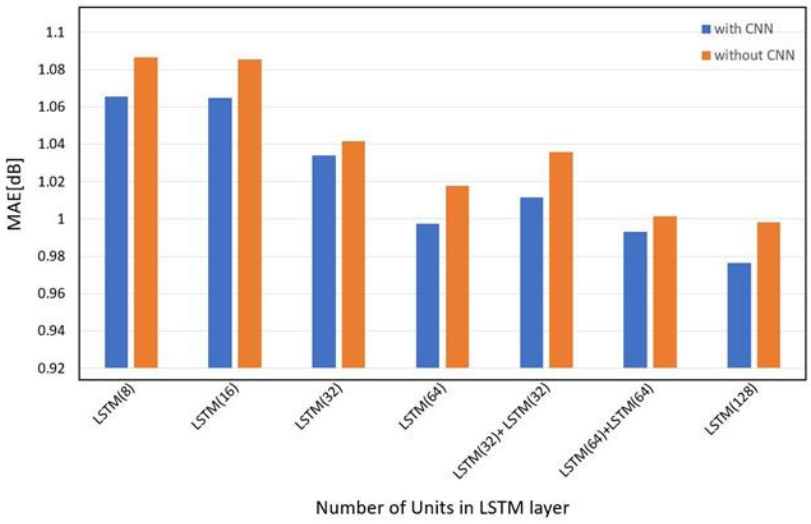


Figure 49 MAE results of using different layer settings for 5 steps ahead prediction

5.2 Forecasting of measurable performance KPIs, capturing contextual RAN low-level and network layer data at the edge for user mobility

Mobile Edge Computing (MEC) emerges as a new paradigm, to support latency sensitive network services by providing computational resources in the close proximity of end users. Network Services can be deployed as AIF chains at the MEC Edge Servers. Mobility management has unique difficulties when MEC is deployed with a BS. A MEC can only be accessible inside of its BS's coverage region, to start. Therefore, upon leaving the cell coverage area, a user must execute both a radio handoff (HO) and a service migration.

To address this issue, we propose an ML algorithm that predicts the association of a user to a specific base station based on user mobility patterns. The algorithm is used by the Integer Linear Programming (ILP) algorithm to decide migration of AIFs from one MEC to another based on whether the user is going to do radio handover or not.

5.2.1 Algorithm and dataset description

For this research, we utilized the PyTorch machine learning framework to implement a Multi-layer Perceptron (MLP) algorithm. The network has an input layer of size 8, a varying number of layers with 1024 hidden units and a single output layer. We included batch normalization between the layers to improve the performance of the model and a final dropout layer. We tested the model with 2, 4 and 6 numbers of layers. The final layer of the network tries to predict whether a user remains or not in the same cell the next iteration, so the final output is a float in the range between 0 and 1. Figure 50 shows a schematic of the model architecture.

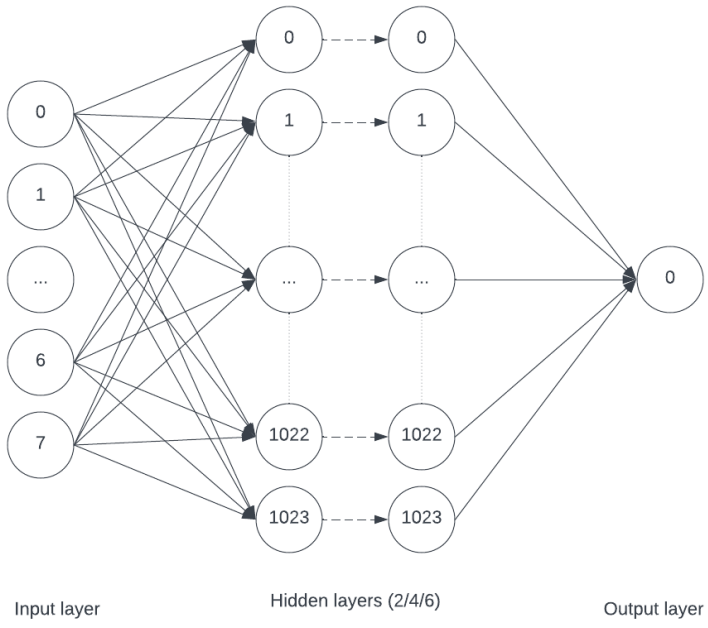


Figure 50 NN architecture of the user mobility prediction model

The dataset we used was produced in NS3. NS3 is a discrete-event network simulator used for simulating and analysing the performance of communication networks. It is an open-source software that provides a platform for researchers to develop and test new network protocols and algorithms. With NS3 we collected datasets of 12 base-stations for 27 runs, which were later separated per base-station.

We split the dataset in 23 runs for the training set and 4 runs for the test set. Data preprocessing was done to separate the data of corresponding base-stations based on user association. To test the variation in the model accuracy, we considered different amounts of temporal data context by including up to 3 previous observations in the input data. This allowed the model to capture the eventual temporal dependencies in the user mobility patterns. Table 4 shows some example rows from the raw dataset we used.

Table 4 Example NS3 dataset with the KPIs for the user mobility prediction

Node	Tagg_start	mean-Dl-rsrp	mean-Dl-sinr	mean-Dl-mcs	mean-Dl-thput	predCellId
4	332	-73.942922	2.929047	8.000000	665469.00	13
4	336	-74.393384	2.529483	8.000000	668556.00	13
4	340	-74.855070	2.234082	7.976111	662310.00	13
4	340	-74.855070	2.234082	7.976111	662310.00	13
4	348	-75.778451	1.627978	6.128863	499502.25	13
4	352	-76.230553	1.675687	6.424540	573598.00	13

5.2.2 Results and analysis

The MLP training function has several parameters that can be adjusted to optimize the performance of the model, some related to the model structure, others to the dataset:

- n_hidden parameter specifies the number of hidden units in each layer of the MLP.
- n_layers parameter determines the number of layers in the MLP.
- learning_rate parameter controls the step size used in the optimization algorithm.
- epoch parameter specify the number of training epoch.
- max_waiting parameter specifies the maximum number of epochs to wait for an improvement in the validation loss before stopping the training process.
- batch_size parameter specifies the number of samples used in each iteration of the training process.
- context parameter determines the number of previous observations included in the input data.
- balance parameter controls whether the training data is balanced or not.

The experiments we conducted involved especially the number of layers and the data context. We maintained the number of neurons in the hidden layers to the constant number of 1024. The learning rate was set to 0.0001, the epochs to 200 and the max_waiting to 10. The loss function used was the Binary Cross Entropy because we needed to discriminate if a user stays or not in the same signal cell, so we had only two possible outputs.

Table 5 reports the total accuracy on the test set and the F1 score for the "transition class".

Table 5 Accuracy and F1(0) Score varying observation

Context	0		1		2		3	
No. of Layers	F1(0)	Accuracy	F1(0)	Accuracy	F1(0)	Accuracy	F1(0)	Accuracy
2	0.23	93.70	0.51	94.70	0.49	94.40	-	-
4	0.23	93.80	0.50	94.60	0.48	94.60	0.49	94.20
6	0.27	93.80	0.49	94.60	0.50	94.60	0.50	94.20

Figure 51 and Figure 52 represent the results according to the f1 score and the accuracy. As it is shown in the charts, the best results are achieved with at least a small-time context within the training data. It seems that the number of layers is not so relevant, but the accuracy and f1 score perform well with a 6 layers structure.

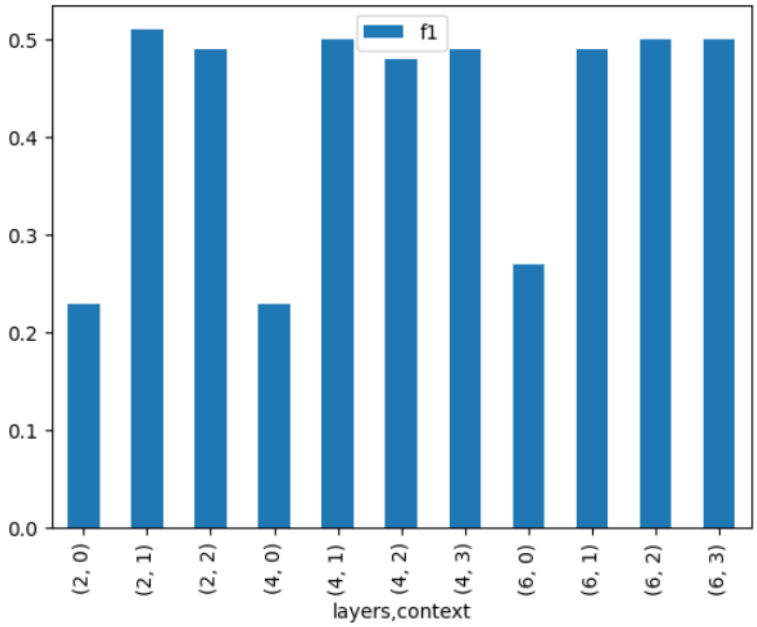


Figure 51 F1 distribution for the user mobility prediction model

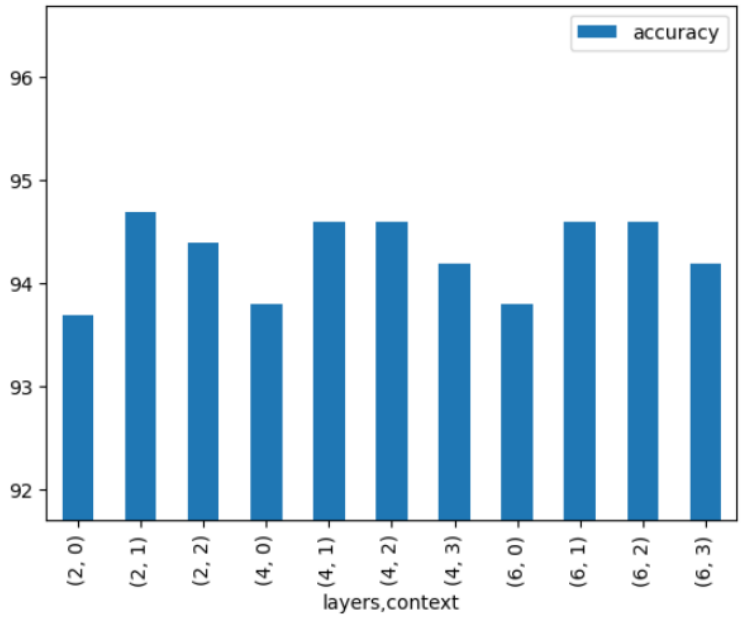


Figure 52 Accuracy distribution for the user mobility prediction model

The algorithm has been implemented and packaged as two AIFs: one for training and one for prediction. The first AIF reads user-supplied data from a datalake (e.g., S3-compatible storage) and performs the training of the MLP algorithm, persisting the resulting model on the same storage. The AIF downloads the model from the storage and exposes it as a REST service.

5.2.3 Summary

To maintain a positive customer experience, network adjustments can be planned and put into place in time by being aware of the exact breaking points in the network well in advance. In this study, the forecasted performance KPIs based on data captured at RAN and Network Layer at the edge, are meant to be used for decision making regarding VNFs migration or re-instantiating.

6. Methods for monitoring and preparing data

In the following sections, we look at two methods that could be utilized in data monitoring. In the first part, we explore a framework for anomaly detection tailored specifically to the challenges posed by highly distributed 5G&B and edge computing environments. To address these complex requirements, a Federated Learning based framework for anomaly detection approach ideally suited for processing data dispersed across multiple devices and locations within the network. The second part deals with methods for the development of datasets that enable the machine learning base algorithm. These datasets have been thoroughly constructed to act as benchmark tools for the evaluation and comparison of data-driven algorithms.

6.1 Anomalous event detection with federated learning

Anomaly-detection modules are crucial elements within the analysis subsystems of the closed-loop in the modern network infrastructure. Their primary function is to trigger decision-making plans to reconfigure the infrastructure stack, making it resilient against the diverse set of impairments that occur during system operations. To accomplish this, we investigate the application of artificial intelligence and machine learning techniques to support these anomaly-detection subsystems within the closed-loop. At the state of the art, the SYRROCA (SYstem Radiography and ROot Cause Analysis) framework was recently proposed as an anomaly detection system for softwarized infrastructures [Diamanti22]. It has a machine learning engine to monitor the infrastructure, making use of LSTM (Long-Short Term Memory) autoencoders for spotting anomalies from a high number of time-series features collected at multiple network and system subsystems, in a centralized way. Although the SYRROCA framework proves its algorithmic capacity to spot anomalies and support fine-grained root cause analysis, its actual implementability to 5G and edge computing environments, which can be highly distributed and require stringent latency and resource efficiency guarantees, does call for a form of decentralization that we describe in this work. More precisely, in such environments, data is usually generated at multiple locations, and often needs to be processed in real-time while ensuring its privacy. To do so, we propose to use federated learning which is an innovative approach to machine learning designed to address the challenges of processing data that is distributed across multiple devices or locations around the network.

