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

D4.1 Design and initial prototype of the AI@EDGE connect-compute platform

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101015922

D4.1 - Design and initial prototype of the AI@EDGE connect-compute platform	
WP	WP4 – AI@EDGE connect-compute platform
Responsible partner	ATOS
Version	1.0
Editor(s)	Enric Pages (ATOS)
Author(s)	Cristina Costa, Arfan Wahla (FBK), Akhila Rao, Shuai Zhu, Bengt Ahlgren (RISE), Flavio Brito (EAB), Enric Pages, Luigi Girletti (ATOS), Estefanía Coronado, Miguel Catalán, Claudia Torres (I2CAT), Chi-Dung Phung, Mario Patetta, Bilal Maaz, Stefano Secci (CNAM), Omar Anser, Jonathan Proeitto-Stallone (INRIA), Daniele Munaretto, Arif Ishaq (ATH), Antonino Albanese (ITL), Stelios Koumoutzelis, George Avdikos (8BELLS), Brendan McAuliffe, Paul Sutton, Tim Forde (SRS), Babak Mafakheri, Leonardo Goratti (SAFRAN)
Reviewer(s)	Nicola di Pietro (ATH), Babak Mafakheri (SPI), Roberto Riggio (UNIVPM), Cristina Costa (FBK)
Deliverable Type	R
Dissemination Level	PU
Due date of delivery	31/12/2021
Submission date	31/01/2022

Version History

Version	Date	Author	Partner	Description
0.1	05/07/2021	Enric Pages	ATOS	ToC
0.2	07/08/2021	Enric Pages, ALL	ALL	Modified ToC
0.3	15/09/2021	ALL	ALL	Initial contributions
0.4	08/10/2021	Cristina Costa, ALL	FBK	Final contributions and refinements
0.4	10/12/2021	Enric Pages	ATOS	Formatting

0.5	20/12/2021	Nicola di Pietro, Babak Mafakheri	ATH, SPI	Review document
0.6	03/01/2022	All	All	New version of the document with reviewers comments addressed
0.7	11/01/2022	All	All	New version of the document with reviewers comments addressed (2nd round)
0.7	21/01/2022	Cristina Costa and Roberto Riggio	FBK, UNIVPM	Final reviewed document
0.8	26/01/2022	Enric Pages, Luigi Girletti	ATOS, CNAM	Contributions from addressing final review
0.8	27/02/2022	Cristina Costa, Omar Anser, Jonathan Proeitto-Stallone	FBK, INRIA	Contributions from addressing final review
0.9	28/02/2022	Estefanía Coronado, Miguel Catalán, Claudia Torres	I2CAT	Contributions from addressing final review
1.0	31/01/2022	Irene Facchin	FBK	Final review

Table of Contents

List of Figures

List of Tables

Glossary	
3GPP	3rd Generation Partnership Project
4G	4th Generation of mobile communication networks
5G	5th Generation of mobile communication networks
5GAA	5G Automotive Association
5GC	5G Core network
6G	6th Generation of mobile communication networks
AGV	Automated Guided Vehicle
AI	Artificial Intelligence
AIF	Artificial Intelligence Function
AMF	Access and mobility Management Function
APN	Access Point Name
AR	Augmented Reality
AUSF	Authentication Server Function
BVLOS	Beyond Visual Line of Sight
C-V2X	Cellular Vehicular communication
COTS	Commercial Off-The-Shelf
CP	Control Plane
CPU	Central Processing Unit
CU	Centralized Unit
DE	Deliverable Editor

DL	Downlink
DNN	Data Network Name
DNS	Domain Name System
DSP	Digital Signal Processing
DU	Distributed Unit
EDA	Electronic Design Automation
EPC	Evolved Packet Core
FaaS	Function as a Service
FL	Federated Learning
FPGA	Field-Programmable Gate Array
FPV	First Person View
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphic Processing Unit
HIL	Hardware In the Loop
HITL	Human-in-the-loop
HSS	Home Subscriber Server
HO	HandOver
HW	HardWare
ICT	Information and Communication Technology
IFE	In-Flight Entertainment

IIoT	Industrial Internet of Things
IoT	Internet of Things
IoU	Intersection over Union
IP	Internet Protocol
I-UPF	Intermediate User Plane Function
KPI	Key Performance Indicator
LUT	Look Up Table
mAP	Mean Average Precision
MEC	Multi-access Edge Computing
ML	Machine Learning
MME	Mobility Management Entity
mMTC	massive Machine-Type Communication
MNO	Mobile Network Operator
MQTT	Message Queuing Telemetry Transport
MTC	Machine Type Communications
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NRF	Network function Repository Function
NSA	Non-StandAlone
NSSAI	Network Slice Selection Assistance Information

NSSF	Network Slice Selection Function
OWL	Web Ontology Language
PCF	Policy Control Function
PCIe	Peripheral Component Interconnect express
PCRF	Policy and Charging Rules Function
PDCP	Packet Data Convergence Protocol
PDU	Protocol Data Unit
PGW	Packet data network GateWay
QoS	Quality of Service
R16	Release 16
RAN	Radio Access Network
RAT	Radio Access Technology
REST	REpresentational State Transfer
RIC	RAN Intelligent Controller
ROS	Robot Operating System
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
RSU	Road Side Units
RTL	Return To Launch
RU	Radio Unit
SA	StandAlone

SBA	Service-Based Architecture
SDN	Software-Defined Networking
SGW	Service GateWay
SMF	Session Management Function
S-NSSAI	Single – Network Slice Selection Assistance Information
TAC	Tracking Area Code
TRL	Technology Readiness Level
UC	Use Case
UE	User Equipment
UL	Uplink
UP	User Plane
UPF	User Plane Function
URLLC	Ultra-reliable Low Latency Communications
VNF	Virtual Network Function
VR	Virtual Reality
V2I	Vehicle to Infrastructure
V2N	Vehicle to Network
V2V	Vehicle to Vehicle

Executive Summary

The main goal of AI@EDGE is to build a platform and a set of accompanying tools for enabling secure and automated management, orchestration and operation of AI-powered services over edge and cloud compute infrastructures, with close to zero-touch of the underlying heterogeneous MEC resources (network, storage, and compute resources). One of the key aspects to achieve this vision, is to develop a set of solutions broadly divided into two distinct areas: (i) solutions for the creation, utilization, and adaptation of secure and privacy-aware AI/ML models; and (ii) solutions managing distributed resources inside the telecom operators' infrastructure.

This document constitutes a key control point for the achievement of MS22 "First release of the AI@EDGE Platform". The report summarizes the work carried out in the scope of "WP4. Connect-compute platform" towards the design, prototype and early validation of a connect-compute platform supporting perceived zero-latency services using a mix of computing and connectivity resources. This document, "D4.1 Design and initial prototype of the AI@EDGE connect-compute platform", is the first of two iterative reports covering the AI@EDGE project period from M5 to M12. The document presents the overall description of the connect-compute platform design, the description of the HW accelerated solutions for AI/ML, the data-driven service lifecycle management solutions for the deployment, management, and monitoring of end-to-end AI-enabled applications, combined with a cross layer, multi-connectivity-enabled disaggregated RAN into a single connect-compute platform, allowing developers to take advantage of the new capabilities offered by 5G using well established cloud-native paradigms.

This report discusses the following research objectives: i) design and validate a connect-compute platform enabling the creation of network slicing, ii) extend ETSI MEC/NFV architectures with applications and models capable of providing the AI@EDGE platform with the context and metadata necessary to take automatically actionable decisions and to realize intelligent data and computation offload control and management of applications and services deployed over the decentralized and distributed AI@EDGE platform, iii) investigate different hardware acceleration solutions (FPGA, GPU, CPU) spanning from the terminals to the cloud for highly decentralized and distributed workload management, iv) analyse and compare dual-connectivity monolithic RANs with cross-layer multi-connectivity disaggregated RANs to see if dynamically adapts the network topology to the network conditions.

The achievements reported in this deliverable report the progress towards fulfilling the project's overall Objective 4 "To design, prototype, and validate a connect-compute platform supporting perceived zero-latency services using a mix of computing and connectivity resources" and Milestone MS22 "First release of the AI@EDGE Platform".

The analysis and results provided within this report belong to the first phase of the AI@EDGE connect-compute platform roadmap that includes the initial elicitation of technical requirements and architecture design together with the description of the computational environments provisioned for the development of the platform.

1. Introduction

This report includes the overall description of the connect-compute system to be delivered, involving resource management, data-driven service orchestration and lifecycle management, and security/privacy/data-protection mechanisms.

The AI@EDGE connect-compute platform has been designed to support a variety of virtualization technologies - VMs, containers, K8s pods as well as FaaS - which enable a very fine-grained exploitation of the available resources across the edge-cloud continuum. In addition, the AI@EDGE platform supports a distributed, multi-layer cloud deployment where orchestration mechanisms take place within both the cloud domain (centralized) and the telecom domain (distributed).

The purpose of the first period from M5 to M12 was the design of a connect-compute fabric integrating state-of-the-art cloud-native technologies - FaaS/serverless - with a disaggregated 5G Radio Access Network (RAN) supporting beyond R16 cross-layer, multi-connectivity. The resulting system will allow workloads to be intelligently spread and scaled across the connect-compute fabric according to their requirements. Moreover, AI@EDGE investigates on leveraging heterogeneous hardware acceleration solutions - CPU, GPU, and FPGA - to optimize energy consumption, performance, and security for specific AI-based workload types.

The content of this report describes the work carried out in the scope of "WP4. Connect-compute platform" towards the initial design, prototype and early validation of the AI@EDGE connect-compute platform. The systems and software modules described alongside the document directly relate to the ones described in WP3 as part of the Network Platform Automation.

The achievements reported in this deliverable report the progress towards fulfilling the project's overall Objective 4 "To design, prototype, and validate a connect-compute platform supporting perceived zero-latency services using a mix of computing and connectivity resources" and Milestone MS22 "First release of the AI@EDGE Platform"

Since the connect-compute platform is a core element of the AI@EDGE platform, this report has links with the technical work carried out across WP2, WP3 and with the validation scenarios described in the scope of WP5.

To get a better understanding of the content of this report, the suggested reading path is the following:

- "D2.1 Use cases, requirements, and preliminary system architecture" → This report provides a preliminary version of the AI@EDGE system architecture together with functional and system requirements associated with the use cases.
- "D2.2. Preliminary assessment of system architecture, interfaces specifications, and techno-economic analysis" → This report provides the initial AI@EDGE system architecture, the interfaces, and the protocols to be used for the data exchange.

To get a complete understanding of the AI@EDGE solution another recommended report to read is:

- "D3.1 Initial report on systems and methods for AI@EDGE platform automation" → This report summarizes technical platform requirements and outline initial systems and methods for the Network Platform Automation.

The primary target of the document is external technical personnel that are willing to adopt the AI@EDGE solution. Additionally, this document can be also valuable to internal AI@EDGE technicians (e.g., from WP3 and WP5) involved in the prototyping and implementation of the AI@EDGE platform.

The document's outline is as follows: the first section introduces the AI@EDGE connect-compute platform and its objectives. The second chapter describes in more detail the AI@EDGE connect-compute platform covering aspects like its component's specification, the envisioned development roadmap as well as the description of the computational environment where each of the modules will be developed and validated. The third chapter presents AI@EDGE support offered as a serverless platform for deploying event-driven, stateless functions. Section 4 covers the connectivity through disaggregated radio access (RAN). The fifth section describes the data-driven service lifecycle for AI-enabled applications. Section 6 covers the cross-layer, multi-connectivity specific subsystems and their roadmap for the project. Section 7 approaches the HW accelerated solutions for AI/ML considered during the first project period. Finally, section 8 focuses on the platform next steps.

2. The AI@EDGE Connect-compute Platform

2.1 Overview

The following section presents an overview of the AI@EDGE connect-compute platform (CCP), referring to the functional architecture, components specification, roadmaps, and features. In AI@EDGE, the Connect compute platform provides the fabric for AIFs orchestration and management across various edge levels/locations. It is also responsible for assuring the connectivity between the different system elements and for managing the available computing resources. It is one of two main components of the AI@EDGE Architecture (presented in D2.2), together with Network and Service Automation Platform (NSAP), which instead implements the system network automation D.3.1. In Figure 1, the Connect-compute Platform is shown, together with its relationship with the NSAP. The figure also shows the location options of the cloud resources, namely Far Edge, Near Edge and Cloud.

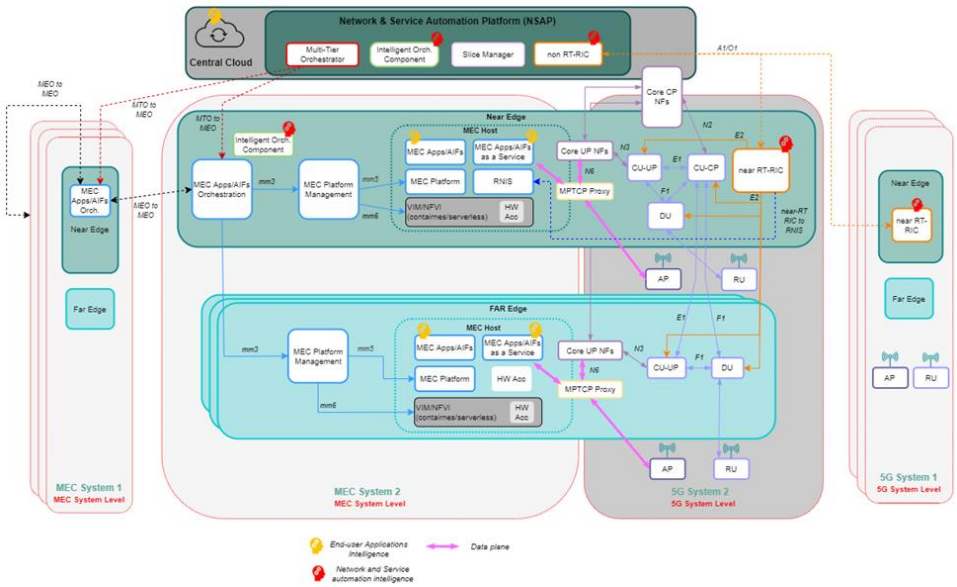


Figure 1 Draft system architecture of the AI@EDGE connected-compute platform.

Figure 1 depicts the Connect Compute Platform in detail, with the location of the platform components distributed between the Central Office and the Radio Access Sites. Multiple MEC systems are considered. A MEC system is a collection of MEC hosts and MEC management processes, which are necessary to run MEC applications. Each MEC host contains a MEC platform and the corresponding virtualization infrastructure, which provides compute, storage, and network resources to MEC applications. The MEC Platform provides the functionalities required to run applications at the edge, enabling them to provide and consume MEC services. The MEC Platform can provide itself with several MEC services, such as the RNIS service. The MEC host is strategically placed by the edges of the network to provide computation and storage capabilities near the access network and provides, among other advantages, lower latency. To this aim, the 5G traffic is steered towards the MEC host where it can be processed (more details on the integration of the MEC host with the 5G infrastructure are given in the following paragraph). The MEC

host can be therefore considered as an edge cloud able to host MEC applications and User Application The virtualization infrastructure provided by the MEC hosts in AI@EDGE will be based on Kubernetes, who manages containerized workloads and services and facilitates configuration and automation processes. The MEC hosts will also deploy a FaaS Platform, supporting event-based, stateless, serverless functions. Finally, the near-RT RIC will support AIFs relying on RAN data for network automation.

The AI@EDGE connect-compute platform envisions an architecture for multi-site and distributed environments. To fulfill this challenge, the consortium is currently evaluating the benefits and drawbacks offered by both the ETSI MEC and the ETSI MEC in NFV architectures. Furthermore, it must be decided which choice is better aligned with the system requirements for the deployment of the CCP, as detailed in D2.1. This document describes the different components and software artefacts being considered for each of the options and the implications they may have to drive the final decision.

Both deployments offer the same distributed layout of the platform, being it composed of a set of MEC systems, which at the same time comprise a group of near edge and far edge nodes. The far edge nodes have limited computational resources and minimal hardware acceleration capabilities compared to the near edge nodes. Moreover, these nodes lack of orchestration capabilities. By contrast, the near edge nodes not only comprise specific modules for the orchestration of applications, but they possess greater computational capacity and more advanced hardware acceleration units (GPUs and FPGAs).

At the same time, they both cover certain exclusive functionalities. On the one hand, the ETSI MEC architecture aims to provide a generic vision for any sort of virtualization infrastructure, which provides greater flexibility to integrate functionalities such as the serverless platform and introduces a more light-weight orchestration and management. On the other hand, the ETSI MEC in NFV variant has aimed to make it possible the adoption of NFV virtualized network functions on the same infrastructure, which has facilitated the progressive transition of telco industries, academia and tools developments. This has also enabled the introduction of very well-known orchestration and management tools, such as Open-Source MANO (OSM), and their adoption in a multitude of previous European projects. While on the one hand the second one may be more mature and could be leveraged given the availability of existing software, it is also true that the former architecture provides greater flexibility to fill the gaps and develop the tools needed to fulfill the AI@EDGE Connect-Compute Platform's requirements.