6.1.1 Implementation of anomaly detection

To customize the SYRROCA framework for the 5G environment, we leverage on a multi-layer LSTM Neural Network (NN) because such systems are optimized to learn both short and long-time correlations and seasonality in time-series, hence allowing to capture the behaviour of complex multivariate sequences. More precisely, LSTM NNs use encoders and decoders [Hochreiter97] in order to form a deep autoencoder to train metrics. An autoencoder is composed of two blocks:

- **Encoder:** it compresses the original high-dimension input into the low-dimension latent space.
- **Decoder:** it reconstructs input from the latent space, likely with larger output layers.

In fact, LSTM autoencoders are trained to reconstruct the normal working conditions of a given system, i.e., based on a representation of the normal operation of the system [ChiPhung22], with minimum errors. When they fail in reconstructing these nominal conditions, the error increases. This occurrence can be translated as an anomaly.

To adapt the SYRROCA framework to the 5G environment [BinRuba22] we group metrics from a reference 5G infrastructure into component-specific groups, i.e., CPU, Memory, Network, Radio and Disk, at different infrastructure levels, namely physical level, virtual level and Radio Access Network (RAN) level. Each sub-dataset is fed to a dedicated autoencoder, where all autoencoders share the same architecture. This reduces the complexity of the architecture and the training time.

In a distributed environment, these autoencoders are deployed in different nodes and trained collectively by means of distributed training. In our case, we employed the Federated learning paradigm to produce a learned common model that aggregates the knowledge obtained from all nodes participating in the process.

6.1.2 FL Deployment of the Anomaly Detection Framework

In a distributed environment, these autoencoders are deployed in different nodes and trained collectively by means of distributed training. In our case, we employed the Federated learning paradigm to produce a learned common model that aggregates the knowledge obtained from all nodes participating in the process.

Federated Learning (FL) [McMahan16] performs a distributed training of AIML models at the edge network without the need for data collection and aggregation at the central cloud. In an FL environment, the whole process is governed by a central server, which is responsible for the initialization and assignment of hyper-parameters. The two main steps consist of training local models at the edge nodes and performing a global aggregation of the resulting parameters at the FL server.

Our FL framework is implemented using Tensorflow Keras library for building and training the ML models at the client's side. For what concerns the aggregation algorithm at the server side, we use Federated Averaging approach (also called FedAvg); FedAvg, besides being computationally efficient, is sufficient for our data that is not highly heterogeneous (high heterogeneity may require a more sophisticated aggregation algorithm). Nonetheless, other aggregation algorithms may be further considered. The framework uses a Remote Procedure Call (RPC) Python library called gRPC to implement the communication system between the FL server and clients AIFs deployed across the network infrastructure. Built on top of HTTP/2, gRPC makes use of features such as bidirectional streaming and builtin Transport Layer Security (TLS). gRPC uses a fast, more compact, binary serialization format protocol called Protocol Buffers (protobuf). As gRPC can run procedures and code routines on remote machines by serializing and marshalling objects, the exchange of training functions, ML models and initialization parameters between the central server and the participating client AIFs, are treated as objects and transmitted as function parameters.

We adapt the general FL process to the anomaly detection framework along the following steps:

- Client registration: the process starts by the FL client's registration with the FL server to be able to participate in the training task;
- Parameters initialization: the hyperparameters and the LSTM autoencoders are set at the server side;
- Parameters sharing: in this step, the server sends the LSTM autoencoder model, the hyper-parameters and the process configuration to the client AIFs;
- Model training: each client trains the autoencoder model with its own available data;
- Parameter aggregation: after each round of training, the model parameters are exchanged with the server and the global model is updated, either for inference or for a new round of training.
- We stop the FL training after a fixed number of rounds, depending on the desired accuracy of the inference anomaly detection model.

6.1.3 Data preprocessing and autoencoders training

As a deployment strategy of the federated learning framework, we adopt an over-the-top edge-computing friendly approach as described in Figure 53, where FL nodes are deployed as servers possibly detached from the actual sources of data yet supposed to be in strong proximity with the sources.

This implies the adoption of a data pipe-lining system able to distribute data or make data available at FL nodes in runtime. We used the 5G3E (5G End-to-End Emulation) [ChiPhung22] dataset, which is implemented at the CNAM CEDIRC lab. This dataset spans a duration of 15 days, with 14 days representing normal operating conditions, which were specifically used for training purposes. In the 15th day, the infrastructure is injected with abnormal functional conditions that used to produce stress testing dataset used for experimenting the framework.

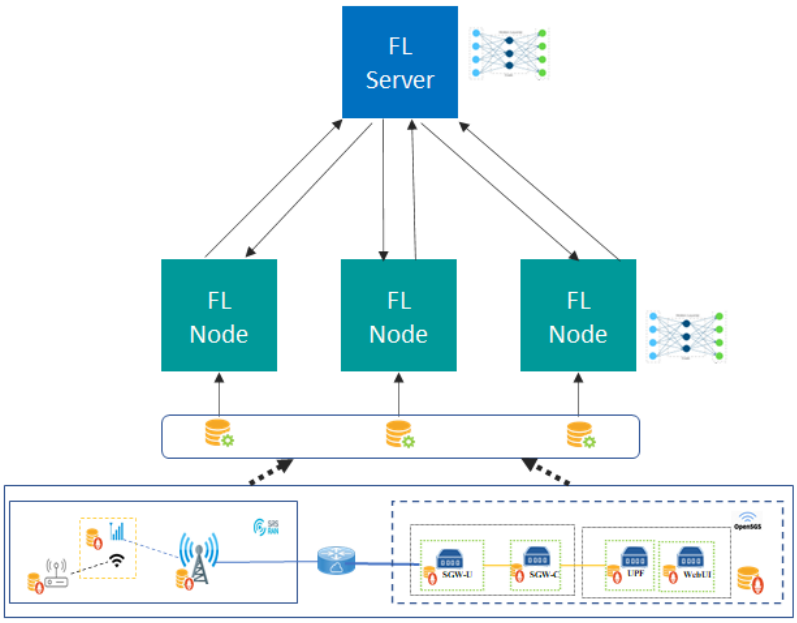


Figure 53 Anomaly Detection Framework deployment

During the training phase, data is fed as input to the autoencoders in order to learn how to map the input metrics to a compact representation through the latent space of the autoencoder. In our implementation each autoencoder is responsible for learning the nominal state of the system based on one type of resource. To do so, a preprocessing step of the dataset is needed.

Data preprocessing involves two main steps. Firstly, the data is grouped based on the type of resource it represents. This step entails dividing the overall dataset into smaller sub-datasets that contain similar types of features (e.g., a sub-dataset could be for CPU-related features).

Secondly, the input data is reshaped to fit the requirements of LSTM networks. The reshaping process involves organizing the data into a structure of "**`number of samples x number of steps x number of features`**". Where, the number of steps refers to a sequence of past observations that are provided as input to map to the output observation.

When data is pre-processed, features are grouped according to the resource type. Each sub-dataset is fed to a dedicated autoencoder, where all autoencoders share the same architecture. This reduces the complexity of the architecture and the training time.

6.1.4 Anomaly detection results

After the training step, we have a common global model that captured the daily normal state of the infrastructure. To evaluate this FL model, we use the 5G3E [ChiPhung22] test set, produced by injecting different anomalies; the anomaly injection is done with different intensities to test the sensitivity of our model to anomaly severity. The main goal is to be able to detect anomalous events which can affect

system performance, regardless of the anomaly behavior type. demonstration we show how the injected anomalies affect the following system performance:

CPU load: to test the effect of CPU overload on the system.

Bandwidth limitation: to show what effect a limited bandwidth has on the running state of system resources.

Packet delivery: different packet loss rates are considered.

Table 6 Anomaly Detection Metrics Evaluation for CPU Resource in Physical Layer

Client #	Test	Acuracy	Recall	Precision	f1_score	FPR
2	stressCPU	0.71	0.91	0.44	0.60	0.97
	stressBw	0.80	0.97	0.80	0.88	0.73
	stressPacketLoss	0.76	0.93	0.80	0.86	0.83
4	stressCPU	0.79	0.96	0.46	0.62	0.93
	stressBw	0.77	0.94	0.80	0.86	0.81
	stressPacketLoss	0.67	0.98	0.39	0.56	0.92
8	stressCPU	0.79	0.99	0.45	0.62	0.91
	stressBw	0.75	0.92	0.80	0.85	0.89
	stressPacketLoss	0.80	0.98	0.80	0.88	0.71
16	stressCPU	0.81	0.99	0.46	0.63	0.91
	stressBw	0.79	0.97	0.80	0.88	0.73
	stressPacketLoss	0.78	0.95	0.80	0.87	0.76
32	stressCPU	0.78	0.95	0.46	0.62	0.94
	stressBw	0.78	0.95	0.80	0.87	0.77
	stressPacketLoss	0.77	0.94	0.80	0.86	0.81

In the following, we present some preliminary results that are obtained from applying the trained model on test data.

Table 6 shows values of 5 of the Anomaly Detection Metrics that evaluate our system performance for different number of clients on the CPU resource at the Physical Layer. We show that the accuracy, which is a fraction of correctly predicted points, is acceptable and in close range in all tests. The same thing is true for the recall matric which shows the fraction of true anomalies that are correctly classified. This is also applicable to all other metrics.

Figure 54 illustrates the values of the mean square error (MSE) which is the mean square of the difference between the actual exhibited value and the predicted one. Here are the MSE values obtained from the CPU resource at the physical layer as a function of the number of clients participating in the training. The plots represent the intensity of anomalies injected for different testsets, here for example in CPU stress test, the CPU stress values are 10, 30, 50,70 per cent of its actual load. From Figure 54, we can observe that the MSE values change with the intensity of the injected anomalies. However, there is no direct correlation between them. This can be explained by the fact that the number of features used for training is quite high, it contains features that are not really related to the stress as we haven't really done any feature selection on the data. Also, the number of clients participating in the training affects the values of the MSE: we could notice that in all types of stress testing the MSE values changed in correspondence to the number of FL participants.

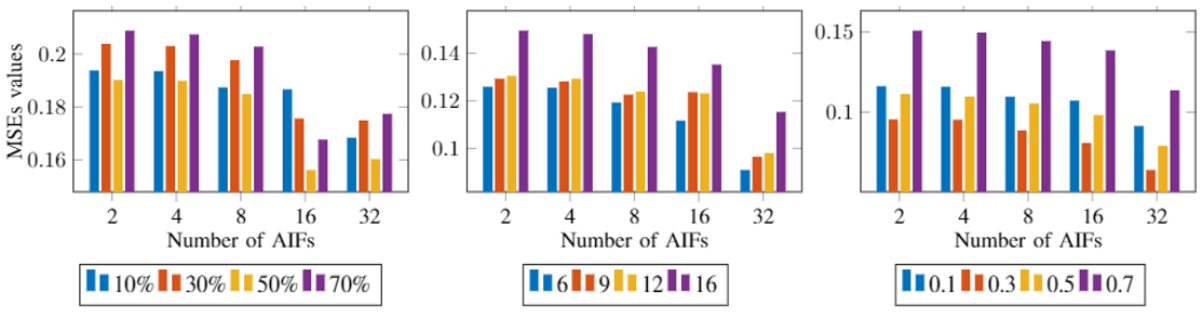


Figure 54 i) CPU Stress test, ii) Bandwidth Stress Test iii) Packet Loss Test of CPU resource at container layer

6.2 Datasets to enable machine learning over diverse RAN and application scenarios

We present here our work on creating curated datasets from multiple scenarios of interest to next generation mobile networks. These datasets have been generated with the objective of creation of benchmark datasets for evaluating and comparing data-driven algorithms as well as accelerating the development of machine learning algorithms from the vantage point of the mobile edge. While the dataset itself enables the development of machine learning algorithms for this domain, the diversity of the scenarios in which different datasets have been created also promote the understanding of the generalisability of the approaches developed through the transferability of knowledge between diverse network and application scenarios.

6.2.1 Introduction

The development of machine learning models relies on having representative data. Large scale data gathering in a radio network comes at an observation cost (storage, I/O bandwidth and, by extension, network QoS). It also requires the observation of the network over various network states (empty/congested, varying number of users) in various application use-case scenarios (flash crowds, high speed mobility). Network infrastructure is also highly heterogeneous with different base stations having different proprietary schedulers, different abilities to do MIMO/beamforming, different frequency ranges of low band and high band, and similar concerns. What does representative mean in the presence of this heterogeneity of radio infrastructure, network state and application use cases? How does one go about creating a representative dataset? Creating a single dataset that captures all the heterogeneity is infeasible because of the non-exhaustive nature of the variations in the environment. So, we must create datasets with clear indication of what scenarios are represented in them. This is essential to inform the community about the scope of the dataset, which directly impacts the scope of the models developed. While other methods of machine learning such as image recognition and natural language processing have for a while had standard datasets and benchmarks for quantitative comparison, this is not true for mobile communication network methods. With the existence of well-defined standard datasets, one can generate pre-trained models that can form the basis for the development of more specific domain-adapted models. From research as well as a development standpoint, this is a better alternative than having each dataset and model in the domain be created and tuned in silos, resulting in disjoint spurious work that does not benefit from incremental improvement.