Moreover, this section also presents the roadmap expected for the inclusion of functionalities required in the Connect-Compute Platform. Notice that the roadmap presented in this section corresponds to the initial project planning, responsibilities and roles played by the various components.

2.2 Functional Architecture

As described in more detail in D2.1, the functional architecture is composed of several tiers, located both at the cloud and at the edge.

Taking as a reference this functional architecture,

Figure 2 and Figure 3 depict its deployment options for the connect-compute platform based on the ETSI MEC in NFV and the ETSI MEC architectures, respectively. It should be noted that independently from the option, the location and functionalities provided by the NSAP are the same, and they just differ in the way and modules involved in the orchestration of applications, as is described later. Both figures show a system comprising various MEC systems, and the NSAP. A connection to the cloud could be implemented if more levels are developed, although for the sake of simplicity the aforementioned figures do not depict the computational capabilities of such a cloud tier, and just provides the view of the NSAP together with the near and far edges.

In both architectures (ETSI MEC and ETSI MEC in NFV) the AI@EDGE Network & Service Automation Platform (NSAP) is presented, which contains besides the multi-tier orchestrator, the non-RT RIC and Intelligent Orchestrator Component.

In both cases, the deployment and management of applications follow a combination of hierarchical and distributed approaches. The initial request for deployment is meant to be received by NSAP, and specifically to be managed by the multi-tier orchestrator (MTO). This entity will decide between Cloud or MEC, by intelligent external selection or some other method to be decided, what happens is that it gets to a MEO, by intelligent external selection or some method to be decided. This is the main difference between the two architectures, in the ETSI MEC, the MEO can do it; however, in the ETSI MEC in NFV, this duty goes to the NFVO. This vision intends that the platform and the applications deployed can continue its execution even if the MTO goes down, since the rest of functionalities would be managed locally at each MEC system.

In terms of modules and functionalities, in addition to the management of all entities in the form of VNFs, the main difference between the two deployment options is the existence of one or more entities dedicated to the onboarding and orchestration operations.

Figure 2 shows the presence, at each MEC system, of a MEC Application Orchestrator (MEAO), an NFV Orchestrator (NFVO) and a VNF Manager (VNFM), while in Figure 3, there is a single entity dedicated to orchestration matters, namely the MEC Orchestrator (MEO). While in the second vision the MEO oversees onboarding, placing and instantiating the applications, in the former vision the onboarding and instantiation are offloaded to the VNFM and NFVO, respectively, given that in this architectural version both applications and MEC systems are effectively deployed as VNFs.

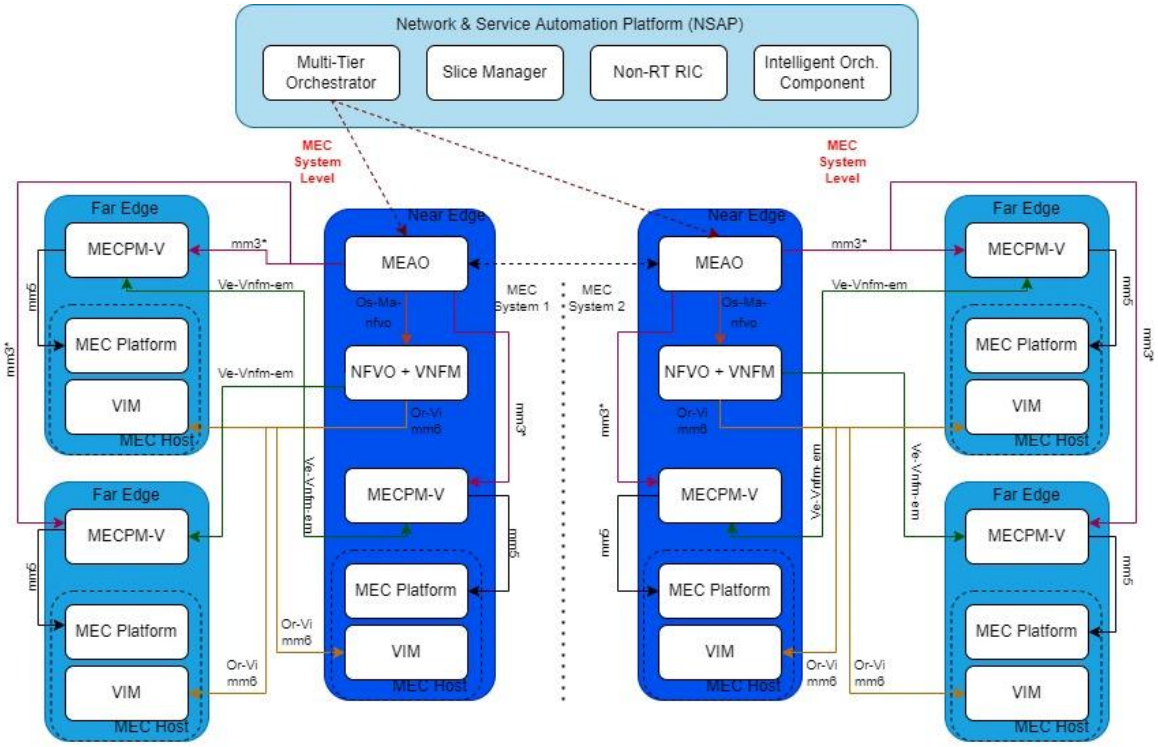


Figure 2 Connect Compute platform (version aligned with the ETSI MEC in NFV architecture)

In the case in which the CCP's deployment follows the ETSI MEC in NFV architecture (

Figure 2), upon receiving a request from the multi-tier orchestrator, the MEAO delegates the lifecycle management of the applications to the NFVO and the VNFM. In this case, the MEC platform is consequently also deployed as a VNF, still exposing the MEC services to be consumed by the deployed applications.

The Os-Ma-nfvo interface between MEAO and NFVO is used for network service instantiation and update, primarily used to manage Network Services, i.e., several VNFs connected and orchestrated to deliver a service [1]. The NFVO and VNFM, from the MEC System Level, which interact directly with VIM in the near and far edges, via the Or-Vi and the mm6 interfaces, respectively, located in each MEC system, have the function to manage virtualized resources e.g., to realize the application lifecycle management and to manage capacity. The MECPM-V (MEC Platform Manager - NFV) is related to the MEC Platform, via mm5 interface, to perform platform configuration, configuration of the application's rules and requirements, application lifecycle support procedures, management of application relocation.

The Ve-Vnfm-em interface between the VNFM and MECPM-V manages the lifecycle of the MEC platform. The MEAO communicates with MECPM-V, through the mm3* interface for the management of the application lifecycle, application rules, requirements, and keeping track of available MEC services [2].

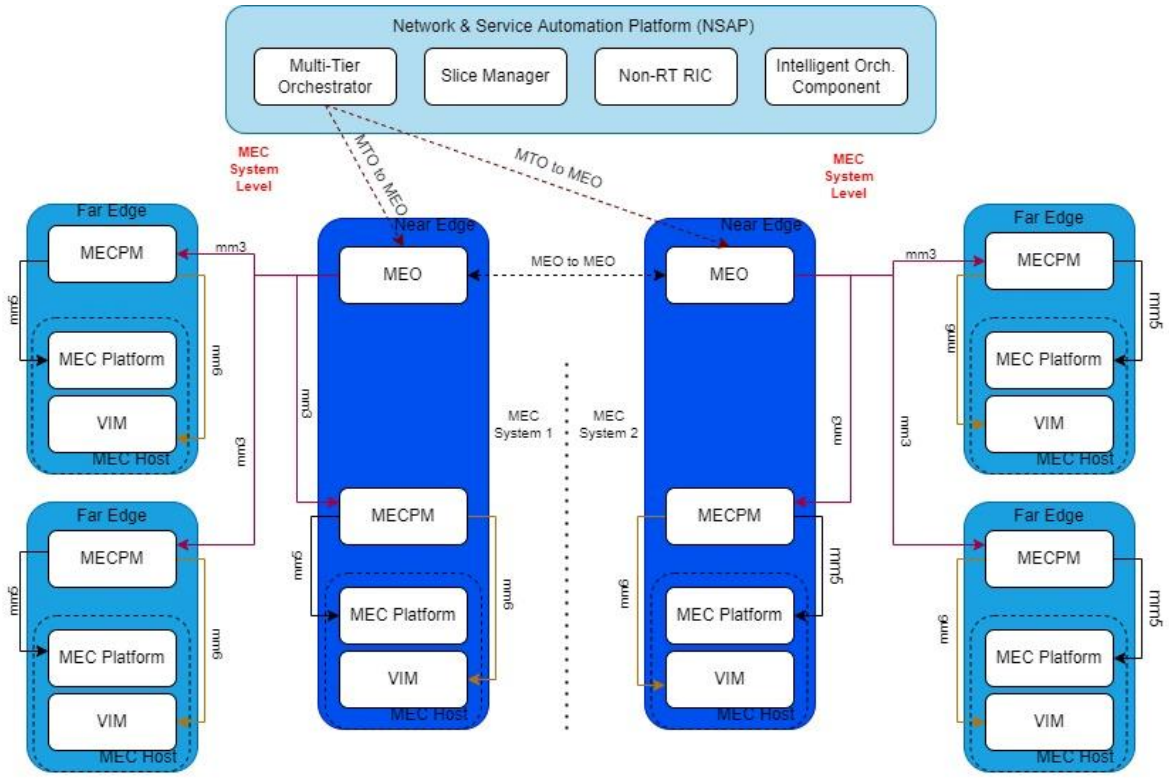


Figure 3 Connect Compute platform (version aligned with the ETSI MEC architecture)

In Figure 3 the CCP is aligned with the ETSI MEC architecture, where again the MTO, defined in [3]is responsible for instantiating MEC Apps. As opposed to the previous case, here the MEO is the entity directly responsible for the lifecycle management of the applications, and directly interacts with the MEC platforms at the different far and near edges. The mm3 interface between the MEO and the MEC platform manager allows keeping track of the available MEC platforms and services and to provide the management of the application lifecycle (limited to this last purpose in the mm3* interface at ETSI MEC in NFV architecture). The *mm5* interface between the MEC platform manager and the MEC platform is used to perform platform configuration, the application rules and requirements configuration and application lifecycle support procedures. The *mm6* interface is used to manage virtualized resources.

The interfaces between NSAP and MEC systems, as well as between MEC systems, are described in more detailed in D2.2.

2.3 Components Specification

Intelligent orchestrator component:

The NSAP provides an environment for data-driven, AI/ML-based methods and algorithms that support decision-making for the purpose of automating various network management tasks, as described in more detail in project Deliverable 3.1 [4]. The intelligent orchestrator component serves as a coordinator of and an interface to these intelligent decision-making functions, which can be utilised by the other NSAP components and the connect-compute platform. The component is also anticipated to manage the data pipelines that make monitoring data and other data sources available in a consistent manner for the data-driven methods of the NSAP, but possibly also to components external to the NSAP. The design of the component is still under discussion.

Multi-Tier Orchestrator:

It is necessary for coordinating various orchestrators. The idea of the MTO, was implemented from scratch for the 5GCity project (https://www.5gcity.eu/). It mediates the triggering of high-level actions such as instantiation and monitoring of (network) services and configurations. The MTO includes an Abstraction API, used to trigger the required API invocation chains on the different orchestrators when a high-level action is performed. It is not a simple abstraction layer that translates high-level API calls to low level invocations. It contains a layer of intelligence (forwarding and coordination logic) and adds in the descriptor the corresponding VNF to the OSM project. The MTO does not deal directly with descriptors management, and does not perform package onboarding, but it is able to handle various orchestrators in the cloud and the edge, as is shown in Figure 4.

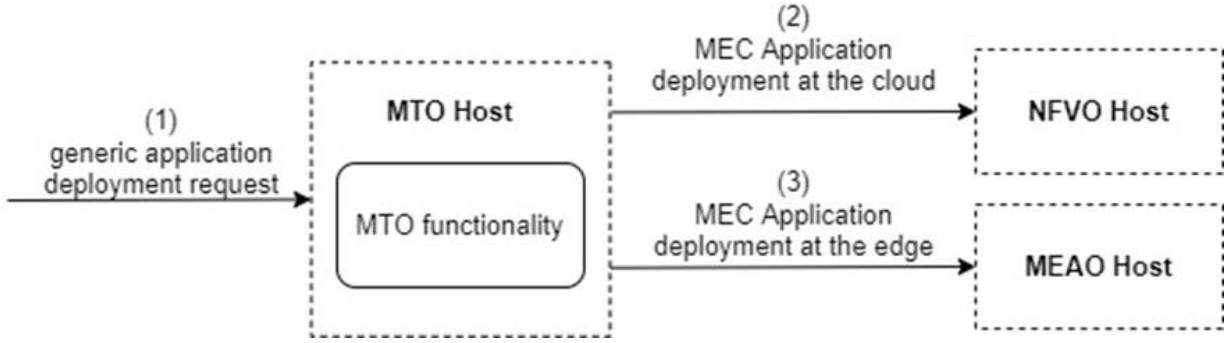


Figure 4 MTO-based scenario

Slice Manager:

The Slice Manager is a key component to provide the slicing features to AI@EDGE. In particular, with the input of a slice template, this component is able to create slice instances and to control their life cycle. Moreover, it manages the deployment of the slices over multiple MEC Systems. The slice template used by the slice manager will contain a set of requirements that needs to be mapped by the slice manager over the logical sliced network. This is possible to be done by Slice Manager leveraging the control over the different NFVOs and VIMs (for the deployment of VNFs of the slice), and the Core and RAN (with the ability to allocate network resources to a slice). Currently, a specific task force (i.e. Slicing Task Force) is being formed in the project to define and implement the slicing functionalities in AI@EDGE. The task force is composed of the different partners with expertise in the different topics connected to slicing (i.e. MEC, RAN and CORE). The details on the implementation and the final design of this component will be provided by the Slicing Task Force in the future deliverable D4.2.

MEO, MEAO:

As stated before, these two elements share certain functionalities in the two ETSI architectures described above. Independently from the deployment option, this is a software component that is in the design phase to be implemented in the project. It is the core functionality at the MEC system level management. The MEO is responsible for the following functions: maintaining an overall view of the MEC system based on deployed MEC hosts, and available resources, keeping a record of on-boarded packages; selecting appropriate MEC host(s) for application instantiation based on constraints, such as latency, available resources, and available services; triggering application instantiation and termination; triggering application relocation as needed when supported. Additionally, each MEO also collects the telemetry from the far and near edges under its MEC system for orchestration and management purposes and interacts directly with the VIM to achieve this purpose. This telemetry could be implemented through message brokers or REST APIs. As mentioned before, the MEAO in the ETSI MEC in NFV architecture would perform similar functionalities except for the offloading of the onboarding and lifecycle management tasks to the NFVO.

MEC Platform, MECPM, MECPM-V:

The MEC Platform has all the functionalities required to run MEC applications on a particular virtualization infrastructure and enables them to provide and consume MEC services. For example, such functionalities include providing services, offering an environment where the MEC applications can discover, advertise, consume, and offer MEC services. The MECPM is responsible for: managing the lifecycle of applications including informing the MEO of a relevant application-related event; and providing element management functions to the MEC platform. The MECPM also receives virtualized resources, fault reports and performance measurements from the virtualization infrastructure manager for further processing.

This role will be taken by the LightEdge software [5] in the project. LightEdge is a microservice-based implementation of the ETSI MEC Architecture. LightEdge is meant to be deployed over a Kubernetes cluster inside a dedicated namespace and is released under an APACHE 2.0 License.

LightEdge consists of the following components:

- lightedge-runtime, the nexus LightEdge component. This microservice keeps track of the other active microservices. Each microservice on bootstrap registers with this component making itself available to the rest of the platform.

- lightedge-rniservice-mananger, the microservice implementing the Radio Network Information Service (RNIS). Together with 5G-EmPOWER it allows MEC apps to obtain information about the RAN.

Among its Top-Level features are:

- Designed to natively run in a Kubernetes environment
- Supports opensource and commercial LTE eNodeBs and EPCs
- Implements standard ETSI MEC service interface.

VIM:

This functional block is responsible for allocating, managing, and releasing virtualized (compute, storage, and networking) resources of the virtualization infrastructure; preparing the virtualization infrastructure to run a software image. In the AI@EDGE project, Kubernetes performs the role of VIM. Kubernetes is a container orchestration system for automating application deployment, scaling and management. In this sense, the deployment designed envisions that each MEC system will be part of the same Kubernetes clusters, and that therefore all MEC platforms of all edges (near and far) in the same MEC system (and therefore managed by the same MEO/MEAO), are part of the same Kubernetes cluster, each of them deployed on a worker node. Recent versions of Kubernetes have brought in many enhancements that help deploy CNFs (Cloud-Native containerized Network Functions). The Kubernetes features/plugins that the CNF will benefit from are CPU Manager, Huge Pages, Topology Manager, Multus CNI, SRIOV CNI, SRIOV Network device plugin for SRIOV/DPDK.