Most mobile radio network prediction and management tasks derive learning from similar base phenomena, like multipath fading, multi-user congestion, and handover. Leveraging fundamental outputs from a pre-trained model results in a base model that can be used as the starting point to fine-tune or domain-adapt for many heterogeneous target domains or tasks, resulting in turn in more efficient modularized learning structures. For example, instead of separate prediction models for network throughput and end-to-end delay, one would start from a pre-trained structure that is then adapted to

predict delay or throughput. Another example is instead of training from scratch separate models for small high-band (mmWave) cells, and large mid-band cells, one would start from a pre-trained structure for 5G networks and domain-adapt with few samples of data from each specific deployment infrastructure. For pre-training to be useful to build on, it must be created over open datasets. For open datasets, one needs to properly communicate the applicability of the data to various learning tasks by stating how representative it is for those tasks. This requires statistics and formal scrutiny. We address this task of creating such a representative dataset, or rather a family of datasets.

The essential components are the following. Below is a list of things that we consider essential for the development and adoption of machine learning methods for the domain of mobile radio networks.

- Standardized open datasets for various application scenarios and network infrastructures
- A standardized way to describe the variance captured in the datasets and the application scenarios they are applicable to
- Formalised procedures for replicability and details on how the model generalises over various sub-domains
- Pre-trained models that capture the base properties of the domain that can be used as a starting point for building domain- and task-adaptation over.

We have a few key research questions that we would like to study and understand through the evaluation of these datasets. Firstly, we would like to understand how diverse in terms of the data generated are the scenarios that mobile networks need to support. Secondly, we want a fundamental understanding of the relationships to be modelled in the network to solve the tasks that we want to, using machine learning. And finally, we would like to evaluate the generalisability or transferability of the machine learning models developed for predictions pertaining each application and each network deployment scenario to other applications and network deployment scenarios, so as to understand if each model needs to be trained stand-alone with datasets specific to the task or the understanding in the models can be reused and built upon.

6.2.2 Implementation

The dataset requirements depend on the problem to solve. Datasets that include application metrics can be used to optimize the application, whereas metrics from the lower layers in the network can be used to both optimize the network and as input to the application. Datasets of network traffic traces are used in the forecasting of traffic to plan for changing resource demands, load balancing and more. Datasets of application metrics can be used to predict application performance. Base station radio metrics can be used to predict user resource requirements, and user QoS. So, with application metrics, one can make the application better, and with radio metrics one can make the network and/or the application better. Our focus is on radio metrics datasets from a base station perspective. The automation and optimization of the base station is where the complexity and main challenges, we believe, are concentrated.

6.2.2.1 Challenges

Generating representative dataset for this domain have some constraints, including the propriety nature of data from commercial radio base station equipment. There are limitations of the accuracy of data from simulator networks. Simulators are useful for testing specific aspects of a network such as a handover policy or an admission policy at large scale. They are also well suited for generating large amounts of data in diverse scenarios. However, they do not fully capture the complexity of the network infrastructure. Non-commercial testbeds, such as the srsRAN 5G SDR testbed, take us closer to reality by capturing real wireless signal propagation properties, and are well suited for prototyping applications. However, they are limited by the number of devices one can connect to them in a testbed environment, and so limiting the multi-user contention environment one can create. The same limitation applies to

commercial testbeds as well that might be at infrastructure manufacturing and telecom companies. One cannot create large multi-user contention scenarios, the handling of which is a major challenge. The advantages of simulators and testbed networks include that they are sandbox environments that allow for the creation of specific or rare scenarios to log and study, for example data from minority classes or rare classes for anomaly detection.

Finally, data from a live network, realistic both in terms of signal and multi-user behaviour, has the restriction of constrained observation granularity in current telecom infrastructure, wherein high logging cost affects network efficiency and user performance. It also restricts the dataset to only contain observations from current and not future scenarios.

Considering the constraints of these approaches for generating high-utility datasets, one needs to understand what can and cannot be learnt from each data source and device a way to learn models by combining multiple data sources to overcome the limitations of each. Such a curated dataset can have great impact in providing open reusable datasets and encourage the development of benchmarks and pre-training on them.

The aforementioned challenges of generating data from non-commercial, and commercial testbeds as well as live networks is the reason, we resort to generating them from well-established network simulators. We choose the ns3 simulator ecosystem (Figure 55) wherein we begin with the base simulator and extend it with third-party components to create richness of scenarios through various traffic generating components, and the 5G radio component, with the ability in the future to extend it to include closed-loop implementations of AI algorithms using ns3-ai and ns3-gym.

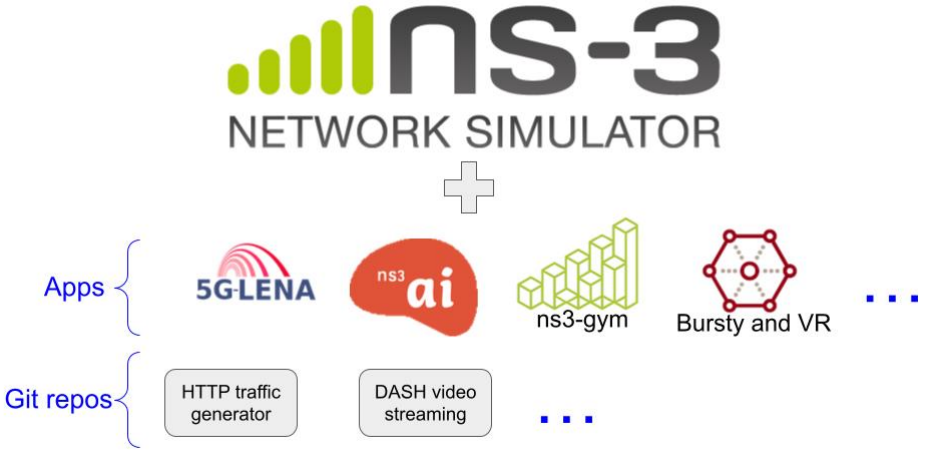


Figure 55 The ns3 ecosystem with several third-party plugin components available for the extension of the main simulator

6.2.2.2 Scenarios

To scenarios from which to generate datasets was decided upon based on the 3GPP document, TR 138 913 - V1420 - 5G; Study on Scenarios and Requirements for Next Generation Access Technologies [3gpp17]. This document presents the scenarios that have been considered as challenging and interesting for the next generation access technologies. The document lists 12 scenarios, Indoor hotspot, Dense urban, Rural, Urban macro, High speed, Extreme long-distance coverage in low density areas, Urban coverage for massive connection, Highway Scenario, Urban Grid for Connected Car, Commercial Air to Ground scenario, Light aircraft scenario, and Satellite extension to Terrestrial. While it is of value to create datasets for all these scenarios. We select the Dense urban scenario since it seems to us to have the most impact. Each scenario here is a different type of infrastructure deployment,

however the different scenarios that we have been talking about previously pertain to variations that can be withing the same infrastructure deployment, with varying densities of users, presence of micro cells, varying densities of traffic etc. Table 7 describes the attributes defined by 3GPP for the dense urban deployment.

Dense urban scenario description

Table 7 Attributes to set for creating a dense urban scenario; Source: TR 138 913 - V1420 - 5G; Study on Scenarios and Requirements for Next Generation Access Technologies (3GPP TR 38913 version 1)

Table 6.1.2-1: Attributes for dense urban

Attributes	Values or assumptions
Carrier Frequency NOTE1	Around 4GHz + Around 30GHz (two layers)
Aggregated system bandwidth NOTE2	Around 30GHz: Up to1GHz (DL+UL) Around 4GHz: Up to 200MHz (DL+UL)
Layout	Two layers: - Macro layer: Hex. Grid - Micro layer: Random drop Step 1 NOTE3: Around 4GHz in Macro layer Step 2 NOTE3: Both Around 4GHz & Around 30GHz may be available in Macro & Micro layers (including 1 macro layer, macro cell only)
ISD	Macro layer: 200m Micro layer: 3micro TRxPs per macro TRxP NOTE4, All micro TRxPs are all outdoor
BS antenna elements NOTE5	Around 30GHz: Up to 256 Tx and Rx antenna elements Around 4GHz: Up to 256 Tx and Rx antenna elements
UE antenna elements NOTE5	Around 30GHz: Up to 32 Tx and Rx antenna elements Around 4GHz: Up to 8 Tx and Rx antenna elements
User distribution and UE speed	Step1 NOTE3: Uniform/macro TRxP, 10 users per TRxP NOTE6, NOTE7 Step2 NOTE3: Uniform/macro TRxP + Clustered/micro TRxP, 10 users per TRxP NoTE6, 80% indoor (3km/h), 20% outdoor (30km/h)
Service profile	NOTE: Whether to use full buffer traffic or non-full-buffer traffic depends on the evaluation methodology adopted for each KPI. For certain KPIs, full buffer traffic is desirable to enable comparison with IMT-Advanced values.

Our ns3 scenario follows these attributes described (for some attributes in a scaled down manner) to create a dense urban deployment. Figure 56 and Table 8 describe the parameters used in the simulation of topology deployment, UE mobility, RAN and lists all the traffic applications being used nu the UEs in the network.

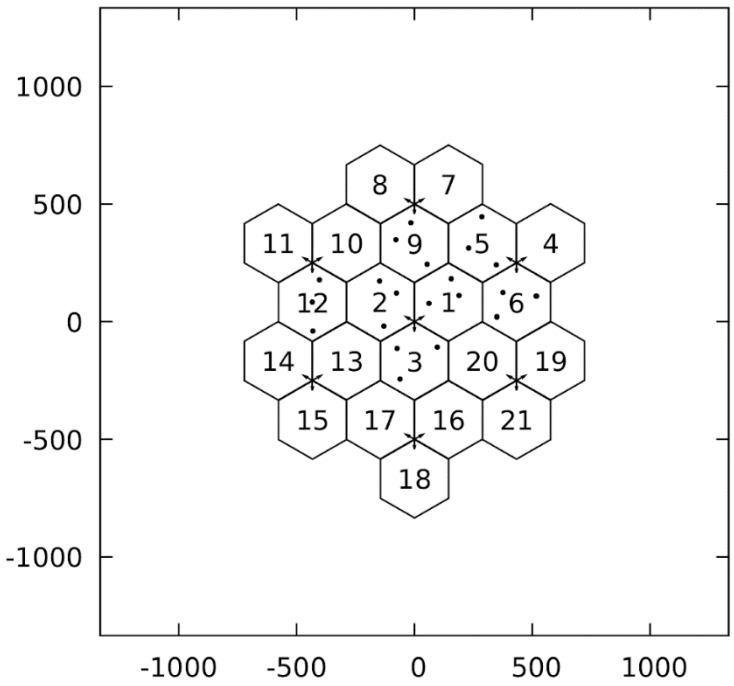


Figure 56 An illustration of the macro and micro layer topology with a hexagonally tessellated macro cells (3 base stations at each cell site with 120-degree beam angles) and random dropped micro cells (dots inside the macro cell areas).

Table 8 Simulation scenario description

Deployment Topology	**UE mobility**
Hexagon tessellation macro cells • Macro cells: up to 21 • Beam width macro cell: 120° Random drop micro cells • Micro cells: 3/macro cell • Beam width micro cell: 360° UEs • Up to 30 per macro cell	Random waypoint mobility • 80% UEs are at slow walking speeds • 20% UEs are at driving speeds
RAN	**Application traffic**

4G
• FDD
• Round robin / proportional fair schedulers
5G
• TDD
• Round robin / proportional fair schedulers
	Traffic sources
	• Web browser activity
	• Adaptive video streaming
	• VR traffic
	  ○ Minecraft
	  ○ Google Earth
	  ○ Virus Popper

In a simulation environment one can create datasets with data sources from the UE RAN side, the base station RAN side as well as from the application server and client side. The ability to observe from all these vantage points to create a combined cross-layer monitored dataset is very valuable for developing

solutions at the edge wherein RAN metrics can be used to predict application specific QoS. Below we list all the observability points in the network that generate logs and provide information on logging resolution and time. Simulation time is only limited by the speed of the simulation and the number of varied scenarios we would like to generate.