NFVO + VNFM:

This functional block is responsible in the ETSI MEC in NFV architecture for managing the network services' lifecycle, including the MEC app instances, treating each MEC app instance as a VNF instance. The VNFM, by one side, oversees handling the lifecycle of the MEC platform using standard NFV LCM procedures, and for the other, managing the lifecycle of the MEC apps, treating each MEC app instance as a VNF instance. For instance, given the wide adoption of OSM, this software could provide the functionalities of both NFVO and VNFM if the MEAO is on top as a software layer telling OSM what to do and controlling the synchronization across MEC systems. OSM is an ETSI-hosted initiative to develop an Open Source NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV [6]. OSM supports container-based services. The OSM approach aims to minimize integration efforts. It is capable of modelling and automating the full lifecycle of Network Functions [7].

2.4 RoadMap and Features

The roadmap of the connect-compute platform has been divided into four main phases that aim to evolve the platform from the initial building blocks of the consortium to a whole vision including the requirements and capabilities offered by the AI@EDGE project.

These four phases have a total duration of 30 months. However, the length of the phases is not the same. The initial phase (phase 1) is the longest one with a total duration of 12 months given the need of the consortium to properly perform a requirement elicitation process and provide an architectural design that covers the identified requirements. The rest of the phases, by contrast, have a duration of 6 months each.

The main difference between the two first and the two last phases is the specific cellular network and local breakout (LBO) technologies used at the various MEC hosts. The first two will use 4G/5G NSA, with the SGW+PGW at the MEC hosts, while the last two will use 5G NSA/5G SA, with SMF + UPF deployed at the MEC hosts. Nevertheless, it is worth highlighting that the roadmap regarding the rest of the features to be included in the CCP, is independent of NSA/SA implementation and does not suffer changes when passing from phase 2 to phase 3.

Figure 5 presents the phases and some features determined for each one. Although this plan was structured at the beginning of the project, its status is periodically revised and modified if needed to better suit the requirements to be fulfilled. Below we provide more details about the requirements incorporated in the platform during the lifetime of the project.

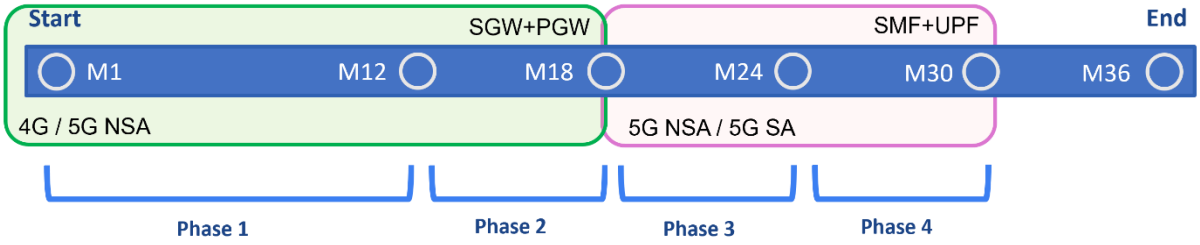


Figure 5 Roadmap and features

Phase 1. This phase will be based on 4G/5G NSA with SGW+PGW LBO integrated with the MEC host. From the MEC system perspective, simple MEC applications will be pre-deployed at all MEC sites without providing migration or onboarding features. Moreover, a preliminary integration of serverless basic functionalities will be considered. This first phase will not include hardware acceleration capabilities but, however, parallel in-lab research and development on GPU and FPGAs is being performed. Concerning the RAN, a preliminary version of the transport layer Wi-Fi and cellular aggregation solution will be incorporated. In addition, an initial definition of E2/A1 interfaces and services in the near-RT and non-RT RICs is being considered. The orchestration capabilities, management components, and the NSAP are being investigated in parallel to the networking/MEC components.

Phase 2. This phase aims to be an evolution from phase 1 providing 5G NSA connectivity with SGW+PGW LBO at the MEC hosts. On that evolutionary road, the serverless and function as service functionalities will also be enhanced, and the MEC applications are expected to be deployed as required. From the RAN perspective, the first subset of E2 messages and services will be implemented. Moreover, a more complete version of the multi-connectivity part will be integrated in the testbed. In addition, initial solutions for supporting 5G network slicing will be developed and made available at the end of Phase 2.

Phase 3. In this phase it is expected to happen the main change in terms of radio access technology, having an evolution from 5G NSA to 5G SA with SMF/UPF LBO by the end of such a phase. This will represent the initial testing phase for supporting slicing capabilities between the gNBs used in the project to the 5GCore. At this stage, MEC application orchestration capabilities will be incorporated in the platform following stateless approaches and exposing the necessary APIs between the NSAP and the MEC orchestrator. Moreover, the MEC-to-MEC interfaces will have a preliminary description. On the other hand, the serverless platform will continue extending functionalities in order to provision Serverless Functions to the MEC APPs and AIF and provide support to additional event sources such as message-based ones, as it is detailed in the next sections. From the multi-connectivity perspective, the consortium is planning to integrate PDCP features on a small-scale prototype.

Phase 4. This last phase will maintain the same radio access technology and local breakout as the previous one, namely 5G SA with SMF+UPF LBO. Concerning the connect-compute platform the MEC orchestrator will incorporate more mature features and will allow the migration of applications across various sites, with the implementation of the MEC-to-MEC interface. The final version of the serverless platform will be consolidated and incorporated in the CCP. With respect to the radio and core networks, final features will be added and tested in the form of S-NSSAI support pre-established in the network. The MEC hosts will count at this phase with hardware acceleration features across the various tiers, which could be used by the orchestrator to instantiate the applications expressing such requirements. At this stage, the E2/A1 interfaces will be finalized and tested between the near-RT and the non-RT RIC. Finally, the cross-layer solutions and experimental testing will be also considered from the multi-connectivity perspective.

2.5 Provisioning of Integration Testbed

The Integration Testbed hosted in the FBK premises is a research testbed that includes both computational (Intel NUCs) and radio resources (a mix of Ettus B210/X310 boards with the associated computing nodes). The entire testbed is designed with cloud-native principles and includes non-3GPP technologies such as Wi-Fi. This testbed has been conceived for being used for pre-integration of WP4 results and for providing a playground for testing the integrated connect-compute platform in various configurations and scenarios. The various components of the connect-compute platform, developed by all WP4 tasks, will be deployed in the testbed as soon as they become available and integrated with existing infrastructure (described in section 2.5.2).
Testbed Roadmap

Two main set-ups characterize the Integration Testbed Roadmap.

- **Integration Phase 1 and 2: 4G/5G Non-standalone (NSA) Integration Testbed set-up**,
    - deploying gNodeB/eNodeB,
    - 5G/4G User Equipment (UE),
    - 4G Evolved Packet Core (EPC).
- **Integration Phase 3 and 4: 5G Standalone (SA) Integration Testbed set-up,**
    - deploying 5G gNodeB,
    - 5G UEs,
    - 5G Core.

Integration Phases 1 and 2: 4G/5G NSA Integration Testbed set-up

The two phases differ on the adopted radio access and core network technologies and how the IP packets are routed to the MEC applications. Steering traffic to/from MEC applications is achieved by configuring the MEC's local Domain Name System (DNS) and the MEC host's data plane accordingly. How the data plane is managed depends on the point where the MEC host is installed in the 4G architecture. In a 4G/5G NSA many choices are possible, but all in all they can be condensed down into some base scenarios, i.e. distributed EPC, distributed service/packet data network gateways (S/PGW).

In Phase 1, the SGW and PGW entities are deployed at the edge site, whereas the control plane functions such as the Mobility Management Entity (MME) and Home Subscriber Service (HSS) are located at the operator's core site. The MEC host's data plane connects to the PGW over the SGi interface.

The SGW and PGW run as VNFs on the NFV platform within a single hardware component provided by ATH. The local SGW selection is performed by the central MME according to the 3GPP standard DNS procedures and based on the Tracking Area Code (TAC) of the radio to which the UE attaches. The 5G-NSA architecture implemented in Phase 1 is depicted in Figure 6.

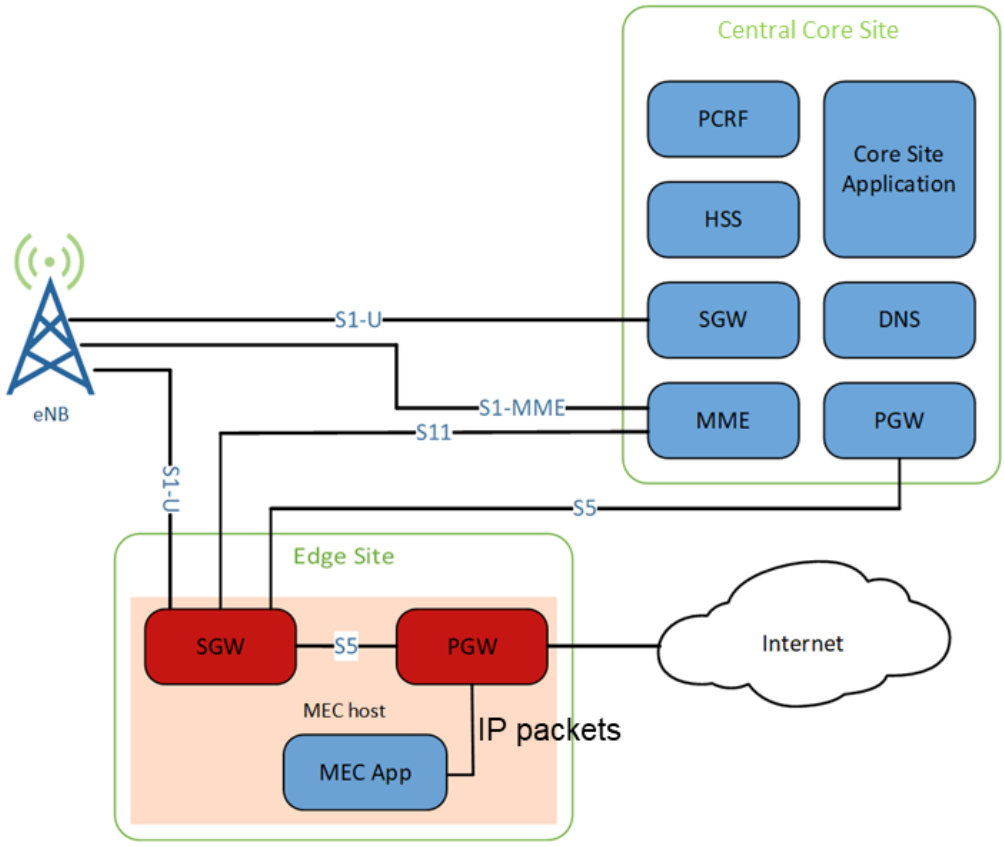


Figure 6 Integration Testbed Phase 1 - 4G/5G NSA

Integration Phases 3+4: 5G SA Integration Testbed set-up

In the 5G SA deployment, the 5G Core Network provides the means to select traffic to be routed to the applications in the local data network. UPFs can be seen as a distributed and configurable data plane from the MEC system perspective. The introduction of a local UPF by the edge site allows to terminate a PDU Session locally and redirect the traffic to the Local Data Network and MEC Apps.

The 5G SA architecture of the testbed is depicted in the following figure:

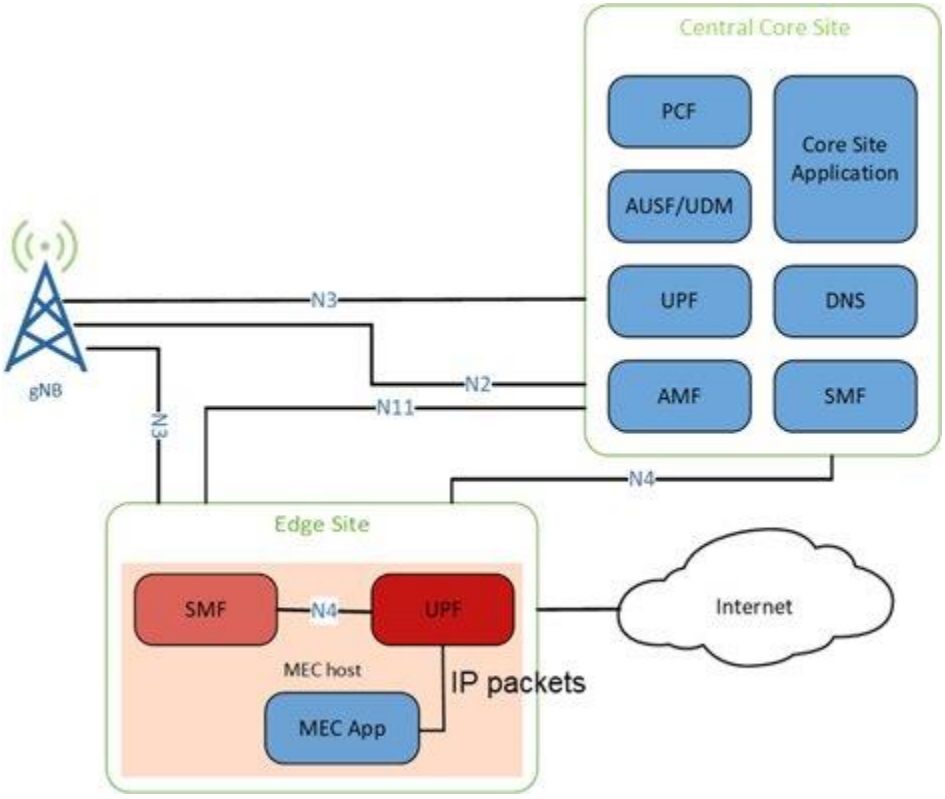


Figure 7 Integration Testbed Phase 2 - 5G NSA

The integration testbed roadmap is synchronized with the availability of the releases of the SRS software radio solutions as follows:

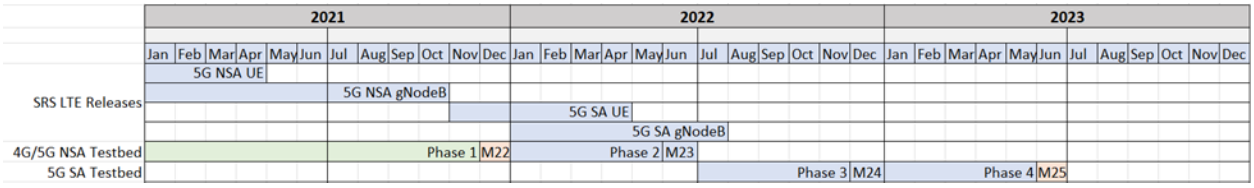


Figure 8 Integration testbed roadmap

2.6 Testbed components

RADIO functionalities and equipment

The testbed will rely on srsRAN, the SRS's free and open-source 4G and 5G software radio suite. srsRAN features both UE and eNodeB/gNodeB applications and can be used with third-party core network solutions to build complete end-to-end mobile wireless networks.

The srsRAN suite currently includes (included in the testbed phase 1):

- srsUE: a full-stack 4G and 5G NSA UE application
- srsENB: a full-stack 4G eNodeB and 5G NSA gNodeB application

The full-stack 5G SA gNodeB application (srsGNB) is expected to be available in 2022 and will be incorporated in the phase 2 of the testbed.

From HW point of view the testbed will rely on:

- 2x ETTUS USRP B210 [8], a fully integrated, single-board, Universal Software Radio Peripheral (USRP™) platform with continuous frequency coverage from 70 MHz – 6 GHz
- 2x ETTUS USRP X310 [9] a high-performance, scalable software-defined radio (SDR) platform for designing and deploying next generation wireless communications systems.
- The testbed will also deploy a Wifi OpenWRT Access Point with the purpose of deploying the multi-connectivity set-up.

Core network functionalities and equipment

ATH provides the core network functions in 4G/5G-NSA for Phase 1 and 5G SA for Phase 2 as a "network in a box" solution, hence running as a virtualized software over common-off-the-shelf hardware servers.

In the 4G/5G NSA version, included in the Testbed Phase 1, the box implements a local PGW/SGW that steers the traffic to the MEC host, and a full EPC. The EPC implements the standard 3GPP LTE network elements as follows: MME for mobility management, HSS for access and authentication procedures, Policy and Charging Rules Function (PCRF) for policy and charging control, PGW for providing access to Packet Data Networks, SGW for anchoring PDU sessions and DNS to resolve the IP address which is to be used for the packet gateway.

It will then be upgraded during the Testbed Phase 2, to a 5G SA implementation, with a local UPF, that steers the traffic to the MEC host, and a full 5G core network (5GC). The 5GC implements as well standard 3GPP 5G network elements as follows: the Access and Mobility Management Function (AMF) for mobility management, Authentication Server Function (AUSF) for access and authentication procedures, Policy Control Function (PCF) for policy and charging control, UPF for the data plane, Session Management Function (SMF) for managing the data plane sessions, Network Slice Selection Function (NSSF) for managing network slices, Network function Repository Function (NRF) for network function discovery.

Such functionalities are and will be instantiated in a Dell 240 server, with VMware as hypervisor. The HW has the following technical features:

- Intel Xeon CPU E-2146G 3.5 GHz 6C/12T
- 32 GB RAM
- x 1 TB 7.2 K RPM SATA 6 Gbps 512n 3.5in Cabled Hard Drive
- x 1 Gb Ethernet

Edge Host

The edge host contains the virtualisation infrastructure which provides compute, storage, and network resources, for the purpose of running mobile edge applications:

- **VIM** - the chosen virtualization infrastructure is container-based, relying on a Kubernetes deployment. Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

- **MECPM** – LightEdge [10, 11]
- **Serverless Platform** – Nuclio [12] Platform has been chosen as the FaaS platform for the integration testbed.