Observability

- Periodic uplink and downlink delay measurement
- Throughput measurement through a full buffer traffic generator
- RAN measurements
    - At UE and BS
        - PHY: Signal quality
        - MAC: Scheduler decisions
        - RLC / PDCP byte traces
        - Handover
    - Additional BS observability created as needed
        - Buffer status report (BSR)
        - RLC buffer status
- Application measurements
    - Packet traces for each application traffic
    - Application specific QoS
- Mobility
    - Periodic location information of UEs
- Logging resolution
    - Per scheduling time instant or TTI (1 ms for LTE and sub-ms for NR)
    - Per packet trace
- Simulated time
    - 10 random instantiations of each scenario of 1000 seconds; 10,000 seconds of data per scenario

6.2.3 Dataset characterization results

We provide a visualization of the observable metrics in the datasets through various plots that showcase the characteristics of the data and the scenario. The presented plots are a subset of the observable metrics, chosen for their significance in providing insightful observations of the datasets. For conciseness, the visualization is limited to the urban dense macro+micro scenario, which encompasses both the macro layer and micro layer properties.

The data in the dataset is categorized, separating those who move within the larger, sparse macro coverage area (referred to as macro UEs) and those in the smaller, dense micro coverage area (referred to as micro UEs). This classification emphasizes the diversity of the scenario and illustrates the differences in the observed metric characteristics based on the deployment context.

Through this analysis, the aim is to highlight the presence of scenario diversity and the difficulties in transferring machine learning models trained in one context to another with different data distributions and relationships.

6.2.3.1 Time Series

Figure 57 shows the time series variations of Signal to Interference and Noise Ratio (SINR), Modulation and Coding Scheme (MCS), and one-way-delay metrics and provides an overview of the RAN state for a single run. Each plot consists of both UL and DL metrics (in red and blue respectively) overlaid with a dashed vertical line showing handover events along the time axis.

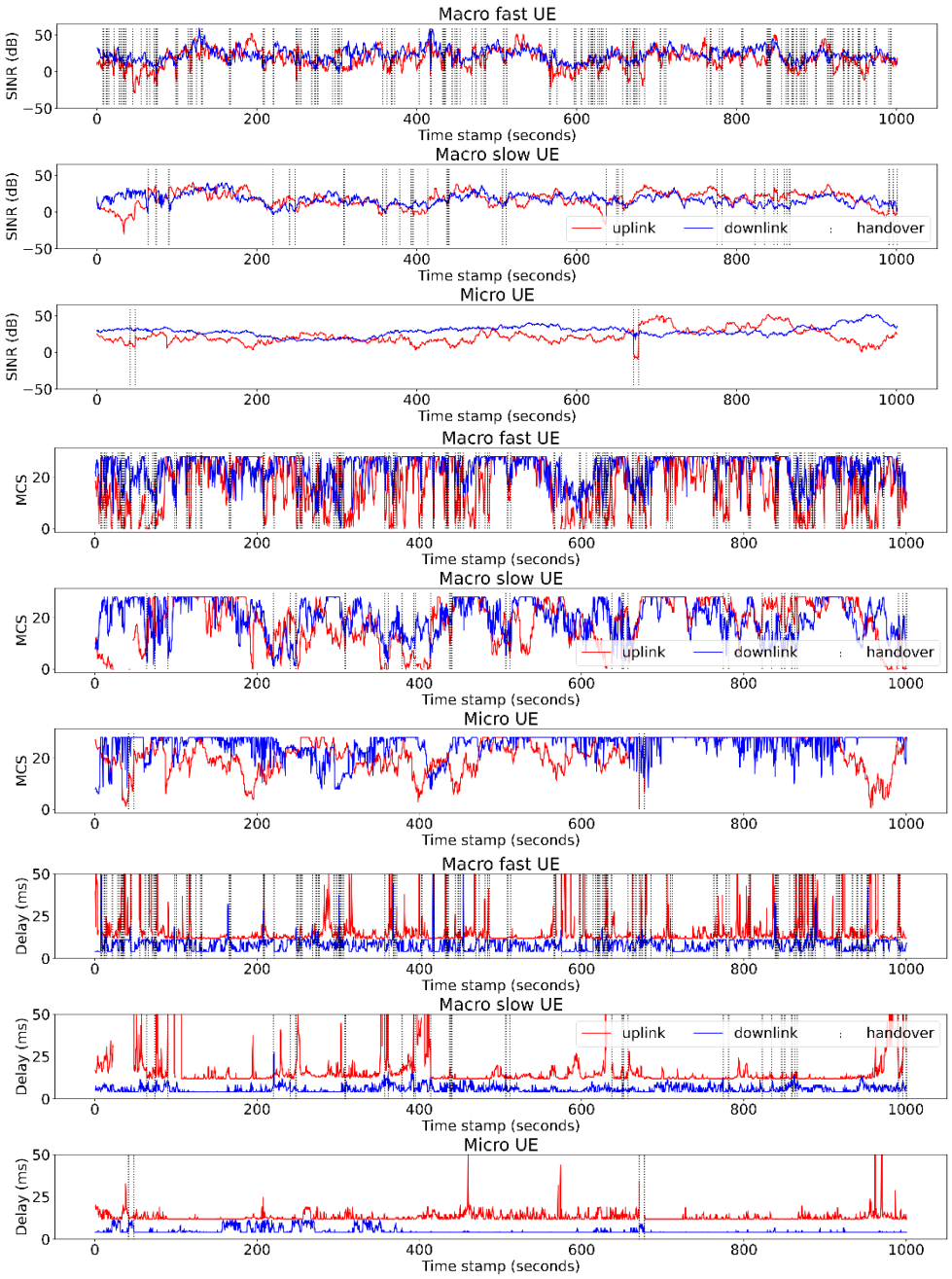


Figure 57 Time series of RAN metrics for sample UEs from different mobility groups of macro and micro layers in an arbitrarily selected run. Each plot consists of an overlay of the uplink metric, plot in red, and the downlink metric, plot in blue. Dashed vertical

For each metric we show the time series for three different UEs in the network to demonstrate the differences between a fast UE, and slow UE in the macro layer and a UE from the micro layer. The macro layer UEs show more variation with time than the micro UEs. The fast macro-UE has significantly more handover events with frequent changes in associated base station due to its experience of fast changing signal quality and mobility across cell edges. The delay plot shows that as expected in a mobile network the DL delay is lower than the UL delay. Sudden spikes in UL delay

especially prominent in macro UEs could be a result of shadowing or high interference from the micro UEs.

Note that the y-axes of the delay plots have been clipped to better show the variations, with the full distributions being shown in the histogram plots in Figure 59.

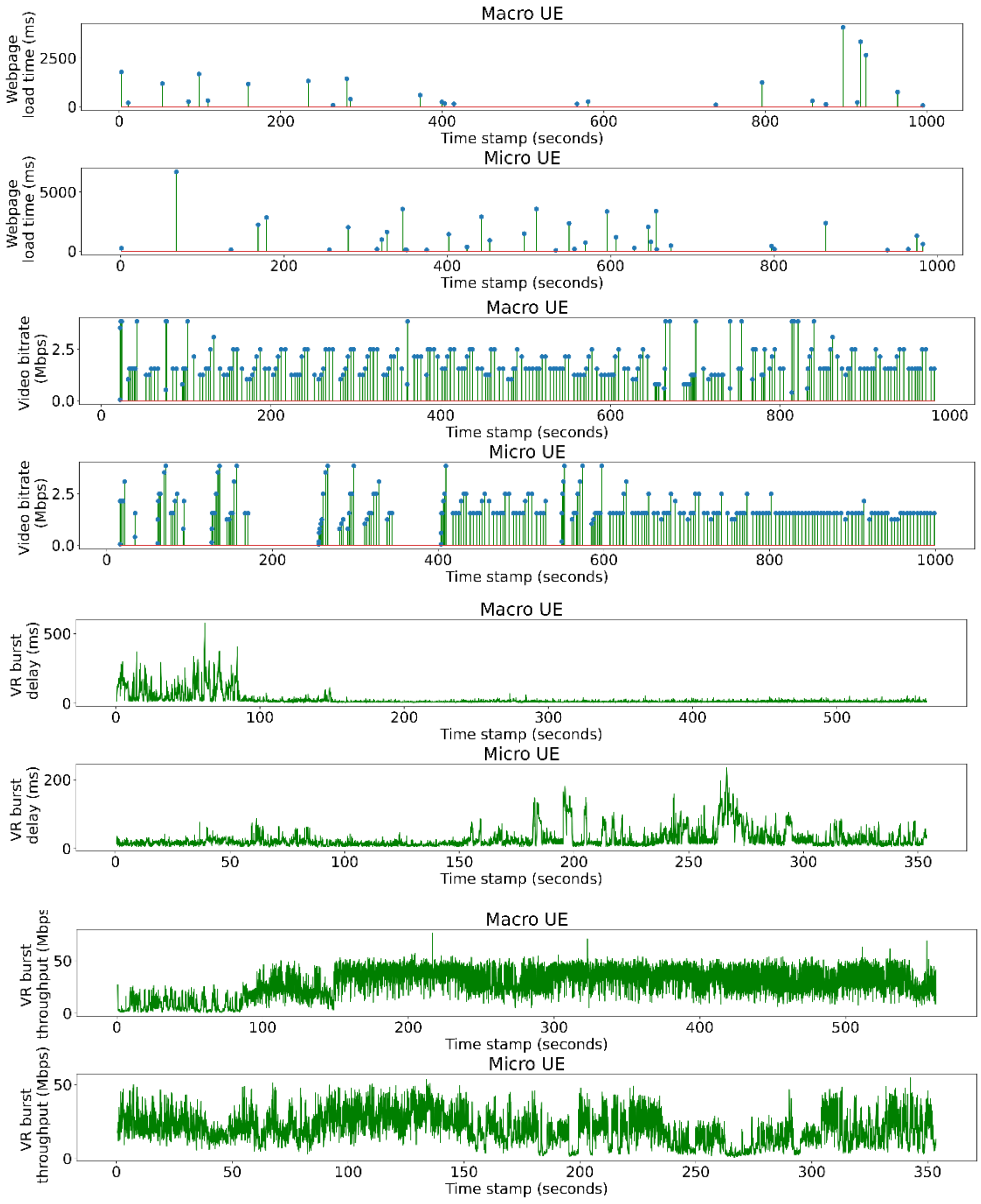


Figure 58 Time series of application metrics for sample UEs from the macro and micro layer groups in an arbitrarily selected run

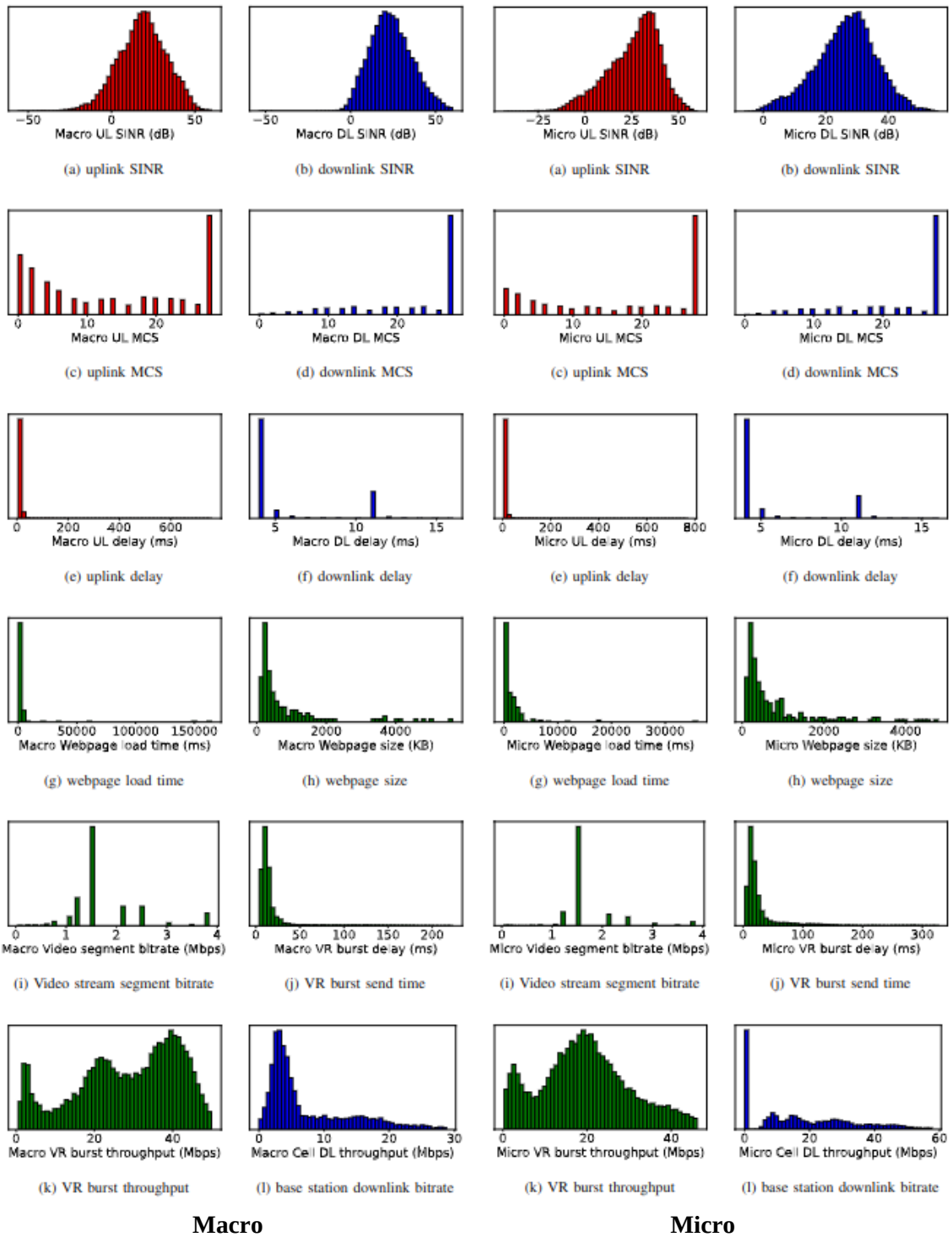


Figure 59 Histograms of a subset of metrics from the macro (left) and micro (right) layer of the urban dense scenario dataset

Figure 58 presents a time series view of the application metrics from the three installed applications: web browsing, adaptive video streaming and VR. The web page load time metric is defined as the time required to request and fully load a web page of varying size, sampled from an estimated distribution

of current web page sizes. The Video Bitrate metric reflects the bitrate at which each video segment was requested, based on the current buffer occupancy, and estimated available bandwidth.