NFVO (edge host orchestration)

A server is deployed with an Open Source MANO (OSM) over Kubernetes installation that is used for the Edge Host deployment and management.
Testbed first set up

The first testbed set-up includes basic functionalities: one Radio Node and UE, one Edge Host and the ATH Box, with the 4G/5G NSA based architecture. The set-up is depicted in the figure:

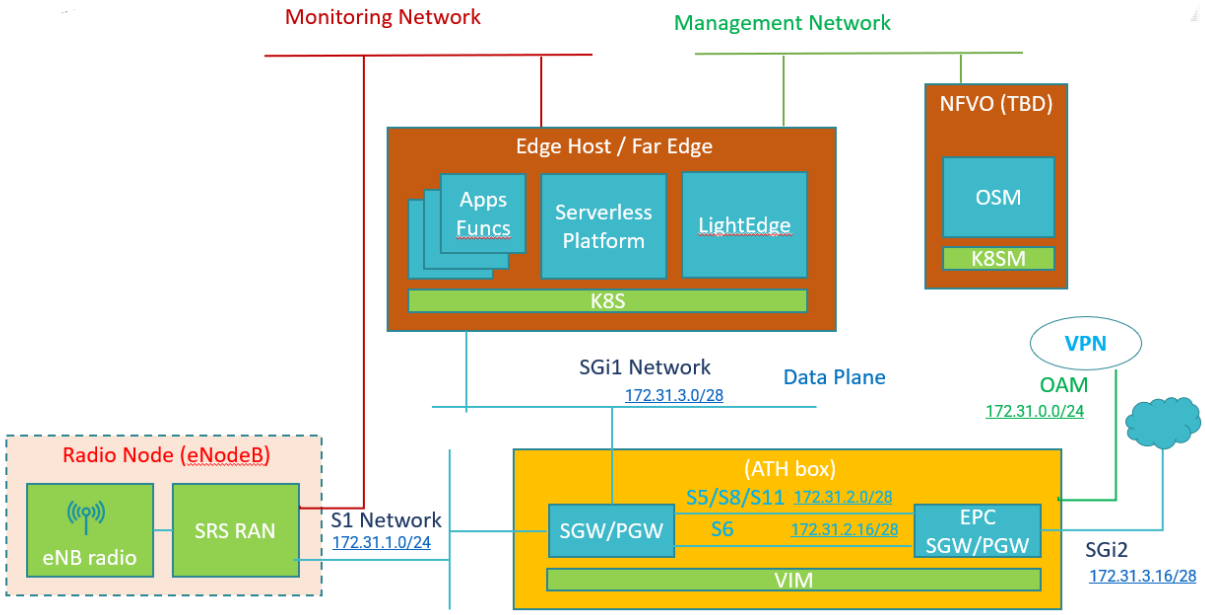


Figure 9 Testbed first setup

2.7 Distributed testbed with ICCS site

To allow conducting early tests of the hardware acceleration solutions, we plan to connect the testbed-cluster at ICCS premises with the integration testbed with a VPN connection. The cluster at ICCS will act as an Edge site that will be integrated with the connect-compute platform deployed in the main Integration Testbed at the FBK site, thus allowing developing and testing the acceleration solutions of AI@EDGE.

A detailed description of the testbed is given in section 0.

3. Serverless Platform

In AI@EDGE we consider Function as a Service (FaaS) based serverless computing platforms. FaaS platforms deploy event-driven, stateless, and often short-lived functions (i.e., stateless execution of remote functions). Computation logic is written as small pieces of code that respond to various set of events. The Cloud Native Computing Foundation (CNCF) states that *"Serverless computing refers to the concept of building and running applications that do not require server management. It describes a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment."* [13] There are many examples of serverless platforms: *AWS Lambda, Azure Functions, IBM Cloud Functions based on Apache OpenWhisk, Google Cloud Functions, Huawei Function Stage and Function Graph, Kubeless, iron.io, function, fission, Nuclio.*

The serverless paradigm has various advantages. First, the required amount of resources to a particular application/task is committed: we can have as many instances as necessary of serverless function, but only when they are needed. Resources are utilized for just the time needed to execute an invoked function. When there is no demand for the functions, the cost of used resources goes near zero, but it scales to as many instances (with some limits) as needed to meet an increased traffic demand. The drawback of this approach is the inherent cold start delay. Various approaches exist to minimize it, such as the deployment of ephemeral instances of each function as necessary (Caching) and generic containers, and they depend on the platform design.

The CNCF identifies the following key elements of a FaaS solution:

- **Event sources** – trigger or stream events into one or more function instances
- **Function instances** – a single function/microservice, that can be scaled with demand
- **FaaS Controller** – deploy, control and monitor function instances and their sources
- **Platform services** – general cluster or cloud services used by the FaaS solution (sometimes referred to as Backend-as-a-Service)

The relationship between these elements is shown in the diagram below:

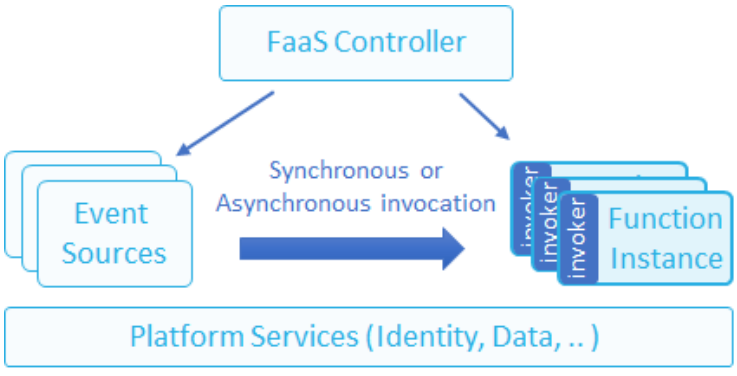


Figure 10 FaaS solution key elements (source: CNCF White Paper)

Open-Source platforms provide a portable way to develop serverless applications and reduce platform lock-in. Popular serverless frameworks rely on Kubernetes APIs to orchestrate and manage the serverless

functions. Often, the FaaS Platform itself is deployed as Kubernetes containers. The typical approach is to extend and provide the Custom Resource Definition (CRD) features necessary to create and deploy the container pods (group of containers). Functions can run from anywhere on any machine, as long as they are still in communication with the rest of the application. Serverless frameworks leverage Kubernetes network model to export services (cluster of function pods) and to route requests to specific functions. They depend primarily on Kubernetes for:

- Configuration management of containers and pods.
- Pod scheduling and service discovery.
- Update roll-outs for functions; and
- Replication management.

When using a FaaS platform, applications should be designed using **serverless principles**. Typically, serverless functions should be used when the workload is:

- Asynchronous, concurrent, easy to parallelize into independent units of work
- Infrequent or has sporadic demand, with large, unpredictable variance in scaling requirements
- Stateless, ephemeral, without a significant need for instantaneous cold start time
- Highly dynamic in terms of changing business requirements that drive a need for accelerated developer velocity

Resource allocation, communication of user data, and the execution of functions are abstracted from the developer. The allocation of infrastructure resources and the execution of the code is done dynamically, typically, in on-demand instantiated containers.

Serverless functions are **Event-driven**. Each Function performs one action only triggered by events originating from multiple heterogeneous event sources such as databases, message queues, or streaming platforms. The Functions can be invoked from different event sources depending on the different use-cases. Event sources can be grouped in the following way:

- Synchronous Request (Req/Rep), e.g., HTTP Request, gRPC call. The client issues a request and waits for an immediate response. This is a blocking call.
- Asynchronous Message Queue Request (Pub/Sub), e.g., RabbitMQ, AWS SNS, MQTT, Email, Object (S3) change, scheduled events like CRON jobs. The event is associated with messages published to an Exchange and distributed to subscribers.
- Message/Record Streams: e.g., Kafka, AWS Kinesis, AWS DynamoDB Streams, Database CDC. A stream can be produced from messages, database updates (journal), or files (e.g., CSV, JSON, Parquet)
- Batch Jobs, e.g., ETL jobs, distributed deep learning, HPC simulation

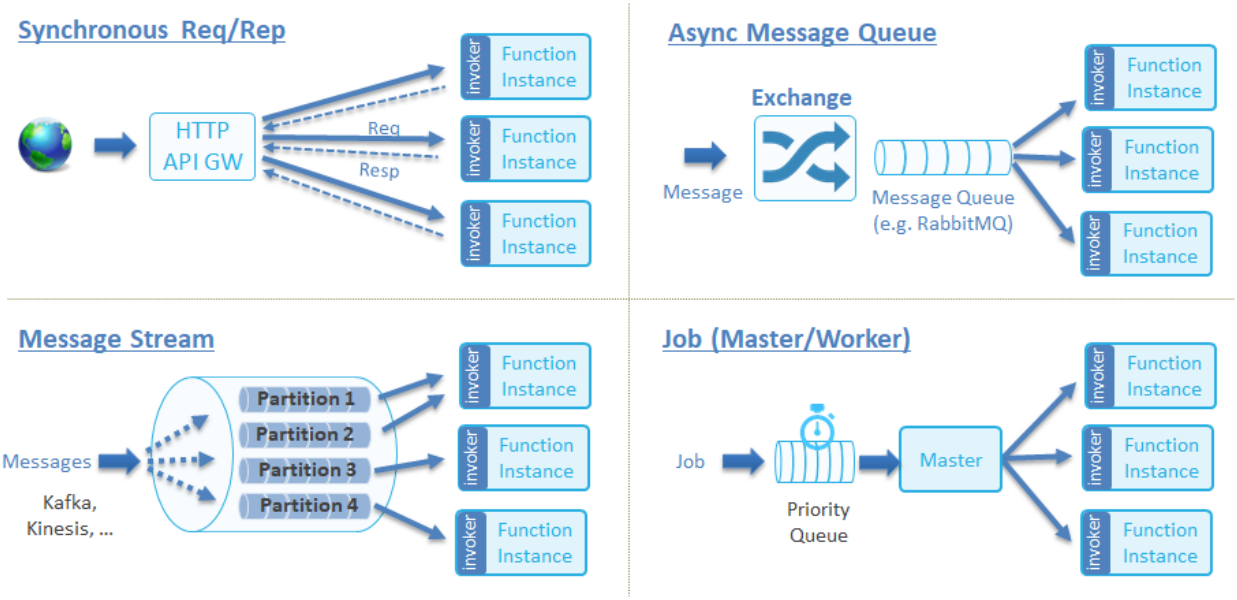


Figure 11 Event sources types (source: CNCF White Paper)

Functions can be implemented in any programming language among those supported by the platform. They can be organized in processing pipelines (depending on the platform) decoupling the orchestration of the dataflow between functions from the choice of language. (e.g., Kubeflow).

The lifecycle of a serverless function begins providing the function code and its specifications file (containing specifications and metadata). Code and specification are then compiled, and an artifact is generated (a code binary, package, or container image). Artifacts can be deployed on a cluster with a controller entity which oversees monitoring the functions and scaling the number of function instances based on the events traffic and/or load on the instances.

The following actions that define and control function lifecycle can be defined:

- Build – create a new function from its spec and code so that it can be deployed on the cluster
- Deploy
- Execute/Invoke - Invoke a specific function directly, instead of through its event source
- Event Source association - Connect a function with an event source
- Get - Returns the function metadata and spec
- Update - Modify the latest version of a function
- Delete - Deletes a function, could delete a specific version or the function with all its versions

3.1 Integration of the Serverless Platform in the Connect-compute platform

The integration of the Serverless Platform in the Connect-compute platform will consider the ETSI MEC reference platform for the Edge. The **FaaS platform** provisioning **Serverless Functions (SF)** can be deployed as a CNF as part of the Edge Host. SF are event-driven, stateless, cloud-native (Auto-Scaling, Design for Failure, Modularity, APIs, Automation) functions.

Deployment of the SF could be triggered by the **MECP Manager** using e.g., FaaS platform APIs.

SFs can be considered as **MEC Apps**. As such they:

- could be deployed by the MECP Manager (MECPM)
- can be exposed as **MEC Services** (when triggered by HTTP)
- can work "stand alone" as **MEC Apps but triggered by events**
- can consume **MEC Services (in event subscription modality)**

Multiple **Event Sources** can trigger the same SF. An SF may be set up to be triggered by an Event Source even if that is not present (in the same MEC host). An Event Source can be deployed as a MEC App on the MEC host.

The following figure shows various Serverless Platform elements in blue and light blue.

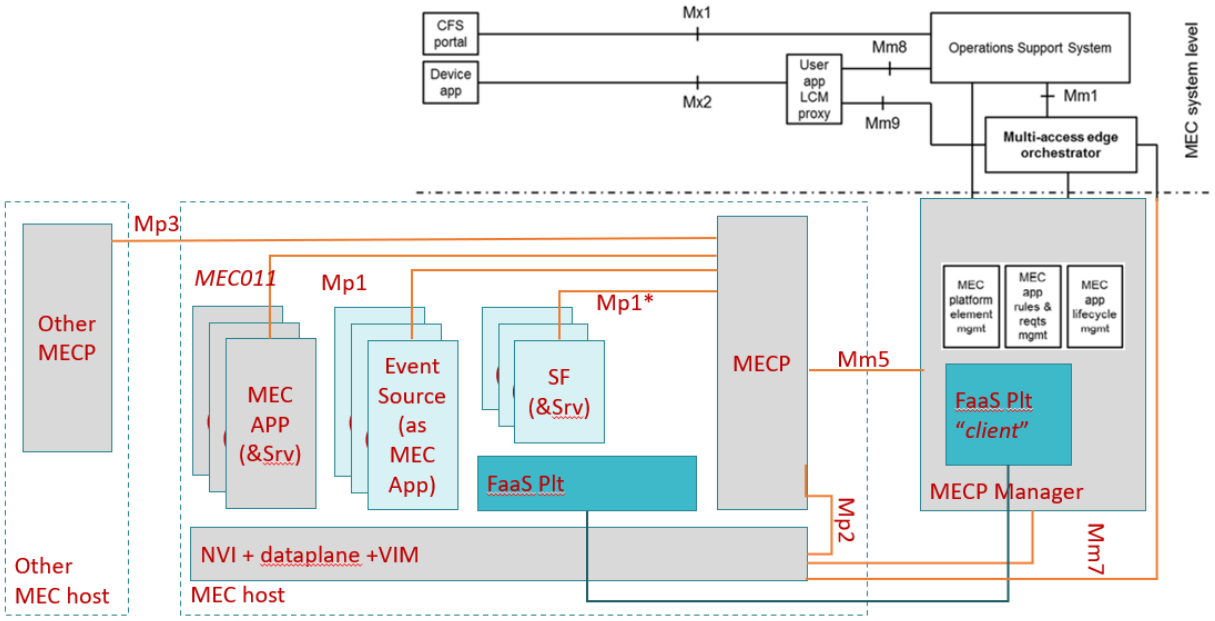


Figure 12 FaaS Platform Integration in ETSI MEC proposed by AI@EDGE

Building the function is the first step of the lifecycle of a serverless function, which starts by providing the function code and its specifications file (containing specifications and metadata). The code and specification can be compiled (in a code binary, package, or container image) and stored in a registry. This task can be performed leveraging on a Staging Platform, as shown in the figure below:

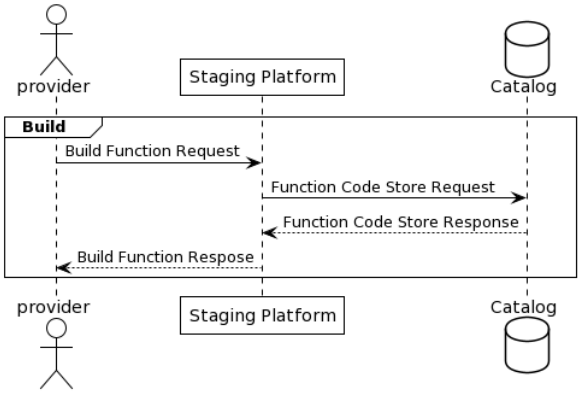


Figure 13 Example of Serverless Function build phase,
the Staging Platform can be an instance of the Serverless Platform used for building the function.

Then, the serverless function can be then deployed in the MEC host. The figure below shows how the deployment can leverage a MECPM add-on that is specific to the chosen FaaS Platform.