The VR burst delay metric depicts the time taken for a VR burst of packets to be received at the UE, where each burst consists of a varying number of packets dependent on the context of the VR application. The VR burst throughput metric shows the throughput achieved over each of these bursts.

Thess plots provide visual insight into the rate of change of the observed metrics over the dynamic network.

6.2.3.2 Histogram

The histogram plots offer a glimpse into the distribution of the metrics and reveal the disparities between the macro and micro layers of the scenario. Figure 59 displays histograms of the RAN and application metrics from the macro and micro layers.

The higher probability mass on low SINR values in the macro layer is translated to a higher mass on the lower MCS values. Additionally, the macro layer exhibits longer tails in web page load time and a higher probability mass on lower values for video bitrate compared to the micro layer. The plots also suggest that the micro cell is empty for longer periods of time, likely due to its higher bandwidth, as indicated by the spike in 0 values for DL cell throughput when observed in time windows.

6.2.3.3 Correlation Analysis

A simple assessment of the relationship between metrics is achieved through a Spearman correlation coefficient heatmap between them. Since application metrics are reported at varying time granularity we plot this for RAN metrics to understand the RAN state for the macro and micro layers. We see that between the heatmaps in Figure 60, the correlation of UL SINR with UL MCS, UL delay and DL SINR are significantly different between the macro and micro layer metrics indicating differences in the dynamics of the network state within them.

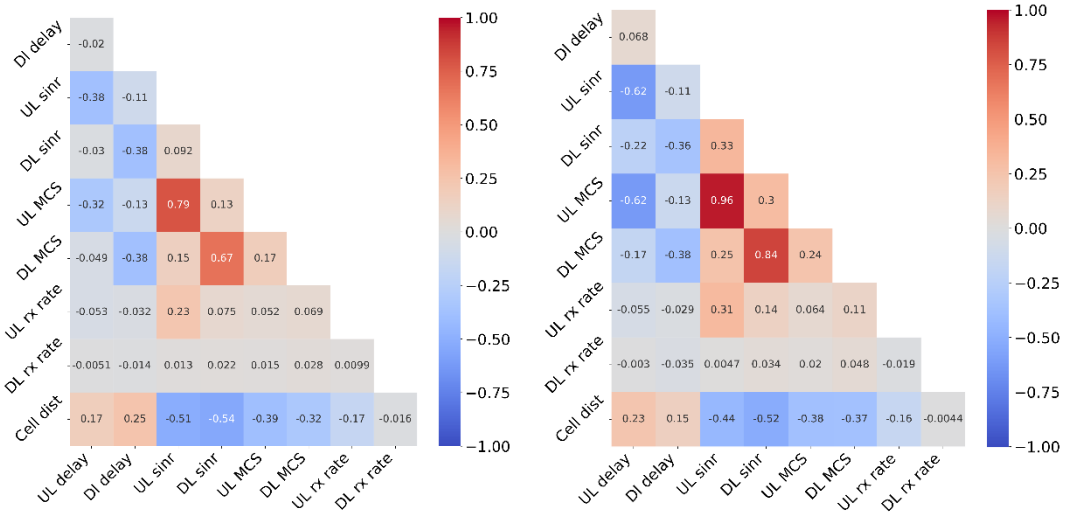


Figure 60 Heatmap of the Spearman correlation coefficient of RAN metrics for the macro and macro layers of the urban dense scenario dataset

6.2.4 Summary

The differences in the dataset characterizing figures here highlight how variations in deployment topology, user density, traffic applications, etc. result in data with different dynamics and distributions.

Through these insights, we pinpoint the need to explore and evaluate the generalizability or adaptability of machine learning models over diverse scenarios. Such an evaluation helps answer questions on how much of a trained model can be directly transferred, as well as how much we need to adapt models for scenarios expected to co-exist in future deployments and use-cases for 5G and upcoming 6G networks.

The datasets created and potentially generated from the simulation scripts can be used to benchmark the performance of ML models. This also enables the community to approach the evaluation of ML models, such as for performance prediction, taking the inherent heterogeneity of networks and scenarios into consideration.

7. Platform Implementation

In this section, we provide an overview of our implementation of two essential components in the NSAP architecture: the IOC and the Data Collection Pipeline (defined in Section 3, here after referred to as Data Pipeline or DCP). Our primary focus has been on creating a functional NSAP, which together with the MTO, is able to make placement decisions based in metrics retrieved and processed by the DCP and used by the IOC. With this implementation approach, we extend the compatibility of the NSAP with the CCP, validating the final implementation with multiple MEC systems.

7.1 Intelligent Orchestration Component

This section provides an exploration of the IOC implementation phases and architecture, providing a view on its core functionalities. Furthermore, a validation scenario will be presented, offering a practical demonstration of its role in enhancing decision-making and resource allocation in the AI@EDGE architecture.

7.1.1 IOC architecture and implementation

The IOC has undergone different prototyping phases, starting as a standalone entity which could retrieve metrics and make placement decisions based on the parameters sent by the MTO and the metrics retrieved from Thanos. This first prototype was developed to test the placement functionalities of the IOC while the rest of the components where not ready. Subsequently, when the Data Collection Pipeline (DCP) and metrics aggregator where present in the platform, the IOC moved into a non-standalone approach, relying on the DCP metrics to take decisions. These are the two versions developed:

1. IOC SA: includes metric retrieval and processing of individual nodes on each MEC system.
2. IOC NSA: has placement functionalities, including Node selection and MEC system selection and relies for the data retrieval job on the DCP.

The IOC has been built in a microservice approach, and is also fully configurable, allowing it to move from different versions on start.

7.1.1.1 IOC SA

This first prototype has been divided in different microservices developed in Go Lang, which are then built on Docker images and finally deployed on 3 different Kubernetes pods. The services, illustrated in Figure 61, are as follows:

- **Thanos service:** in charge of retrieving the necessary metrics from Thanos, to process them and send them to the Database service. This metrics include CPU, RAM and Disk availability.
- **Database service:** acts as interface between the Thanos Service and the Cache database, implementing different methods to store all data.
- **Cache DB:** A Redis instance that allows the IOC to have instant access to the necessary metrics for placement.
- **Placement service:** This service oversees selecting the best node to place an AIF. It uses the metrics received and the requirements from the AIF Descriptor in order to calculate a priority for each node. Once calculated, the resulting list is sent to the REST API Server.
- **REST API Server:** This server is continuously running waiting for placement request from the associated MTO. When a placement request is received, the information is sent to the Placement service, and the response from this microservice is then sent back to the MTO.

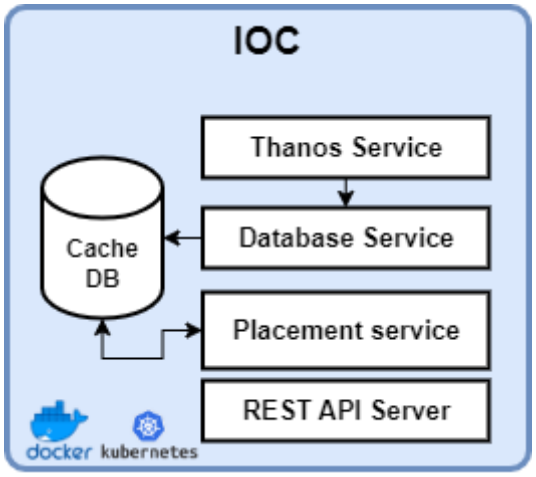


Figure 61 IOC standalone architecture

7.1.1.2 IOC NSA

This non-standalone version adds compatibility with two entities: the metric aggregator and the DCP.

The metric aggregator is in charge of processing and aggregating data from different nodes to group them per MEC system, leaving a global view of each MEC system to the IOC, and allowing the Intelligent Acceleration Resources Manager (IARM) to choose the node to deploy the AIF. Furthermore, HW acceleration metrics are made available, allowing the IOC to make better decisions for the AIFs which require it. The available metrics are listed in Figure 62.

This approach allows to reduce the volume of data sent from the different CCPs to the NSAP. By consolidating the metrics at the MEC system level, we streamline the data flow, resulting in more efficient communication and a reduced burden on network resources. Additionally, this approach allows a higher granularity at the CCP, enabling the IARM to take almost immediate actions at node level.

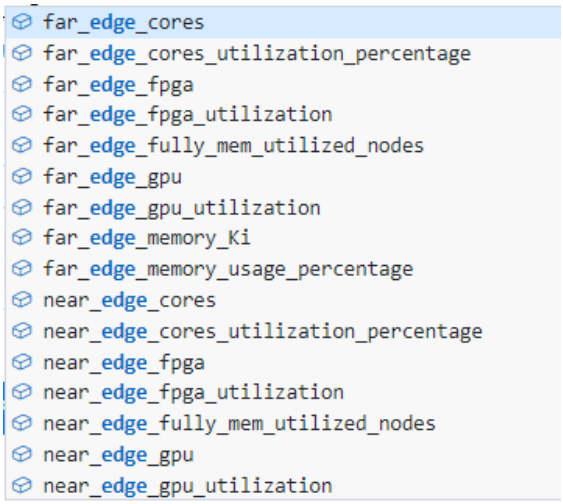


Figure 62 Metrics list from the Metrics Aggregator

Regarding the DCP support, it allows to introduce a refined division of responsibilities between the different components in the NSAP, relying on the DCP for the processing and storing of the necessary metrics. Furthermore, this version introduces a flexible system that supports both MEC system selection and node-level selection, depending on the predefined deployment configuration. This adaptability

allows for dynamic adjustments based on specific use cases, where for example the metric aggregator at MEC system level is not available.

As can be seen in Figure 63, the NSA IOC has a smaller package than the previous version, introducing the DCP service which now assumes the role of retrieving metrics directly from the DCP through the REST API Server.

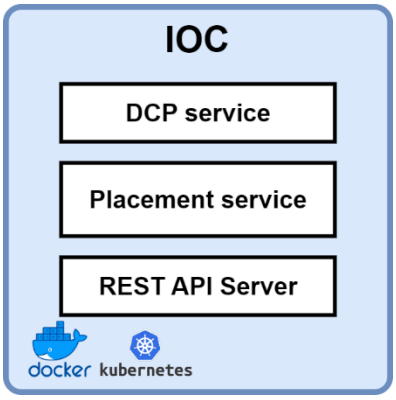


Figure 63 IOC non-standalone architecture

7.1.2 IOC – MTO interface

The IOC has implemented a REST API interface to communicate with the MTO. In these communications, the MTO can request a placement decision to the IOC, and the IOC will send a list of sorted MEC systems to the MTO to trigger the placement on the selected MEC system. An example request is shown in Figure 64.

When the IOC starts running, it starts an HTTP server that listens for MTO requests, with a Placement method that expects a list of AIF Descriptor parameters, that will allow the IOC to make a decision.

```
json data string [{"name": "sentimentanalysis", "helm-chart": "aiatedge/sentiment-analysis-chart", "requiremen
tualMemory": {"virtualMemSize": 512}, "virtualDisk": {"sizeOfStorage": 512}}, "accelerationDescriptor": {"hard
 [{"id": "latency", "minValue": 5, "maxValue": 100}]}]
```

Figure 64 Example of Placement request from MTO

7.1.3 Validation methodology

A validation scenario is proposed to validate the standalone version of the IOC with node system selection, which corresponds to IOC SA. Accordingly, IOC NSA version will be validated later in this document, in section 7.2.2, where the integration with the Data Pipeline is analysed, and in D4.2, where the IOC is integrated with CCP components.