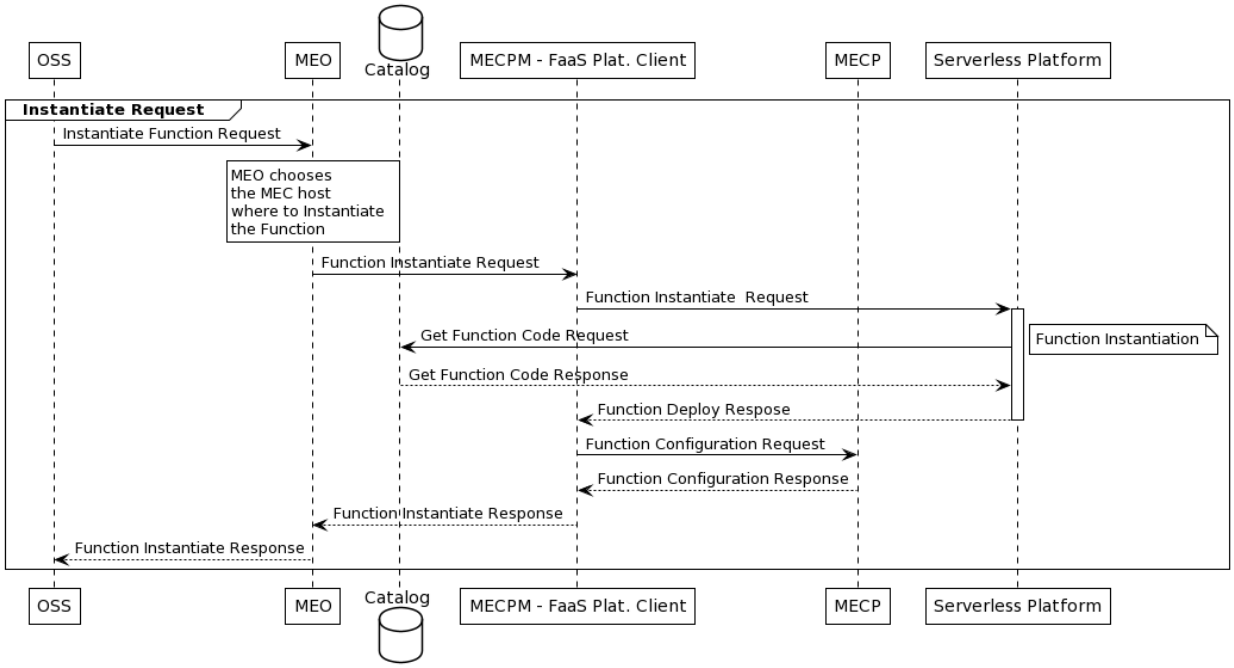


Figure 14 Serverless Function Deployment showing the interaction between the MECPM and the Serverless Platform

For the integration of the serverless platform in the connect compute platform, our design choice was to add a layer of abstraction from the actual serverless platform of choice and keep as much as possible isolated the specificities of the actual chosen serverless platform through a modular approach. As far as it regards the Serverless Platform selection, our choice was to use the Nuclio Serverless Platform [14] Community Edition. The main reason of this choice is for being a Lightweight Solution (with a portability across low-power devices, laptops, edge and on-prem clusters, and public clouds) with interesting real-time processing capabilities. Besides it natively integrates with a large variety of data sources (data bindings) and triggers,

and supports various programming languages, e.g., Python, Go, and Java. It provides REST APIs for deployment and allows to specify function specs in YAML file.

4. Disaggregated Radio Access

The O-RAN specification is an initiative promoted by the O-RAN Alliance which aims to standardize the architecture and procedures in virtualized RAN environments, focusing on AI-powered RAN control to fulfill SLA. In this sense, the AI@EDGE concept of closed-loop automations naturally fits within the O-RAN paradigm, where the NSAP implements a subset of functionalities of the Service Management and Orchestration (SMO) layer and the Connect-Compute platform hosts other O-RAN functions and NFs like the Near-RT RIC, the CUs and the DUs, as is done in the O-Cloud platform. In this Section we will introduce the architecture and functionalities of the RAN Intelligent Controllers present in the O-RAN architecture, the non-Real Time and the near Real Time, with focus on the methods that enable intelligent network automation.

4.1 Provisioning of Integration Testbed

The non-Real Time RIC (nonRT-RIC) is a core element of the SMO layer which enables non-real time control and optimization of RAN components and resources. Figure 15 shows the service-based view of the non-RT RIC as described in the O-RAN specification [15]. As shown in Figure 15, although the main functionality is to realize the A1 interface termination, the non-RT RIC also exposes different SMO functions to the rAPPs (e.g. O1 and O2 interfaces), which are applications that run on the non-RT RIC and implement the intelligent operations. In AI@EDGE we will focus on A1-P and AI-EI interfaces, i.e. Policy management and Enrichment Information (EI) functionalities, which provide the needed subset of mechanisms to manage closed-loop automations at the near-RT RIC using xAPPs. Nevertheless, other SMO functionalities, like xAPPs onboarding, data and data exposure could be partially developed and demonstrated during the project if available in the different envisioned realizations.

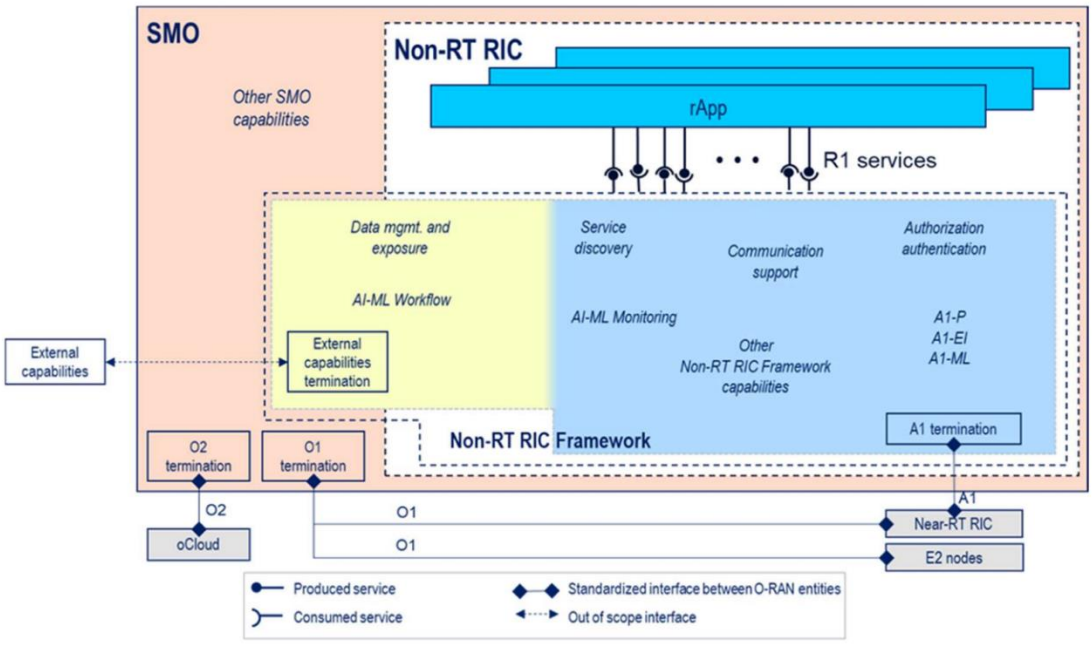


Figure 15 O-RAN's Non-Real Time RIC architecture service-based view [O-RAN.WG2.Non-RT-RIC-ARCH-TR-v01.01]

Figure 16 shows the non-RT RIC architecture being developed. As aforementioned, the focus will be on implementing the components and interfaces that are specific from the non-RT RIC, i.e., the Policy

Management and Enrichment Information Coordinator and the A1-P and A1-EI, respectively. To facilitate the integration with different near-RT RIC implementations, i.e., O-RAN compliant but also O-RAN non-compliant, it is envisioned that the A1-P and A1-EI adaptors will do the needed adaptations to vendor-specific implementations and interfaces (denoted as A1-P* and A1-EI* in the figure). As shown in Figure 16, Policy- and EI-related functions will be exposed to rAPPs via the R1* interface (i.e., an adaptation or implementation of a subset of functionalities of O-RAN's R1 interface). The Policy Management component of the non-RT RIC manages the life cycle of the policies, each of them comprehending one scope and multiple statements, which are defined as follows [16]:

- The scope of the policy defines what the statements are to be applied on (i.e., UE, groups of UEs, slices, QoS flows or cells).
- The statement defines the objective or goal of a policy (e.g. QoS or QoE targets) and how to use the RAN resources to achieve it.

Regarding Enrichment information, the EI coordinator will create EI jobs to serve the data available at the SMO level according to rAPPs and near-RT RIC/xAPPs demands. This data might include aggregated RAN data obtained from near-RT RIC and E2 nodes, but also external data obtained from applications or other network elements. Finally, the Figure shows how the rAPPs might use other SMO functions, like the O1 or O2 interfaces (e.g. to onboard xAPPs or deploy AI/ML models).

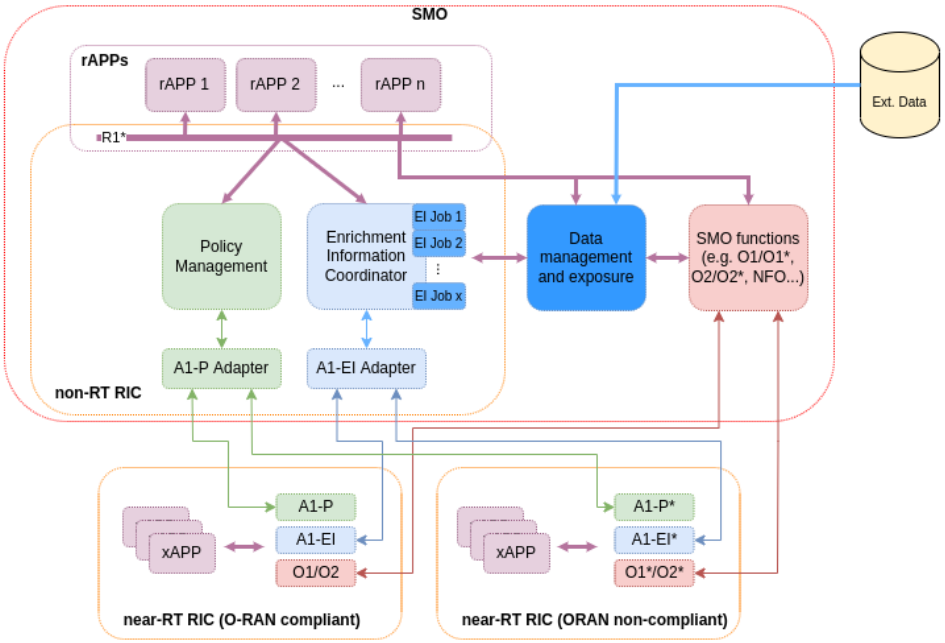


Figure 16 AI@EDGE Non-Real Time RIC architecture and interfaces to near-RT RIC

4.1.1 Roadmap

The development roadmap is divided into two phases, (i) implementation of the Policy Management and A1-P interface and (ii) implementation of the EI Coordinator and the A1-EI interface. The implementation will be based on O-RAN specification and validated towards an O-RAN compliant near-RT RIC (e.g. the O-RAN simulator available in the O-RAN Software Community [17]).

4.1.2 Integration

Apart from the validation via O-RAN's near-RT RIC simulator, we will work on the integration with two near-RT RICs from different vendors. The first one, which is a commercial one called dRAX and developed by Accelleran [18], will be used to validate different non-RT RIC and SMO features for enabling closed-loop automations using simple rAPPs and xAPPs created for this purpose. In parallel, we will work on the integration with 5G-EmPOWER, which is the open-source near-RT RIC selected for the AI@EDGE project. In this case, the scope will be to demonstrate a subset of A1 functionalities to manage xAPPs related to the experiments and testbeds of the project.

4.2 Near-Real Time RIC

The Near Real Time RAN Intelligent Controller (near-RT-RIC) is a logical function pioneered by O-RAN Alliance to enable RAN programmability and service optimization. With an open architecture, near real time RIC allows on-boarding of RAN control applications for near-real time fine-grain performance optimization and policy tuning. ML-based algorithms are implemented as external applications, i.e., xApps, deployed on the near real-time RIC. They can deliver specific services such as inference, classification, and prediction pipelines to optimize the per-user quality of experience, controlling load balancing and handover processes, or the scheduling and beamforming design [19]. In addition to the O-RAN near-RT-RIC reference Open-source implementation, many others exists. For example, the FlexRIC [20] RAN Intelligent Controller, which interfaces with the OAI radio software stack with the O-RAN defined E2 interface to monitor and control the RAN in real time. FlexRIC's built in service models can easily be customized to support diverse 5G use cases. In the scope of this project 5G-EmPOWER will be used as near-Real Time RIC in the AI@EDGE platform. The detailed description and supported features will be provided in the following sections.

4.2.1 5G Empower

5G-EmPOWER is a lightweight state of the art RAN Intelligent Controller. Its resilient architecture enables the agile development of innovative services across multi-tenancy radio equipment. A high-level view of 5G-EmPOWER is depicted in Figure 17. The separation of control and data planes in 5G-EmPOWER is achieved via two main components i.e., a centralized controller and a set of agents. The centralized controller acts as 5G-EmPOWER Operating System (OS) with a global network view of all the underlying infrastructure and its functionalities. 5G-EmPOWER OS sends the control directives to 5G-EmPOWER agents via OpenEmpower communication protocols. 5G-EmPOWER agent manages the LTE user plane. It consists of the platform independent 5G-EmPOWER agent itself and the platform dependent Wrapper. In addition, 5G-EmPOWER provides an SDK environment for network application developers to write their own applications and services as empowerApps.

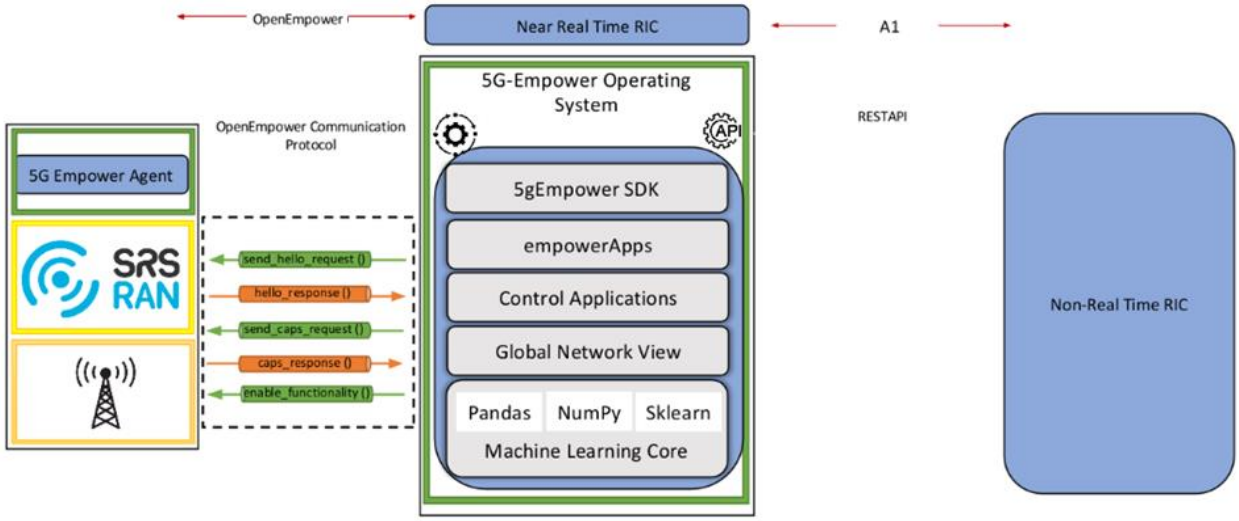


Figure 17 High-level view of the 5G-EmPOWER architecture.

4.2.2 Roadmap

We plan to carry out the integration of the RAN Controller in two Phases, i) In the first phase which corresponds to Phase 2 timelines mentioned in section 2.4, the 5G-EmPOWER interface with the RAN will be used to support monitoring capabilities and metrics collection. ii). The Non-real time RIC policy translation will be integrated in 5G-EmPOWER. The timeline to add this feature in the testbed fully integrated and tested corresponds to phase 4 mentioned in section 2.4.

4.2.3 Integration

The integration of 5G-EmPOWER in the connect and compute platform testbed will be carried out in two stages. In the first stage, RAN monitoring capabilities will be demonstrated over 4G/5G- NSA RAN elements in line with the project phase 2 timelines and goals outlined in section 2.4. In this stage, the communication between the RAN Controller and the RAN will happen through the OpenEmpower protocol. In the second stage, which corresponds to phase 3 and phase 4 timelines and goals, 5G-EmPOWER integration to non-real time RIC interface will be established. 5G-EmPOWER RAN monitoring capabilities will be extended to 5G-SA gNBs, but it depends upon the gNB ability to communicate with 5G-EmPOWER Operating System.

In addition to above mentioned points, the integration of the connect and compute platform with Open-Source Near-Real time RIC projects (e.g., O-RAN Near Real Time RIC, FlexRIC) will be studied in parallel. However this option depends upon the availability of their respective stable releases and Inter-Operability with the Connect and Compute Platform in the next future.

4.3 Subset of functionalities of A1 Interface of relevant AI@EDGE

The non-RT RIC uses A1 interface to manage near-RT RIC policies and to provide enrichment information [21]. These functionalities have a direct impact on the optimizations performed by the xAPPs. Table 1and

Table 2 describe the minimum subset of A1 procedures needed to implement them, while Table 3 details some additional/optional O-RAN procedures that could be also involved.

Table 1 Subset of A1-P procedures needed for enabling policy management

Procedure	Producer	Description
Query Policy Types	Non-RT RIC	Queries the policy types present at the near-RT RIC
Query Policy Type	Non-RT RIC	Queries the description of a specific policy type
Create Policy	Non-RT RIC	Creates a policy in the near-RT RIC related to a specific policy type/xAPP
Delete Policy	Non-RT RIC	Deletes a policy in the near-RT RIC related to a specific policy type/xAPP
Update policy	Non-RT RIC	Updates a policy in the near-RT RIC related to a specific policy type/xAPP

Table 2 Subset of A1-EI procedures needed for providing enrichment information

Procedure	Producer	Description
Query EI Types	Near-RT RIC	Queries the available EI types present at the non-RT RIC
Query EI Types	Near-RT RIC	Queries the description of a specific EI type
Create EI job	Near-RT RIC	Creates a policy in the near-RT RIC related to a specific policy type/xAPP
Deliver EI job result	Non-RT RIC	Delivers the result of a EI job to the non-RT RIC

Table 3 Additional/Optional O-RAN procedures involved in policy/EI management

Procedure	Producer	Description
xAPP onboarding	SMO	Deploys a specific xAPP in the near-RT RIC (O2 interface)
Create Policy Type	SMO	Creates a specific policy type in the near-RT RIC/xAPP (O1 interface)
RAN data delivery	Near-RT RIC, E2 nodes	Delivers RAN data to the SMO (O1 interface)
External data delivery	Ext. component	Delivers external data to the SMO (interface not defined)

More details on the workflows between non-RT RIC and near-RT RIC related to the A1 interface can be found in D2.2.

4.4 RAN Controller – RAN interface functionalities

In 5G-Empower, OpenEmpower Communication protocol allows remote management of RAN elements. The OpenEmpower protocol operates in a way analogous to the E2 Interface to control RAN elements. 5G-Empower communication protocol is built around three event types, described below

- **Single Events:** Standalone events requested by Operating Systems and notified back immediately by the agent
- **Scheduled Events:** Initiated by the Operating System and then executed by the agent periodically.
- **Triggered Events**: Instructions to activate/deactivate a control plane functionality based on parametric changes at the agent.

More details with elaborated descriptions for reference are defined in Table 4.

Table 4 Actions Supported by 5G-EmPOWER Communication Protocol [22]

Operation	Event Type	Description
Hello	Single	Periodic heartbeat message sent by the eNB to the operating system

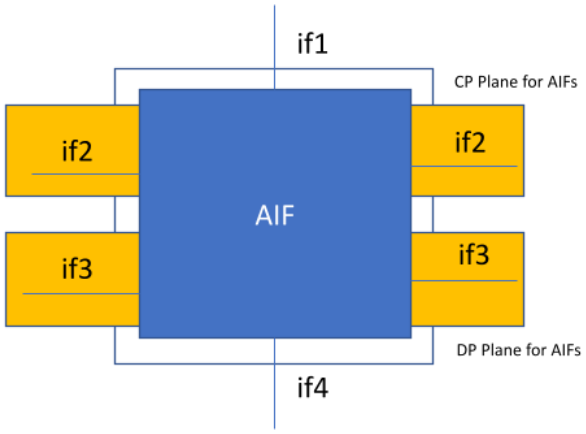
Handover	Single	Triggers an X2 handover. The message specifies the UE RNTI and the target eNB/Cell.
MAC Reports	Scheduled/Single	Collects the PRB utilization statistics from the MAC scheduler (uplink/downlink)
UE Reports	Triggered	Triggers a message when UEs attach/detach from an eNB
RRC Measurements	Triggered	Instucturs a UE to start RSRP/RSRQ measurements on one or more channels and with certain internal

5. Data-driven Service Lifecycle for AI-enabled Applications

5.1 Introduction

The AI@EDGE system supports the orchestration and management of Artificial Intelligence Functions (AIFs). The AIFs represent an abstraction of software modules that implement Machine Learning (ML) models and expose various interfaces dedicated to (i) the input/output of data; and (ii) the AIF configuration. The AIFs implement the intelligence of the system and can support both network automation and user applications at the edge. AIFs are further described in D2.1 and in D3.1, where NSAP AIFs are described. Additionally, each UC will produce its own AIFs that will be specific for each scenario. These are introduced in D5.1.

In D3.1, a Reference Model for AIFs has been introduced (Figure 26), representing the concepts, functionalities, and interfaces for a generic AIF. The AIF Reference Model intends to provide a consolidated view of all information elements present as part of the interface specifications. This model is a tool to check consistency between information elements as well as to provide a logical relationship between information elements across different interfaces. Further details on the AIF reference model and its interfaces can be found in D2.1 and D3.1.



AIF's Interfaces:

- *if1: (re)configuration*
- *if2: model parameters exchange*
- *if3: data exchange*
- *if4: reconfiguration of other entities*

Figure 18 AIF Reference model [D2.1, D3.1]

5.2 End-to-end Decentralized and Distributed Orchestration

In D2.2, we described the AIFs orchestration part workflow from a functional point of view, while here, we will describe all the phases and methods used to support these features more accurately.

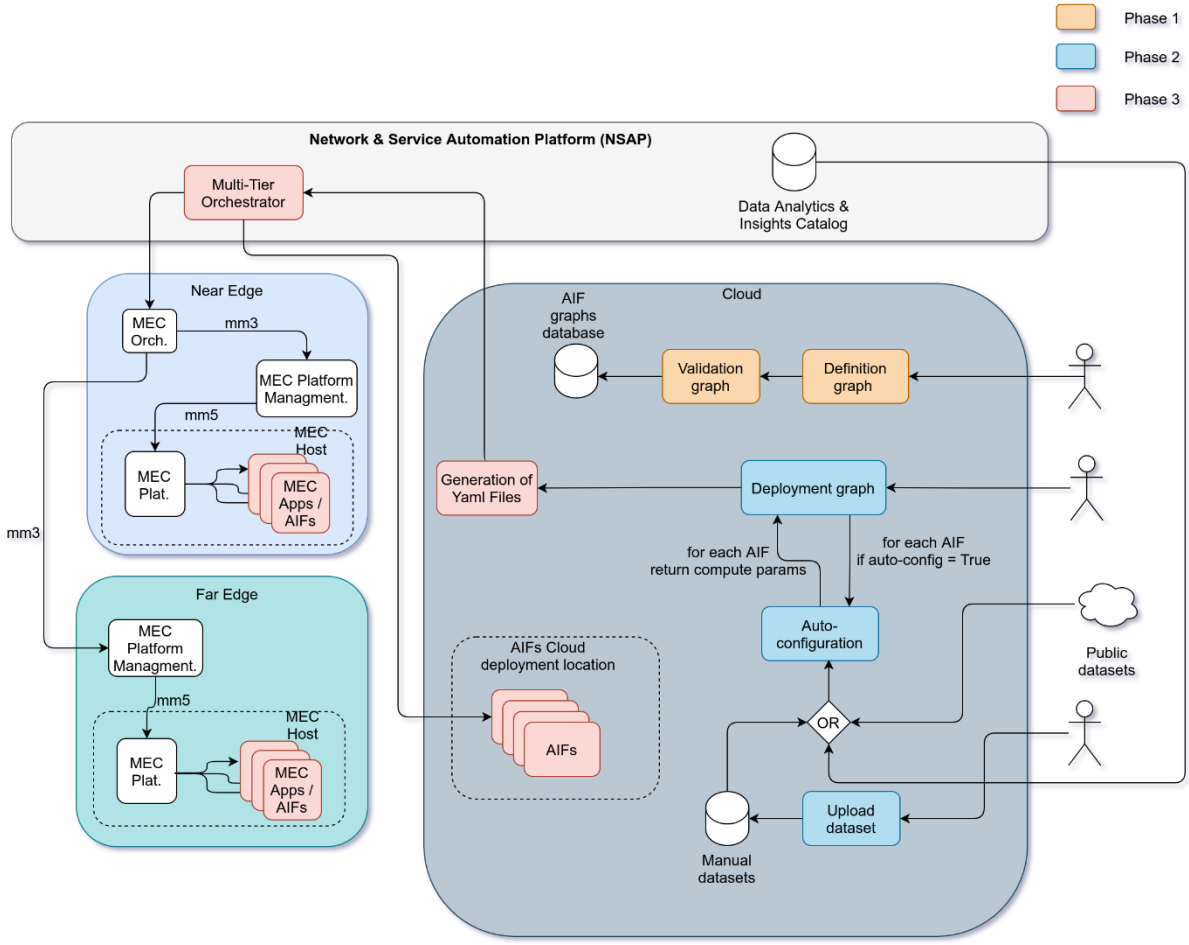


Figure 19 AIF lifecycle

As a reminder of D2.2, this is the AIF lifecycle. There are three main phases. In the first one, an AIF graph can be defined and validated. The next one is when the willing to deploy a chosen AIF graph from those already defined and validated and perform the AIFs parameters auto-configuration when the "auto-config" parameter has been set to "True". And the final one is the generation of the descriptor files and the whole deployment part from the multi-tier Orchestrator after receiving the files.

During the validation stage, at a high level it is a question of validating both the semantics of the AIF graph built upstream using the user's inputs and potentially ensuring the cleanliness of the associated data for the construction of the AIFs deployed at the end on the connect-compute platform.

Then, once the graph is chosen, each AIF must be configured with some parameters (or hyper-parameters in the case of ML functions) in order to have the best possible results.

Moreover, once the parameters are settled and saved, we will create all the files we need to be able to deploy the complete application into the orchestrator cluster. For that, we will use the tool named Helm to generate all the files and then use them to deploy the application.

This section will talk about what is concretely an AIF graph and then the 3 main phases.

5.2.1 AIF graph

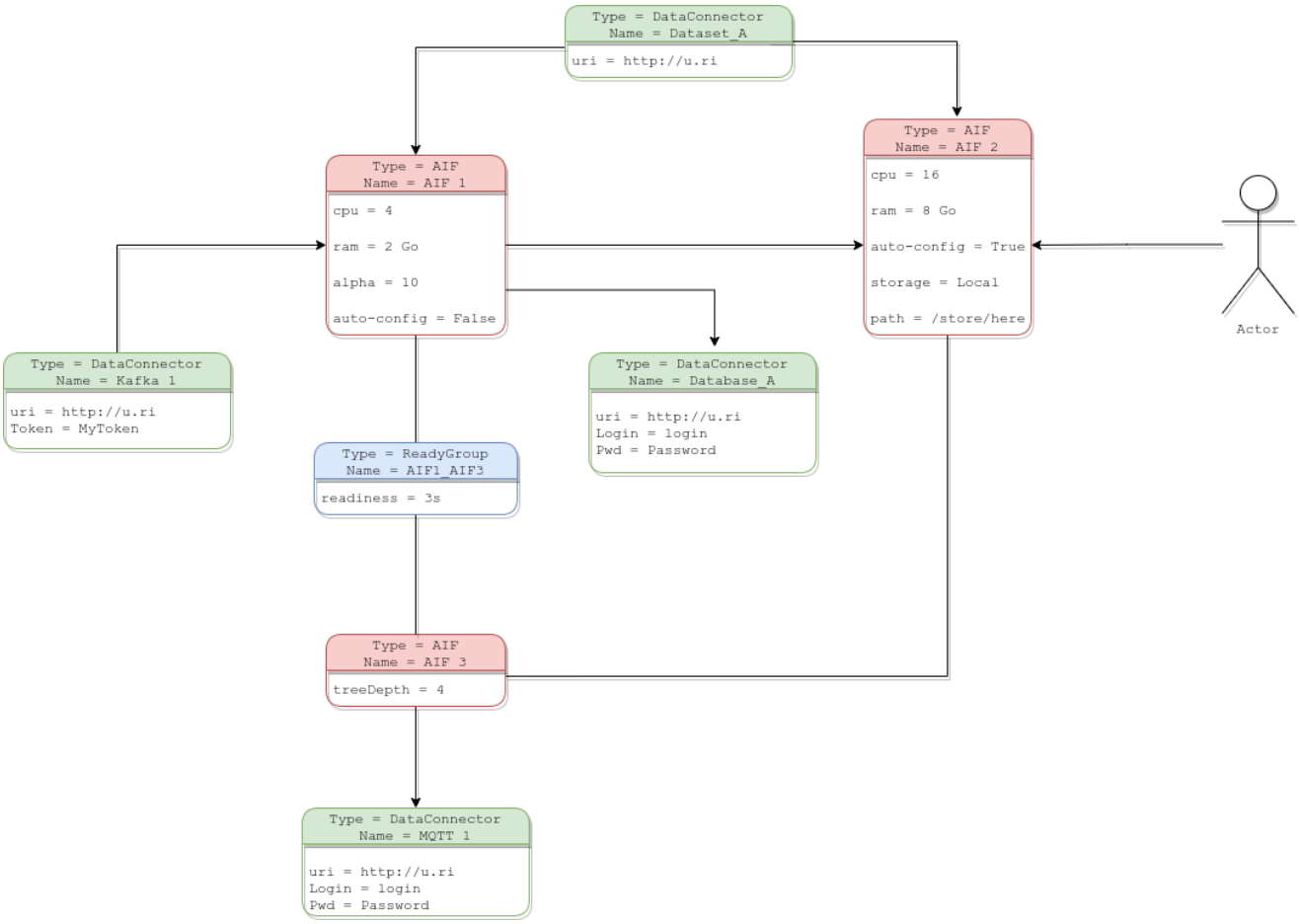


Figure 20 Example of AIF Graph

Because the system needs to know what an input or output is, we need to have all the information in one file. To have a better view, we can represent this file as a graph. The graph in Figure 21 is an example of how to represent this information. It shows several objects: coloured nodes, actors, oriented links and non-oriented links. This is a non-exhaustive list of elements that can compose the graph. Let us take a deeper look into these elements.

First, we have the colored nodes. These represents the main part of the graph: the AIF and their interactions. Each colored node has at least: a **type**, that will define the type and the allowed properties of the node and a **name**, that will distinguish the node from the others.

There are several types of nodes:

- **AIF**: a node of this type will define an AIF. It can have several properties which will override all the default options of the AIF we can find into the AIF graph catalog. An AIF can be connected by an oriented link with a data-connector or another AIF to know if it's an input or an output. It can also have be connected by a non-oriented link with one (or more) ReadyGroup node because it is not relevant to it.

- **ReadyGroup**: at least two nodes (AIF or DataConnector) need to be wired to this type of node. It must have at least the property "Readiness" with a numerical value in seconds. This will add a condition that all the nodes linked to that node have to be ready as one under the readiness value.
- **DataConnector**: this type of node will simply define a connection to an external data connector. It can be a dataset, a database, a message queue, etc. It always has to be linked at an AIF node and be connected by oriented links as an input or an output.

An actor can at any moment modify an AIF Graph. An actor can be another AIF who decide itself to modify the graph for some needs or a human. Further in the project, we can have more node types and actors.

5.2.2 AIF lifecycle phases

Phase 1: AIF graph and data validation

Technically, the AIF graph will be a structured file (XML, JSON, YAML, ...). In the first iteration, we will do a simple parsing validation. Each type of node has a fixed list of properties and a type of value associated with that property. If a property is not unknown or an associated value is not the right type, an error will be raised.

Also, nodes may have properties that include links. These may point to a local access folder, so in this case the validation process has to verify if the application has the rights to access that specific folder. The link can also be an URI, in which case the URI will be tested, and the validation process will try to download the file (in the case of a dataset) or access it with the given token or credentials (in the case of a database or message queue).

Once the whole graph is validated, it will be stored in a dedicated database from which an actor will be able to fetch them all.

Phase 2: AIFs deployment and configuration

For this phase, we need to have at least one graph stored in the dedicated database for AIF graphs. An actor can select any graph to deploy and launch the deployment process. For each AIF node, if it has the property "auto-config" equal to "True", it will pass through the parameter auto-configuration process.

Parameters auto-configuration process:

The performance of many algorithms in the field of machine learning or AI, in general, depends on tuned hyperparameter configurations [23]. Especially recent deep neural networks crucially depend on a wide range of hyperparameter choices about the neural network's architecture, regularization, and optimization [24]. Many research directions show a tendency towards increasingly complex algorithms with more and more hyperparameters [25]. Thus, the optimization of algorithm hyper-parameters (HPO) is crucial for achieving peak performance.

This part proposes a cheap approach to infer the best (hyper-)parameters configuration of an AIF. This solution is based on Meta-learning. The aim is to learn the relationship between task's meta-features and their optimal configurations by building a meta-model that recommends the configurations of a new task given its meta-features. The remainder of this part is structured as follows. First, we discuss background on (hyper-)parameter optimization methods paying particular attention to Blackbox model-based methods and meta-learning. The last paragraph will focus on our method to perform (hyper-)parameters optimization.