- This scenario focuses on the validation of the IOC placement decisions at node level, which is being analysed, together with the integration with the MTO through the MTO-IOC interface.
    - Testbed: FBK testbed.
    - Metrics (KPIs):
        1. Correct placement of AIFs
        2. Placement decision time

3. Time for recovery after unfulfilled AIF requirement

- Steps:
  1. Deploy a Load generator AIF to increase the resource utilisation of a node.
  2. Deploy an AIF through the MTO interface and check the IOC made a correct placement decision.
  3. Repeat steps 1 and 2 to saturate a node and measure all KPIs.

7.1.4 Node placement validation results

As a first prototyping phase, this feature was developed to achieve a standalone IOC version which could take independent decisions without the existence of the IARM at MEC system level. In this case, as can be seen in Figure 65, a Prometheus instance in each node sends all relevant metrics to a Thanos deployment at NSAP level, which aggregates metrics from all nodes to make them available to the IOC. The IOC then retrieves Thanos data, processes, and store them in a Cache DB, to guarantee the immediate availability and reduce the placement time to a minimum.

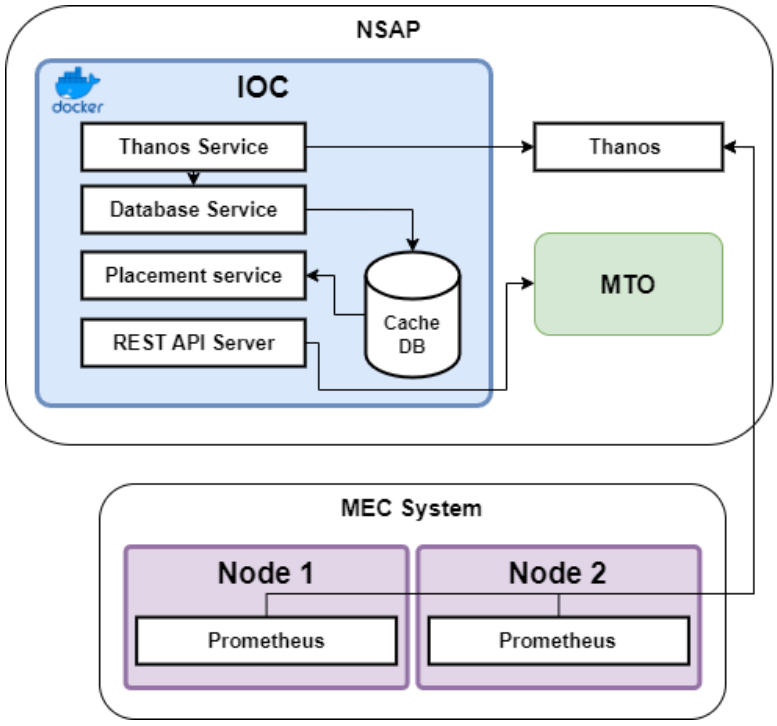


Figure 65 Node placement validation scenario architecture

This test evaluates three aspects:

- Correct placement of AIFs
- Placement decision time
- Time for recovery after unfulfilled AIF requirement

7.1.4.1 KPI: Correct placement of AIFs

First, we evaluate the capability of the IOC to select the node with best resource availability, together with the time to decide. To emulate resource utilisation of the nodes, a Load Generator test AIF has been created, making it possible to configure it to fill up RAM and CPU resources. Due to the nature

of Go Lang and the simplicity of the script that runs to fill up the resources, we do not get a constant RAM and CPU usage, but it is sufficient to evaluate the output and decision-time of the IOC.

When a placement request is received, the IOC gives a score to each node, depending on the requirements of the AIFD received and the availability of the nodes. In this scenario, we have two different nodes: cluster-master and cluster-worker1. Both start with similar resources available, even though cluster-worker1 has slightly more RAM and disk availability, this is why in the first placement, the IOC decides to place the sentiment analysis AIF in this node, as can be seen in Figure 66.

```
aiatedge@nsap:~$ curl -X 'POST'   'http://20.79.248.33:30543/meo/aifd/deploy/'   -H 'accept: application/json'   -H 'Content-T
ype: application/json'   -d '{
  "aifd_name": "AIF_v4_sentiment_analysis_descriptor.yml",
  "namespace": "ioc"
}'
{"aifd_id":"64df3b2e11f8d66df7b67c6a","bodytosend":{"release_name":"sentimentanalysis-64df3b2e11f8d66df7b67c6a","repochart_nam
e":"aiatedge/sentiment-analysis-chart","values":{"nodeSelector":{"kubernetes.io/hostname":"cluster-worker1"}},"command":{"nam
espace":"ioc"},"id":"64df3b2f43aaa9c8e28e2bc9","name":"sentimentanalysis-64df3b2e11f8d66df7b67c6a"}
```

Figure 66 Placement request for sentiment analysis AIF

Following this deployment, a Load Generation Test AIF is deployed to the cluster-worker1, filling up to 4 CPUs and 6GB of RAM memory. As we can see in and Figure 67 a) and b), cluster-worker1 corresponds to the IP 192.168.0.211 (yellow line), and cluster-master to the IP 192.168.0.213 (blue line). Following this deployment, a new placement request is sent to the IOC for a sentiment analysis AIF. At the time of this request, the load generation test AIF is using up 2 CPUs, but no RAM. This reduction of CPU availability is enough to trigger the IOC into placing the new sentiment analysis AIF to the cluster-master, which has more available CPUs and will meet the requirements of the AIFD.

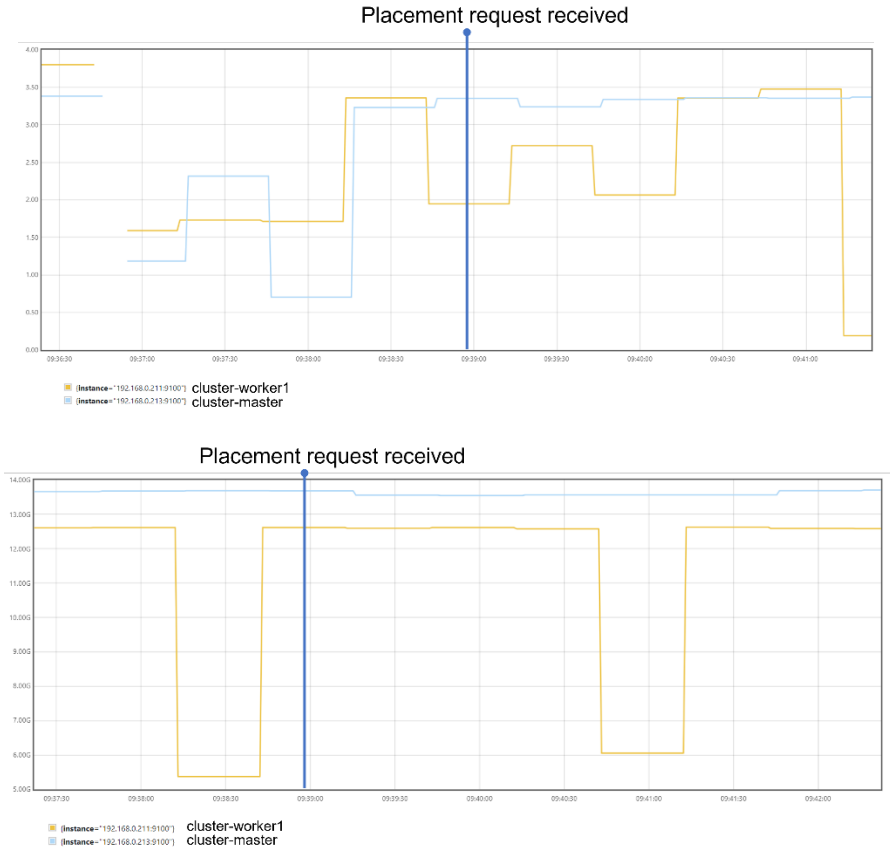


Figure 67. a) RAM availability of both nodes in the MEC system, b) CPU availability of both nodes in the MEC system

We can finally see (Figure 68) the new sentiment analysis AIF deployed in the cluster-master, together with the Load Generator AIF and the previous sentiment analysis AIF deployed at the cluster-worker1 at the beginning of the scenario.

```
ubuntu@DESKTOP-HPE5GNP:~/AI-AT-EDGE-INTELLIGENT-ORCHESTRATOR/load_generator_AIF/helm-chart$ kubectl get pods -n ioc -o wide
NAME                                                      READY   STATUS    RESTARTS      AGE     IP              NODE
S
lg-aif-654964b56c-vzztb                                   1/1     Running   4 (65s ago)   2m43s   192.168.39.25   cluster-worker1
sentimentanalysis-64df3b2e11f8d66df7b67c6a-sentiment-analybhh4v   1/1     Running   0             6m11s   192.168.39.18   cluster-worker1
sentimentanalysis-64df3c3211f8d66df7b67c6d-sentiment-analytcqlh   1/1     Running   0             110s    192.168.200.56  cluster-master
```

Figure 68 AIF deployed at both nodes

7.1.4.2 KPI: Placement decision time

In order to measure the placement time, a series of time measurements were included in the IOC's code. Then, a placement test was conducted10 times, to be able to extract a mean of the time it takes for the IOC to decide. The results can be seen in Table 9. In this case, two measurements have been taken, the metrics retrieval time from the Cache database, and the total placement decision time, resulting in a mean of 4.9 ms for the metrics retrieval and of 7 ms for the total placement decision time.

Table 9 IOC AIF placement time measurements

Attempt #	Metrics retrieval time (ms)	Total placement decision time (ms)
1	14.4	15.2
2	2.9	3.7
3	7	9.1
4	1.6	10.2
5	3.3	3.6
6	0.9	1.2
7	6.6	7.4
8	5	5.8
9	3.4	8.1
10	4.2	5.1
Mean	4.9	7

In general, the metrics retrieval represents the highest amount of time, delaying the decision at certain points up to 14.4 ms. In any case, this mean of 4.9 ms represents a big improvement with respect to the metrics retrieval from Thanos, which was measured to be around 18 ms. This difference shows the importance of pre-processing and storing the data to make it available for the IOC when a placement decision is required, lowering the total placement decision time.

7.1.4.3 KPI: Error-recovery time

Finally, we analyse the impact of making an incorrect placement decision, which could happen on an IOC with random placement, or a very saturated scenario with few resources available. This incorrect placement decision would mean that the assigned node would run out of resources, having to kill the new process in order to guarantee the normal functioning of the rest of the running services. In this test, we deployed two replicas of the Load Generation Test AIF in the cluster-master, with a total CPU utilization of 8 CPUs, and 12 GB of RAM, surpassing the total resources allocated to the node, which are 4 CPUs and 12 GB of RAM in total.

As can be seen in Figure 69, Kubernetes confirms the process has been killed with the status OOMKilled. When this happens, a fallback scenario should start, triggering the IOC to make a new placement decision in a node with higher availability. This scenario has not been implemented; thus, we analyse the impact of deploying an AIF in a node without enough available resources.

```
ubuntu@DESKTOP-HPE5GNP:~/AI-AT-EDGE-INTELLIGENT-ORCHESTRATOR/load_generator_AIF/helm-chart$ kubectl get pods -n ioc -o wide
NAME                                                    READY   STATUS      RESTARTS      AGE     IP               NODE
TES
lg-aif-654964b56c-jj2gg                                 0/1     OOMKilled   1 (42s ago)   57s     192.168.39.36    cluster-worker1
lg-aif-654964b56c-k2zg2                                 1/1     Running     2 (32s ago)   57s     192.168.39.32    cluster-worker1
lg-aif-654964b56c-zndj9                                 1/1     Running     1 (4s ago)    57s     192.168.200.60   cluster-master
sentimentanalysis-64df3b2e11f8d66df7b67c6a-sentiment-analybhh4v   1/1     Running     0             17m     192.168.39.18    cluster-worker1
sentimentanalysis-64df3c3211f8d66df7b67c6d-sentiment-analytcqlh   1/1     Running     0             12m     192.168.200.56   cluster-master
```

Figure 69 Terminal showing AIF process killed

When this AIF is placed, the node normal functioning is interrupted until the new process is killed. This can be seen in the Thanos metrics (Figure 70), which stopped being received in cases where node resources were very scarce, confirming Prometheus process was affected by the AIF placement. In some cases, Thanos stopped receiving CPU metrics from the cluster-master Prometheus instance for 1:06 min, showing the importance of making correct placement decisions in the nodes.