In general, every Blackbox optimization method can be applied to HPO. Blackbox HPO methods can be divided into: model-free and model-based methods. The model-free standard for hyperparameter optimization in machine learning is Grid Search (GS). GS is a basic automated method for hyperparameter

optimization (HPO). It is based on a brute force method that evaluates all the hyperparameter combinations given to the grid configurations. Hyperparameters are one of the inputs to experiment with, and the values to try. Users must have some preliminary knowledge of these hyperparameters because they are who generate all candidates. GS can be easily implemented and parallelized. However, the consumption of computational resources increases exponentially when more hyperparameters are awaiting tuning. Therefore, this solution is inefficient for high dimensionality hyperparameter configuration space. A more sophisticated approach which is also a model-free method is Random Search (RS). RS is an improvement on GS. Instead of trying out all possible combinations, RS randomly selects a pre-defined number of samples between the upper and lower bounds. The searching process continues till the predetermined budget is exhausted. RS is more efficient than GS for large search spaces, but it is still a computationally intensive method. Both GS and RS are implemented in the scikit-learn Python open-source machine learning library.

Bayesian optimization (BO) has emerged as a powerful Blackbox model-based solution for this kind of problem. It is a sequential model-based method that determines the future evaluation points based on the previously obtained results. BO uses two key components: a surrogate model which is a probabilistic model of the objective function, and an acquisition function that aims to detect the optimal hyperparameter values on the surrogate model. BO balances exploration and exploitation to avoid trapping into the local optimum. This trade-off can be controlled via the acquisition function.

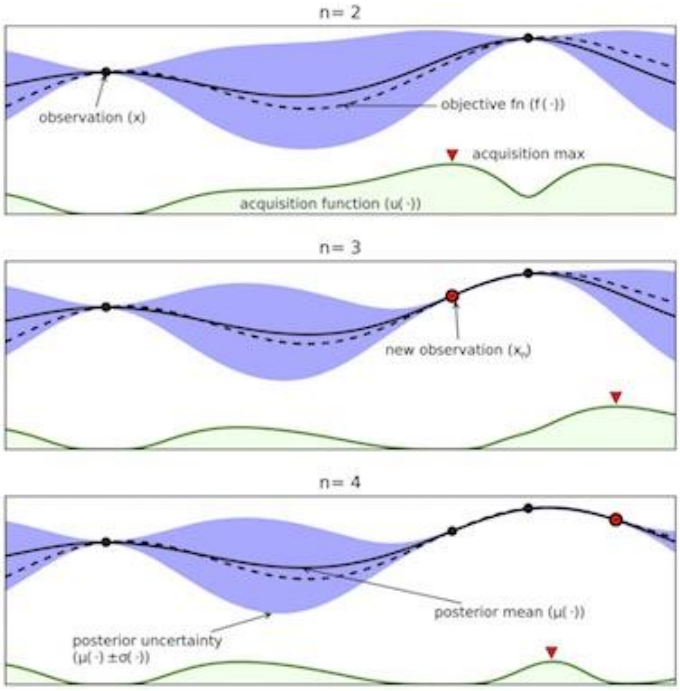


Figure 21 Bayesian Optimization model

The three most popular implementations of Bayesian optimization are Spearmint [26], which uses a Gaussian process (GP) [27] model as the surrogate model; SMAC [28], which uses random forests [29] modified to yield an uncertainty estimate [30]; and the Tree Parzen Estimator (TPE) [31] implemented in the Hyperopt Python library, which constructs a density estimate over good and bad instantiations of each hyperparameter to build the surrogate model.

Despite the effectiveness of the methods listed below, the Blackbox view is too slow for deep learning and big datasets. Indeed, if doing a single function evaluation takes a week then getting 50 samples would take a year. In order to overcome this concern, several beyond Blackbox approaches were proposed [32]. In the following paragraphs, we will introduce two of them: multi-fidelity optimization and meta-learning. We considered the second one to implement our solution.

Multi-fidelity methods enable the use of HPO even for costly models, by exploiting approximate performance measures that are cheaper than full model evaluations. The cheap approximation has to behave in the same way as the original Blackbox function. If a hyperparameter setting does well in the cheap approximation, it typically does well on the expensive Blackbox. To get these cheap approximations, we can proceed in different ways, the most common and efficient one is to use subsets of the data or more generally subsets of a defined budget (time for example). Successive Halving is a simple approach that consists in just sampling a number of random configurations on the cheapest fidelity (for example a small subset of the data) and take the best fraction to move them to the next budget (a bigger subset of the data) and so on until the original Blackbox function is used. While successive halving is an efficient approach, it suffers from the budget-vs-number of configurations trade-off. HyperBand (HB) [33], which is an extension of Successive Halving, fixes this issue by calling the latter as a subroutine on each set of random configurations that are created by dividing the total budget into several combinations of number of configurations vs. budget for each. Since HyperBand is based on random search, it does not exploit knowledge about which hyperparameter settings work well and where Bayesian optimization is strong. BOHB [34] combines Bayesian optimization and HypeBand to achieve the best of both worlds: BO for choosing the configuration to evaluate and HB to decide how to allocate the budgets.

To summarize what to use in which situation based on the available open-source implementation:

If multiple fidelities are applicable (i.e., if it is possible to define substantially cheaper versions of the objective function of interest) BOHB is recommended as a robust, efficient, versatile hyperparameter optimization method. If multiple fidelities are not applicable, then:

- If all hyperparameters are real-valued and one can only afford a few dozen function evaluations, a Gaussian process-based Bayesian optimization tool may be suitable, such as Spearmint.
- For large and conditional configuration spaces, both the random forest-based SMAC or TPE proven to have strong performance on such tasks.

The second way of going beyond the Blackbox function is to use meta-learning [35]. Our solution considers this approach to perform hyperparameter optimization.

As humans we never run into the situation when we start a task from scratch, we always have prior experience that we can use to solve the task more efficiently than when we would have to start from scratch.

Learning is a never-ending process, every time we encounter a new task, we learn how to do it efficiently based on prior experience. This process is called meta-learning, we learn more efficiently with less trial and error and fewer data. The main idea is to transfer an inductive bias from prior learning iterations to the new task. An inductive bias is any assumptions or priors added into a learning system of the new task except for the training data. If we extract useful information like constraints, beliefs, or representations from previous tasks, the new task becomes much more manageable. This dramatically speeds up and improves the design of machine learning pipelines and replaces hand-engineered algorithms with methods learned in a data-driven way.

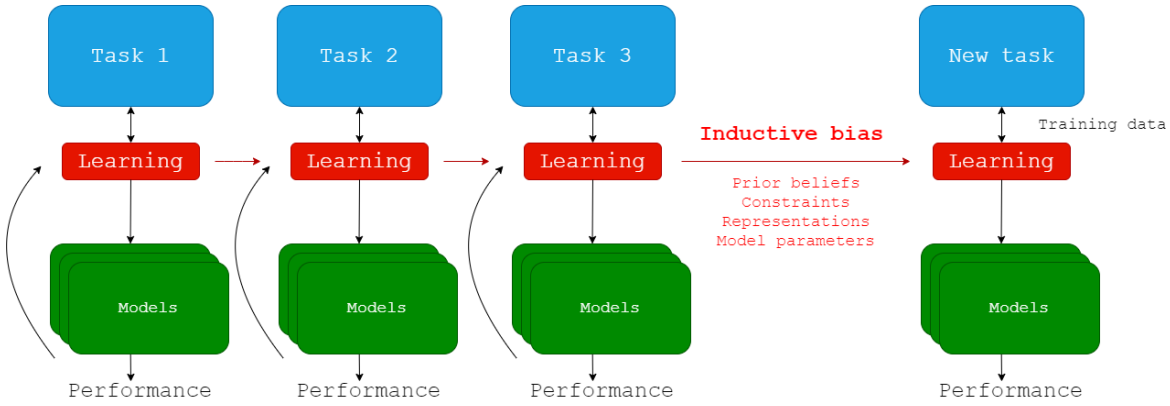


Figure XX

Figure 22 Bayesian Optimization model

The underlying part is that the prior tasks must be similar; if not, it may harm the learning. The collected data is called meta-data, which is data about prior learning episodes, and we transfer that to the meta-learner. The meta-data describes prior tasks and previously learned models. The meta-learner gets a bunch of meta-data and has to make sense of that and use it in a useful way to construct a base learner. Then, we can do the actual modelling. In some cases, the meta-learner and the base-learner are squashed together, and the meta-learner will directly build models.

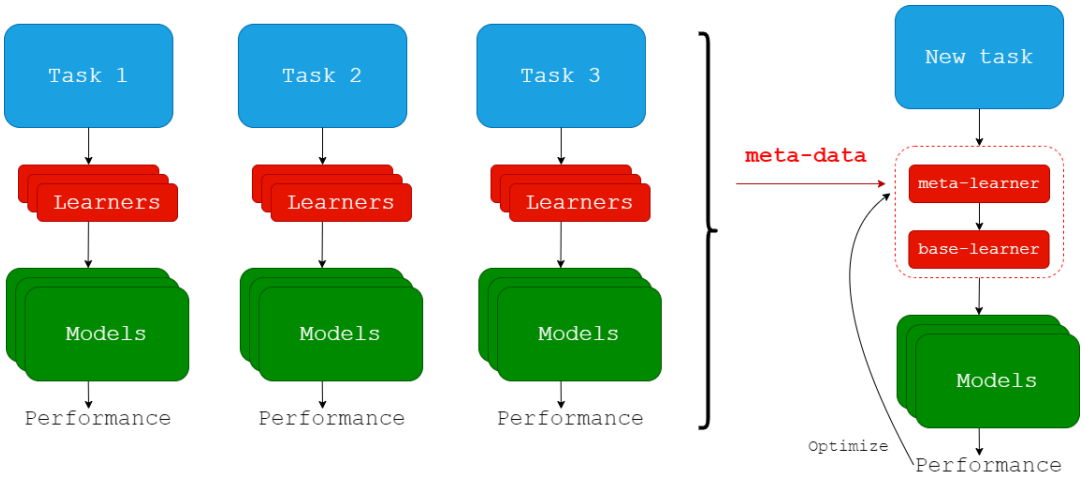


Figure 23 Meta-learning tasks

Meta-learning can be subdivided into three levels, each one requiring more and more similar tasks. The first type of problem is where the tasks can be very different from each other, and then we just generalize general knowledge about tasks. Typically, what humans do when they are confronted with tasks that they are not familiar with, they just try whatever worked well in the past. The second type is when we have more information about the tasks. We can characterize tasks by extracting meta-features to more explicitly express task similarity and thereafter compare a new task to prior tasks. Meta-features, also called characterization measures, are able to characterize the complexity of datasets and provide estimates of algorithm performance and a list of well-performing configurations. This method allows us to reason about

the difference between a new task and previous tasks in order to transfer information in a much more useful way.

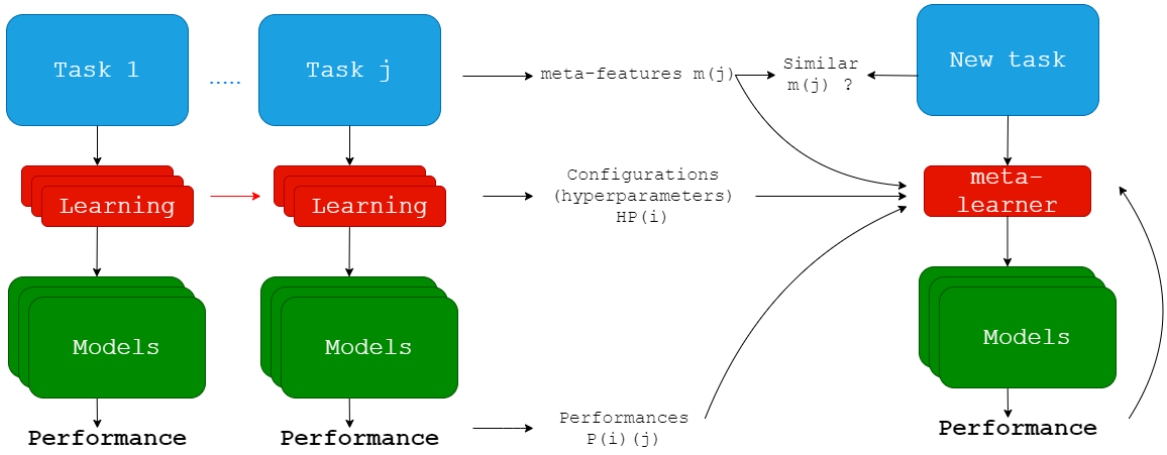


Figure 24 Meta-learning levels

In the last type we come to tasks that are so similar that it is possible to take a train model from a prior task and then repurpose it for solving a new task. Our contribution is based on the second type of meta-learning.

By using the evaluations and meta-features of previous tasks, the method aims to train a meta-learner to build a meta-model that tries to predict a set of optimal hyperparameter configurations for a new task. The meta-features used are computed on single features, or combinations of features. The main meta-features extracted are general meta-features which consist of simple measures of the dataset (such as the total number of attributes and the number of binary attributes) and statistical meta-features which are measures that capture statistical properties of the dataset (such as average, standard deviation, correlation and kurtosis). These two groups represent the most common and traditional approaches to data characterization [36]. Other characterization measures are described in the literature [37]. Before computing meta-features, a feature selection is performed on each task.

Figure 25 Meta-learner

Phase 3: Generation of descriptor files and orchestration

Once we have all the configurations we need (those in the AIF catalogue and the graph and those computed by the auto-configuration process), the last thing to do is generate the descriptions files for the multi-tier orchestrator.

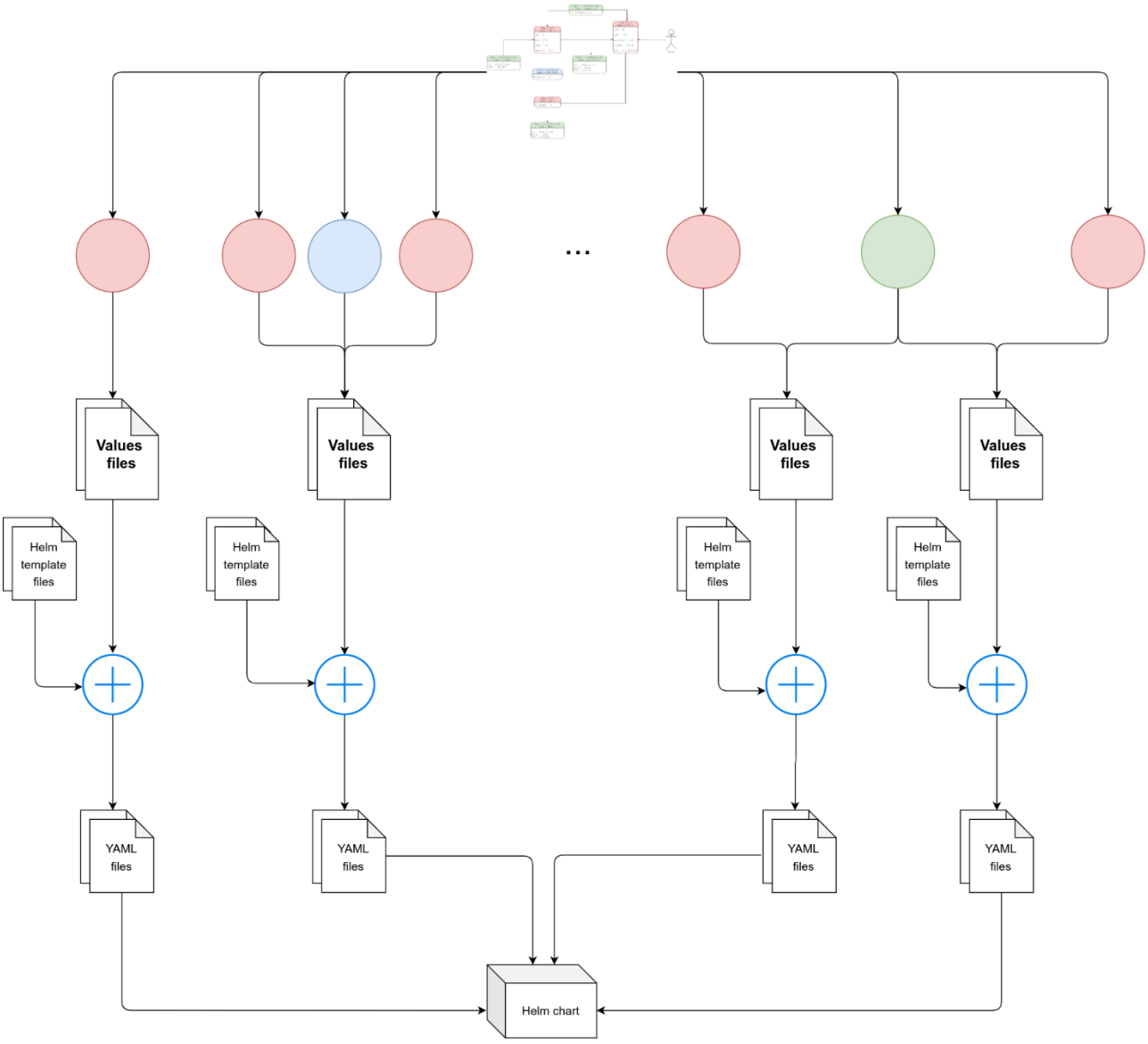


Figure 26 Helm chart overview

To do that, the AIF graph (as a structured file) will be parsed to extract several components or groups. Each will be transformed into one or several values files (deployment file, secret for credentials, configMap for configuration, services, etc.). Once everything has been transformed, we can generate the correct and useful YAML files by combining the values files from the structured file and the auto-config result for each AIF node with the associated template. In the end, we can gather all the YAML files into a single chart.