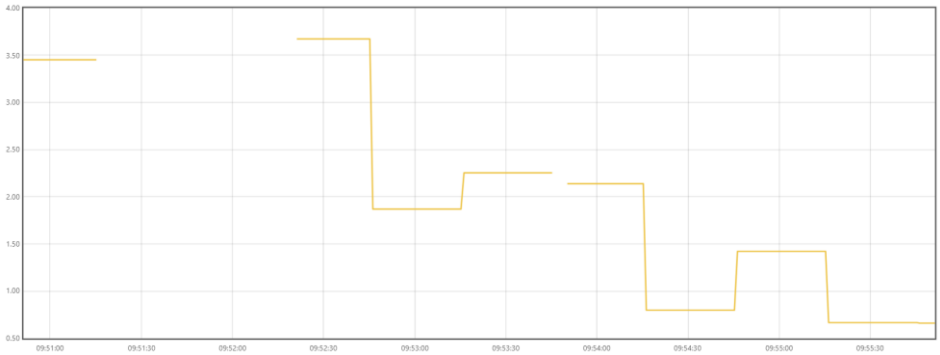


Figure 70 CPU metrics on cluster-master after node saturation

These tests on this scenario have analysed the correct placement decisions of the IOC, the time to make a placement decision, and the impact of making an incorrect placement decision in a saturated node. This big impact of making incorrect decisions, is one of the reasons why the IOC will not take placement decisions at node level, and will leave this final decision to the IARM, which will be placed at MEC system level, being able to retrieve more granular metrics and take faster decisions that will imply a better placement overall.

7.2 Reusable data pipelines

7.2.1 DCP interfaces

The DCP interfaces described above in Section 3.1 are implemented as follows. Figure 71 shows the general architecture of the DCP, with its different main components and examples of connections to data consumers and data producers.

The **Data Server** is the main component in the pipeline. It serves as a first contact point for functions that need to consume data, and for producers to share and store data. For example, the IOC contacts the Data Server to ask for access to measured metrics about resource availability in the CCP. The Data Server then provides the IOC with access to the Data Repository (currently implemented as a Redis Database).

The **Data Repository** is the main database to store and share data. Other smaller local repositories can also be deployed in the CCP.

The **Data Collector** is a component that can be deployed in different places to collect data from producers, such as Prometheus, and send them to the Data Server, which in turns will store them in the main Data Repository. The Data Collector can also store data in a local instance of the Data Repository, for faster access at the Near Edge or Far Edge.

The **Data Collector Registry** is a database where all Data Collectors are registered. When the Data Server is requested to provide data that is not already collected in the main Data Repository, it checks the Data Collector Registry for the appropriate Data Collector to contact and request data.

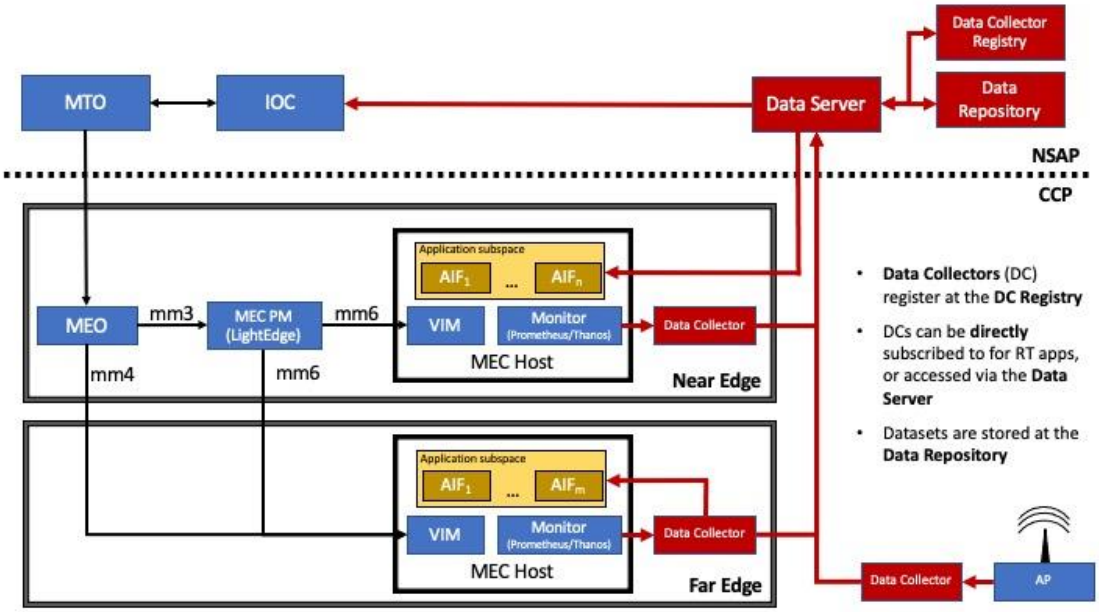


Figure 71 General Architecture of the Data Collection Pipeline

7.2.2 Data pipeline and IOC integration

The DCP has been implemented to allow an easy integration to other components that are producing and consuming data. In particular, the IOC component has been chosen as a use case to demonstrate the data pipeline functionalities and help making implementation choices for these functionalities. In these terms, the integration of the DCP with the IOC has been proven successful. For this integration, the IOC has been deployed in non-standalone mode, and with metrics aggregation support. In this case, all metrics have been served by the DCP directly, which oversees processing metrics from different data sources (Prometheus, metrics aggregator etc.).

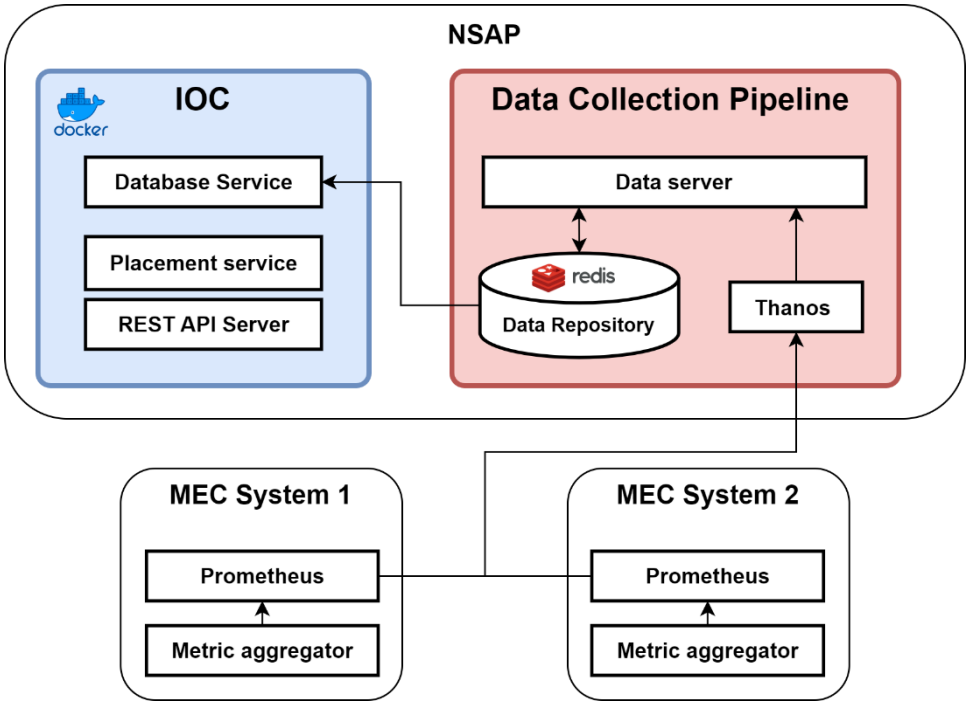


Figure 72 DCP and IOC integration architecture

The IOC adapted the retrieval of the metrics to support the new DCP formatting and storing of data, retrieving the latest metrics available to take a placement decision. In this scenario, we have successfully deployed all necessary entities for AIF placement that were described in the AI@EDGE NSAP architecture, including the MTO the DCP and the IOC. This can be seen in Figure 73, which shows all pods running in the NSAP, highlighting the main ones evaluated in this scenario.

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
dp	dp-66d874b8c4-dhzcf	2/2	Running	0	20h
drift	grafana-56cbc8dbd9-zdlrh	1/1	Running	0	44h
drift	mysql-0	1/1	Running	0	44h
ioc	ioc-6cfbb7cc99-hwhjj	1/1	Running	0	113m
kube-system	coredns-787d4945fb-mqvcr	1/1	Running	0	46h
kube-system	etcd-minikube	1/1	Running	0	46h
kube-system	kube-apiserver-minikube	1/1	Running	0	46h
kube-system	kube-controller-manager-minikube	1/1	Running	0	46h
kube-system	kube-proxy-42jww	1/1	Running	0	46h
kube-system	kube-scheduler-minikube	1/1	Running	0	46h
kube-system	metrics-server-5f8fcc9bb7-rrsdm	1/1	Running	0	46h
kube-system	storage-provisioner	1/1	Running	0	46h
mlflow	mlflow-7c6cd799f8-5kf84	1/1	Running	0	46h
mlflow	mlflow-minio-5b857c4988-z6pl5	1/1	Running	0	46h
mlflow	mlflow-postgresql-0	1/1	Running	0	46h
mto	mto-deployment-74bb64569b-9pggq	2/2	Running	0	19h

Figure 73 Containers running in NSAP

To test the integration of the DCP with the IOC, several AIF deployments have been made, and the metrics retrieval time has been measured in order to guarantee a high performance and low-latency communication between the IOC and the DCP. As can be seen in Figure 74, an AIF has been deployed through an instantiation request to the MTO, using the beforementioned integrated versions of the IOC and DCP.

```
aiatedge@nsap:~$ curl -X 'POST'   'http://20.79.248.33:30543/meo/aifd/deploy/'   -H 'accept: application/json'   -H 'Content-Typ
  "aifd_name": "AIF_v4_sentiment_analysis_descriptor.yml",
  "namespace": "ioc"
}'
{"aifd_id":"6502bfb54022fe9bf0bb691a","bodytosend":{"release_name":"sentimentanalysis-6502bfb54022fe9bf0bb691a","repochart_name"
fe1","name":"sentimentanalysis-6502bfb54022fe9bf0bb691a"}
```

Figure 74 Instantiation of an AIF through the MTO

When a placement request is made, the IOC extracts the necessary information from the AIFD, and requests from the Data Repository of the DCP all necessary metrics to make a decision. In this case, the IOC is working on metrics selection node, thus it will only take into account aggregated metrics which are being exposed by the DCP. As can be seen in Figure 75, the IOC is only retrieving metrics from MEC systems on IPs 10.9.251.10 and 10.142.39.88, ignoring data at node level.

```
Data Pipeline is present, working in SA mode. IARM is  present.

Descriptor mapping time: %s
 29.101µs
Starting metrics retrieval
instances [192.168.0.211:9100 192.168.0.213:9100 192.168.1.251:9100 10.142.39.88:9878 10.9.251.10:9879]
GetXrev in instance: 10.142.39.88:9878
XRevRange result [{1694678965076-0 map[availableCpu:0 availableDisk:0 availableRam:0 far_edge_cores:12 far_edge_cores_utilizatio
32686644 far_edge_memory_usage_percentage:0.006265556047907518 near_edge_fpga:0 near_edge_fpga_utilization:0 near_edge_fully_me
alCpu:0 totalDisk:0 totalRam:0]}]
Inside for {1694678965076-0 map[availableCpu:0 availableDisk:0 availableRam:0 far_edge_cores:12 far_edge_cores_utilization_perce
4 far_edge_memory_usage_percentage:0.006265556047907518 near_edge_fpga:0 near_edge_fpga_utilization:0 near_edge_fully_mem_utili
 totalDisk:0 totalRam:0]}
Available CPU: 0
Available Disk: 0
Far Edge Cores: 12
Far Edge Cores Utilization Percentage: 0.3
GetXrev in instance: 10.9.251.10:9879
XRevRange result [{1694678965075-0 map[availableCpu:0 availableDisk:0 availableRam:0 far_edge_cores:12 far_edge_cores_utilizatio
32686644 far_edge_memory_usage_percentage:0.006265556047907518 near_edge_fpga:0 near_edge_fpga_utilization:0 near_edge_fully_me
alCpu:0 totalDisk:0 totalRam:0]}]
Inside for {1694678965075-0 map[availableCpu:0 availableDisk:0 availableRam:0 far_edge_cores:12 far_edge_cores_utilization_perce
4 far_edge_memory_usage_percentage:0.006265556047907518 near_edge_fpga:0 near_edge_fpga_utilization:0 near_edge_fully_mem_utili
 totalDisk:0 totalRam:0]}
Available CPU: 0
Available Disk: 0
Far Edge Cores: 12
Far Edge Cores Utilization Percentage: 0.7
```

Figure 75 IOC logs on AIF placement request

Finally, to make an average of the metrics retrieval time from the DCP, a set of AIFs have been instantiated, and the metrics retrieval time can be seen in Table 10, having an average of 4.4 ms. In this case, we cannot see a loss in performance when comparing the IOC standalone version with the non-standalone version. Having the DCP in a separate pod and changing the type of metrics retrieved has not affected negatively in the performance, allowing the IOC to make fast placement decisions.

Table 10 Metrics retrieval time from DCP

Attempt #	Metrics retrieval (ms)
1	1,5
2	7,1
3	1,2
4	1,7
5	1,5
6	3,3
7	9,3
8	1,4
9	1,7
10	15,6
Mean	4.4

In conclusion, the IOC and DCP validation, together with the metrics aggregations at MEC system has been proven successful, showing a fast response time at the same time the DCP has brought some changes in how data is accessed and stored, allowing for future versions of the IOC to retrieve historical data and the possibility of making forecasts of the reported metrics for a higher resilience in IOC decisions.

7.3 Conclusion

In this section, we have presented two implementations of NSAP components from AI@EDGE architecture which have been implemented, the IOC and the DCP. The IOC provides the platform the possibility to optimise the placement of the AIFs in different MEC systems and has been successfully tested and validated both in its standalone version, as well as in its non-standalone version, working together with the DCP. On the other hand, the DCP provides a data collection and processing functionality, that in this specific use case, has enabled the IOC to retrieve essential metrics for its decision making. This implementation has also been validated and tested, working seamlessly with the IOC in the NSAP.

8. Data sources

In Section 5 of D3.1 [D3.1], we reported on raw data made available to AI/ML algorithms by hardware and software components utilized in the project's implementations. They corresponded to: i) container-level data, collectable via Kubernetes in the SYRROCA framework (Section 4.2.1 of D3.1 [D3.1]); ii) physical-server-level data, collectable via Prometheus; iii) base-station- and UE-level data, collectable from SRS virtualized components; iv) WiFi-AP-level data, collectable from 5G-EmPower virtualized WiFi access points; v) application-server-level data, collectable from a video streaming application that uses the DASH protocol; vi) 5GC-level data, collectable from ATH 5GC. In the next subsection, we update the latter.

8.1 Network performance real-time monitoring

In this section, we provide an update of the list of KPI and alert data that can be collected in Prometheus format (https://prometheus.io) from the Athonet 5G core network, deployed as part of the 5G network of AI@EDGE's integration testbed and of UC1, UC2, and UC4's testbeds (cf. project Deliverable 4.2 [D4.2] and Deliverable 5.2 [D5.2]). This data is exploitable by AIFs to monitor the network status and consequently take decisions on resource allocation, security, etc. An initial version of Table 11 was provided in Section 5.6 of project Deliverable 3.1 [D3.1]. The version provided here contains some additional parameters, made available with more recent versions of Athonet's software.

Table 11 Athonet core network KPI data

Description of the collectable parameter	Type	3GPP specification reference
Number and type of NAS 5GS messages received over N1 interface	KPI	TS 24.501
Number and type of NAS 5GS messages sent over N1 interface	KPI	TS 24.501
Number of NAS 5GS message decoding failures by the AMF	KPI	N/A
Number and type of NGAP messages received over N2 interface	KPI	TS 38.413
Number and type of NGAP messages sent over N2 interface	KPI	TS 38.413
Number of NGAP RRC establishments and causes	KPI	TS 38.413
Number of active UE connections	KPI	TS 28.552
Number of UEs in the AMF	KPI	TS 28.552
Number of UEs in MM-REGISTERED state	KPI	N/A
Number of NAS 5GS messages received over N11 interface by the SMF	KPI	TS 24.501
Number of NAS 5GS messages sent over N11 interface by the SMF	KPI	TS 24.501

Number of NAS 5GS message decoding failures by the SMF	KPI	TS 24.501
Number of PDU sessions currently active in the SMF per DNN or slice	KPI	TS 28.552
Number of SUPIs in the SMF	KPI	TS 28.552
Number of configured IP addresses per SMF	KPI	N/A
Number of IP addresses currently in use per SMF	KPI	N/A
Number of IP addresses reserved for a detached user per SMF	KPI	N/A
Number of IP addresses associated with a detached user per SMF	KPI	N/A
Number of UPF sessions per DNN or slice	KPI	N/A
Number of GTP-U (N3) interfaces per UPF	KPI	TS 28.552
Number of IP (N6) interfaces per UPF	KPI	TS 28.552
Number of configured IP addresses per UPF	KPI	N/A
Number of allocated IP addresses per UPF	KPI	N/A
Number of allocated IP addresses per UPF	KPI	N/A
Number of forwarded packets per UPF	KPI	N/A
Number of dropped packets per UPF	KPI	N/A
Number of forwarded bytes per UPF	KPI	N/A
Number of dropped bytes per UPF	KPI	N/A
Number of N5 sessions	KPI	N/A
Number of N7 sessions	KPI	N/A
Number of provisioned SUPIs in the UDR	KPI	N/A
Number of provisioned data profiles in the UDR	KPI	N/A
Number of PFCP messages received	KPI	N/A
Number of PFCP messages sent	KPI	N/A
Number of PFCP message decoding failures	KPI	N/A
Overall CPU usage above 50% for more than a minute	Alert	N/A

Memory usage above 70%	Alert	N/A
Memory usage above 90%	Alert	N/A
Disk usage above 70%	Alert	N/A
Disk usage above 90%	Alert	N/A
Ethernet link down (for a specific interface)	Alert	N/A
Network function service down (for a specific network function)	Alert	N/A

9. Conclusions

This deliverable reported on the achievements related to the development of systems and methods for the automation of the AI@EDGE connect-compute platform. Updated descriptions of the internal design of the Network and Service Automation Platform (NSAP) have been provided. The NSAP is a framework for automation of network management that provides an environment for data-driven methods supporting decision making.

The data-driven methods are supported by a data collection pipeline, which collects, stores, and redistributes monitoring data for multiple consumers, and an ML model manager, which provides life-cycle management for machine-learning methods, including performance monitoring and re-training. The data collection pipeline and the model manager have been evaluated experimentally.

Three classes of data-driven AI/ML methods for network automation have been developed and experimentally evaluated. Methods for optimising the deployment of distributed AI controls placement and migration of services at the edge in several context, including federated learning. Methods for predicting or forecasting future performance form a basis for proactive data-driven network management services. Methods for monitoring and preparing data deal with monitoring the network and compute platform, or with preparing or generating data for training ML-based algorithms.

Two core components of the NSAP, the intelligent orchestrator component and the data collection pipeline have been implemented as software prototypes and integrated into the project's connect-compute platform. These components have been experimentally tested and evaluated together with the platform.

The achievements reported in the deliverable are largely corresponding to the overall project Objective 3 on designing and implementing a general-purpose network automation framework, capable of supporting flexible and reusable pipelines for the end-to-end creation, utilisation, and adaptation of secure and privacy-preserving AI/ML models.

Bibliography

[3gpp17] ETSI 3GPP TR 38.913 version 14.2.0 Release 14, 5G; Study on Scenarios and Requirements for Next Generation Access Technologies, 2017-05.

[Behravesh22] R. Behravesh, A. Rao, D. F. Perez-Ramirez, D. Harutyunyan, R. Riggio and M. Boman, "Machine Learning at the Mobile Edge: The Case of Dynamic Adaptive Streaming Over HTTP (DASH)," in IEEE Transactions on Network and Service Management, vol. 19, no. 4, pp. 4779-4793, Dec. 2022, doi: 10.1109/TNSM.2022.3193856.

[BinRuba22] S. Bin Ruba, N-E-H. Yellas and S. Secci, "Anomaly Detection for 5G Softwarized Infrastructures with Federated Learning", 6GNet 2022.

[ChiPhung22] D. Chi Phung et al., "An Open Dataset for Beyond-5G Data-driven Network Automation Experiments", in 6GNet 2022.

[Cover91] Cover, Thomas M.; Thomas, Joy A. (1991), Elements of Information Theory, John Wiley & Sons, p. 22

[D2.3] Adzic, J. (ed.). D2.3 Consolidated system architecture, interfaces specifications, and techno-economic analysis. AI@EDGE Deliverable 2.3, (H2020-ICT-52-2020).

[D3.1] Ahlgren, B. (ed.). D3.1 Initial report on systems and methods for AI@EDGE platform automation. AI@EDGE Deliverable 3.1, (H2020-ICT-52-2020).

[D4.2] Lentaris, G. (ed.). D4.2 Results on the validation of the AI@EDGE Connect-Compute Platform. AI@EDGE Deliverable 4.2, (H2020-ICT-52-2020)

[D5.2] Reti, D. (ed.). D5.2 Preliminary Validation and Use Case Benchmarking. AI@EDGE Deliverable 5.2, (H2020-ICT-52-2020)

[Diamanti22] A. Diamanti, J. M. S. Vílchez and S. Secci, "An AI-Empowered Framework for Cross-Layer Softwarized Infrastructure State Assessment," in *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4434-4448, Dec. 2022, doi: 10.1109/TNSM.2022.3161872.

[DuelHallen07] A. Duel-Hallen, “Fading channel prediction for mobile radio adaptive transmission systems,” Proceedings of the IEEE, vol. 95, no. 12, pp. 2299–2313, Dec. 2007.

[Eshratifar19] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, “Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services,” IEEE Transactions on Mobile Computing, vol. 20, no. 2, pp. 565–576, 2019.

[Georgiadis19] G. Georgiadis, “Accelerating convolutional neural networks via activation map compression,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 7085–7095.

[Gupta18] O. Gupta and R. Raskar, “Distributed learning of deep neural network over multiple agents,” Journal of Network and Computer Applications, vol. 116, pp. 1–8, 2018

[Hochreiter97] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, Dec. 1997.

[Horgan18] D. Horgan, J. Quan, D. Budden, G. Barth.Maron, M. Hessel, H. Hasselt, D. Silver, “Distributed Prioritized Experience Replay”, arXiv:1803.00933, 2018.

[Jacob18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 2704– 2713.

[Jiang19] Wei Jiang and H. D. Schotten, “Neural Network-based Fading Channel Prediction: A Comprehensive Overview”, IEEE Access, vol. 7, Aug. 2019, pp. 118112 – 118124

[Jiang20] W. Jiang and H. D. Schotten, "Deep Learning for Fading Channel Prediction," IEEE Open Journal of the Communications Society, vol. 1, Mar. 2020, pp. 320-323

[Ko18] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, “Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms,” in 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveil- lance (AVSS). IEEE, 2018, pp. 1–6.

[LeCun98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[Li20] Z. Li, E. Wallace, S. Shen, K. Lin, K. Keutzer, D. Klein, and J. Gonzalez, “Train big, then compress: Rethinking model size for efficient training and inference of transformers,” in International Conference on Machine Learning. PMLR, 2020, pp. 5958–5968.

[Llorens21] A. Llorens-Carrodeguas, S. G. Sagkriotis, C. Cervelló-Pastor, and D. P. Pezaros, “An Energy-Friendly Scheduler for Edge Computing Systems,” *Sensors*, vol. 21, no. 21, p. 7151, Oct. 2021.

[Lu19] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, Guangquan Zhang, “Resource management at the network edge: A deep reinforcement learning approach,” IEEE Network, vol. 33, no. 3, pp. 26–33, 2019.

[McMahan16] HB McMahan et al. “Federated learning of deep networks using model averaging. CoRR abs/1602.05629”.In: arXiv preprint arXiv:1602.05629 (2016).

[Mnih16] 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,” in International Conference on Machine Learning, vol. 48, p. 1928–1937, 2016.

[Nardini20] G. Nardini, D. Sabella, G. Stea, P. Thakkar, A. Virdis, Simu5G–An OMNeT++ Library for End-to-End Performance Evaluation of 5G Networks, IEEE Access 8 (2020) 181176–181191. doi:10.1109/ACCESS.2020.3028550.

[OranAIML] O-RAN Alliance, “O-RAN AI/ML workflow description and requirements 1.03”, O-RAN.WG2.AIML-v01.03 , June 2023

[OranArch] O-RAN Alliance, “O-RAN Architecture Description 9.0”, O-RAN.WG1.OAD-R003-v09.00, June 2023

[OranR1] O-RAN Alliance, “O-RAN R1 interface: General Aspects and Principles 5.0”, O-RAN.WG2.R1GAP-v05.00, June 2023

[PerezRamirez23] D. F. Perez-Ramirez, C. Pérez-Penichet, N. Tsiftes, T. Voigt, D. Kostić, and M. Boman. 2023. DeepGANTT: A Scalable Deep Learning Scheduler for Backscatter Networks. In Proc. 22nd International Conference on Information Processing in Sensor Networks (IPSN '23). ACM, New York, NY, USA, 163–176. *https://doi.org/10.1145/3583120.3586957*

[Schulman17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” CoRR, vol. abs/1707.06347, 2017.

[Schulman18] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High- Dimensional Continuous Control Using Generalized Advantage Estimation,” arXiv:1506.02438 [cs], Oct. 2018. arXiv: 1506.02438.

[Simu5G] Simu5G - 5G New Radio User Plane Simulator for OMNeT++ and INET. URL http://simu5g.org/

[Stock19] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. J´egou, “And the bit goes down: Revisiting the quantization of neural networks,” arXiv preprint arXiv:1907.05686, 2019.

[Sutton18] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: A Bradford Book, 2018.

[Vepakomma18] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, “Split learning for health: Distributed deep learning without sharing raw patient data,” arXiv preprint arXiv:1812.00564, 2018.

[Yellas22] N-E-H. YELLAS, B. Addis, R. Riggio and S. Secci, "Function Placement and Acceleration for In-Network Federated Learning Services," in CNSM 2022.