One of the main challenges of deploying AIFs at an edge node is the lack of a large pool of resources, high cost of resources, and heterogeneity of resources, to name a few. Additionally, the AIF graph may impose some constraints such as the availability and type of HW acceleration, constraints on co-location of AIFs, latency constrains or the vicinity with the source of data. For this reason, resource availability check and allocation is a critical task for the deployment of the AIFs.

A sizable body of research has been conducted on devising new approaches on service orchestration and resource allocation to achieve performant solutions for the embedding of services into the network and at the same time efficiently utilize edge resources. The works presented in [38, 39, 40, 41, 42, 43] are examples of studies carried out on service orchestration and resource allocation at the network edge. The main drawback of the listed approaches is that although they reach an optimal solution to the problem, they are not scalable and cannot be considered as practical solutions in real-world scenarios. To meet 5G and B5G requirements, service placement and resource allocation at the edge cannot rely on static algorithms but should consider a continuous optimization of the network.

To support the orchestration of the AIFs, we introduce a resource allocation module responsible of implementing such algorithms. This module's task is to assist the MEO in orchestrating both the AIFs and non-AIFs components needed for composing the application or service. This module relies on the information present on the automatically generated YAML file described before, which contains all the information for the orchestration of the elements contained into the AIFs graph and on the input from the MEC Orchestrator regarding the available resources.

Based on the output of the resource allocation module, the MEO will be able to choose the most appropriate MEC hosts where to deploy the various AIFs components and to allocate the appropriate resources.

In the next stages of the project, we will focus on the study of advanced approaches for service orchestration and resource allocation at the edge targeting to achieve a shorter time and at the same time guarantee a certain level of performance. We will also study machine learning solutions that use operational data in the network for network management decisions, fully compliant with the zero-touch approach to network management.

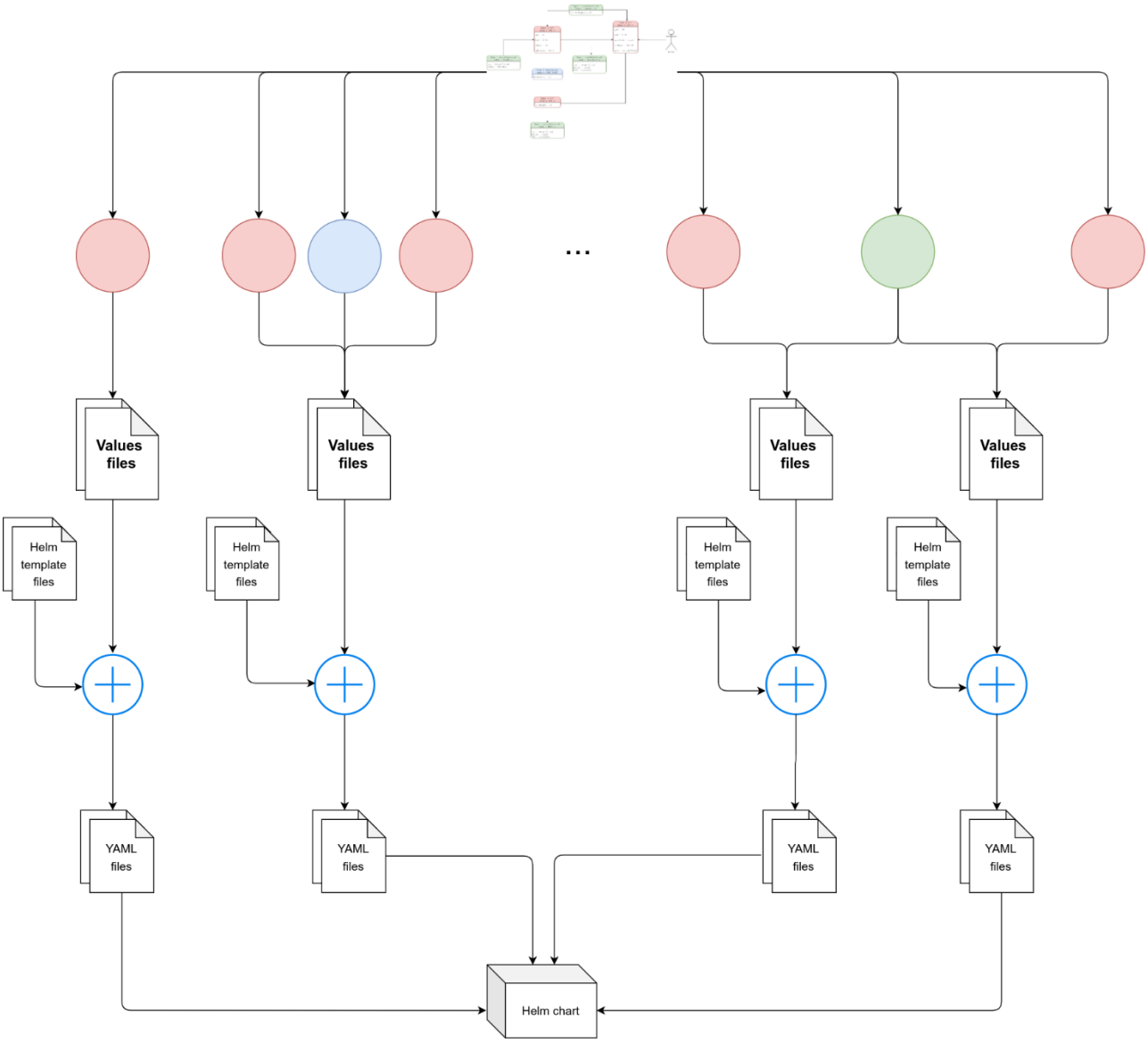


Figure 27 Helm chart overview

Another aspect that is crucial in edge computing is monitoring the deployed AIFs or VNFs. As a first step, metrics from the deployed AIFs and the VNFs need to be collected and stored in a timeseries DB (e.g., Prometheus) to provide temporal information over the collected metrics. The evaluation of the metrics can happen in mainly two ways:

- Reactive monitoring, in which the metric is compared over a threshold. The threshold could be defined, for example, as part of the descriptors included in the AIF graph Orchestration – NSD. Once the monitored metrics overcomes the threshold (e.g., CPU level higher than x%) a specific action is triggered.
- Proactive monitoring, in which the threshold is applied on a forecast (i.e., prediction) made over the metric. The prediction could be made, for example, using other AIFs in a serverless fashion. In

this approach, the system preemptively acts to avoid disruption (e.g., performance degradation or failures of the AIF/VNF).

An example of the actions that could be triggered by the monitoring includes scaling-up, scaling-down or migrating a service. Moreover, a less intrusive approach could send an alert notifying the risk of violating the defined threshold. Considering the importance of an automated monitoring and alerting system in an edge environment, its definition and implementation in AI@EDGE will be further studied in the next stages of the project.

5.3 AI@EDGE AIF Descriptor Information Model

5.3.1 Introduction

The AIF Descriptor (AIFD) is a deployment template, which consists of information used by the connect-compute platform for the life cycle management of an AIF. It is a part of the AIF package and describes AIF requirements and rules required by the AIF provider. An AIF Package contains all of the required files and meta-data descriptors required to validate and instantiate an AIF. It focuses on a holistic end-to-end view of the AIF Package lifecycle, from design to runtime, capturing development as well as operational views. The AIFD will be based on descriptors currently used for describing MEC Apps and Containers, and will reference or augment them, when necessary.

As it will described in more detail in the next paragraph, the AIFD may include, augment or reference descriptors of its constituent objects. The Descriptor models that will be considered or used as a reference during the definition of the AIFD are the following:

ETSI MEC Application Descriptor (AppD) – The MEC application descriptor (AppD), including its attributes, is defined in ETSI GS MEC 010-2 [44]. It may be included in a MEC application package, encoded e.g., in TOSCA or YANG format. MEC application specifies in its descriptor (AppD) [45] some MEC-specific fields, such as the maximum tolerated latency, the set of required MEC platform services, traffic rules that allow to redirect the traffic to the MEC application, and the preferred deployment location. An application Descriptor (AppD) is a part of application package and describes application requirements and rules required by application provider. The application descriptor AppD in an application package contains the traffic rules required for the data packets to reach the MEC app instance running on a MEC host. The onboarded MEC application package contains an application descriptor (AppD) specifying the application requirements and desired traffic steering rules. Traffic steering rules comprise of traffic filters to identify the packets, actions to be taken, and the destination interface to receive the packets (the MEO passes the traffic steering rules in the AppD to the MEP of the chosen MEC host). The ServiceDescriptor data type describes a MEC service produced by a service-providing MEC application.

HELM Charts – Helm is based on YAML and is a commonly used Kubernetes package and operations manager. A chart is a collection of files that describe a related set of Kubernetes resources: they contain the declarative Kubernetes resource files required to deploy an application. It can also declare one or more dependencies that the application needs in order to run. A single chart might be used to deploy something simple, like a cached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on. Helm charts are used to deploy an application, or one component of a larger application. Charts are created as files laid out in a particular directory tree. A Helm chart can contain any number of Kubernetes objects, all of which are deployed as part of the chart. A Helm chart will usually contain at least a Deployment and a Service, but it can also contain an Ingress, Persistent Volume Claims, or any other

Kubernetes object. They can be packaged into versioned archives to be deployed [46]. Helm charts are Kubernetes YAML files that accept variables.

OSM Information Model (VNFD) -The VNFD follows a format defined in OSM, augmenting SOL006, because the modeling of CNF or any Kubernetes applications has not yet been included in ETSI NFV SOL006. The CNF descriptor (VNFD) models one or more KDU (Kubernetes Deployment Unit) with the specified helm chart/s, connection points (mgmt-ext) where Kubernetes services of this helm-chart are be exposed, and certain k8s-cluster requirements. By default, it is assumed that the helm version for the helm charts is v3. If the helm chart is based on v2, the descriptor should add the line helm-version: v2 in the kdu section.

ONAP Application Service Descriptor (ASD) and packaging Proposals for CNF The Application Service Descriptor (ASD) contains the minimum information for the orchestrator, and pointers to cloud-native artifacts and code (including configuration) required for the LCM implementation. Helm Charts are the primary deployment artifact for a containerized application and ASD avoids any possible source of error or confusion that such duplication would cause.

5.3.2 AIF Descriptor Information Model initial definition

In this paragraph, we discuss the initial data structure definitions that will be used by the AIF descriptor information model.

The AIF Descriptor Information model is structured into domains and modules to differentiate between different types of information elements and their use. A core model provides generic information elements which are applicable to multiple interfaces. Each model is structured in Domains. In order to provide an end-to-end model view, it is possible to federate Information Models from different sources (MEC, NFV, etc.).

The **AIF's descriptor** is a descriptor provided by the AIF provider which describes the rules and requirements of an AIF module. When deployed in the MEC host, the AIF is deployed as a MEC Application, and the AIF descriptor will describe the rules and requirements of the AIF module as a MEC Application [47]. For this reason, the AIF Information Model will consider as a basis the MEC App Information model.

The **AIF package** is a bundle of files provided by the AIF provider to be on-boarded into the MEC system and used by the MEC system for AIF instantiation. It typically includes the AIF descriptor, a software image (in this case a container) or a URI to a software image, and a manifest file. It can contain also other optional files. The AIF package is a file containing the necessary information of an AIF, used by the MEC system for AIF lifecycle management.

Since the AIF is deployed in the MEC system as a MEC application, the AIF package contains also all the information required by a regular MEC application package (as described in [48]). The MEC application package unifies the MEC package format for both the classical and MEC NFV deployment cases. As such, it is defined in such a way that the package can be both used by the MEAO directly and can be on-boarded as a VNF package to the NFVO without any change.

To facilitate the integration, it is envisioned that the AIF application package format should be aligned with the MEC application package.

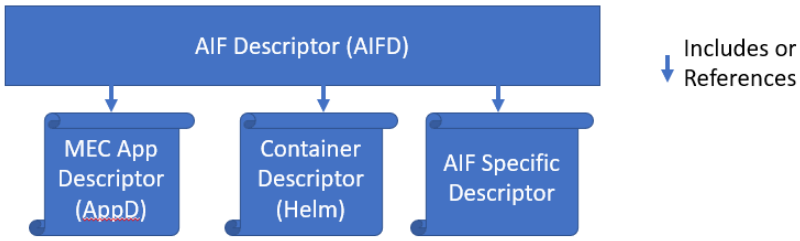


Figure 28 AIF Descriptor domains

AIF Information Model will consider as a basis the MEC App Information model. The MEC Application descriptor, the AppD, includes both MEC App LCM management information (e.g., it describes CPU, memory, hardware acceleration requirements; the reference to the virtual image of the MEC App; etc.) and MEC application configuration parameters, to be enforced by the MEC Platform. These include the description of MEC services:

- services a MEC application requires to run.
- services a MEC application may use if available.
- services a MEC application is able to produce to the platform or other MEC applications. Only relevant for service producing app
- features a MEC application requires to run.
- features a MEC application may use if available.
- Transports, if any, that this application requires to be provided by the platform. These transports will be used by the application to deliver services provided by this application. Only relevant for service-producing apps. This attribute indicates groups of transport bindings in which a service producing MEC application requires to be supported by the platform in order to be able to produce its services. At least one of the indicated groups needs to be supported to fulfill the requirements.

The AppD also reports the information necessary for handling the traffic and implementing the route of IP packets to MEC applications:

- The traffic rules the MEC application requires.
- The DNS rules the MEC application requires
- the maximum latency tolerated by the MEC application.

Since the AIF is deployed in the MEC system as a MEC App, also the AIF descriptor should allow to report the same information.

Attribute name	Cardinality	Data type	Description
appDId	1	String	Identifier of this MEC application descriptor. This attribute shall be globally unique. See note 1.
appName	1	String	Name to identify the MEC application.
appProvider	1	String	Provider of the application and of the AppD.
appSoftVersion	1	String	Identifies the version of software of the MEC application.
appDVersion	1	String	Identifies the version of the application descriptor.
mecVersion	1..N	String	Identifies version(s) of MEC system compatible with the MEC application described in this version of the AppD.
appInfoName	0..1	String	Human readable name for the MEC application.
appDescription	1	String	Human readable description of the MEC application.
virtualComputeDescriptor	1	VirtualComputeDescription	Describes CPU, Memory and acceleration requirements of the virtual machine.
swImageDescriptor	1	SwImageDescriptor	Describes the software image which is directly loaded on the virtualisation machine instantiating this Application.
virtualStorageDescriptor	0..N	VirtualStorageDescriptor	Defines descriptors of virtual storage resources to be used by the MEC application.
appExtCpd	0..N	AppExternalCpd	Describes external interface(s) exposed by this MEC application.
appServiceRequired	0..N	ServiceDependency	Describes services a MEC application requires to run.
appServiceOptional	0..N	ServiceDependency	Describes services a MEC application may use if available.
appServiceProduced	0..N	ServiceDescriptor	Describes services a MEC application is able to produce to the platform or other MEC applications. Only relevant for service-producing apps.
appFeatureRequired	0..N	FeatureDependency	Describes features a MEC application requires to run.
appFeatureOptional	0..N	FeatureDependency	Describes features a MEC application may use if available.
transportDependencies	0..N	TransportDependency	Transports, if any, that this application requires to be provided by the platform. These transports will be used by the application to deliver services provided by this application. Only relevant for service-producing apps. See note 2.
appTrafficRule	0..N	TrafficRuleDescriptor	Describes traffic rules the MEC application requires.
appDNSRule	0..N	DNSRuleDescriptor	Describes DNS rules the MEC application requires.
appLatency	0..1	LatencyDescriptor	Describes the maximum latency tolerated by the MEC application.
terminateAppInstanceOpConfig	0..1	TerminateAppInstanceOpConfig	Configuration parameters for the Terminate application instance operation.
changeAppInstanceStateOpConfig	0..1	ChangeAppInstanceStateOpConfig	Configuration parameters for the change application instance state operation.
NOTE 1: The appDId shall be used as the unique identifier of the application package that contains this AppD. NOTE 2: This attribute indicates groups of transport bindings which a service-producing MEC application requires to be supported by the platform in order to be able to produce its services. At least one of the indicated groups needs to be supported to fulfil the requirements.			

Figure 29 Attributes of the MEC AppD. [49]

5.3.3 AIFs' features and attributes (relevant to the AIF descriptor)

The MEC application descriptor specifies the attributes that need to be provided for the MEO (mobile edge orchestrator) to deploy the MEC application (Figure 28). An AIF, when deployed at the MEC, is also a MEC application with these specifications also applying to them. However, an AIF application is qualitatively different from a generic MEC application because it has an AI function which brings with it an additional set of attributes to define for deployment. These attributes should be informed by the type of AI functions implemented and should be applicable to a wide range of AIFs. Here we list additional AIF

specific attributes that need to be provided to the MEO to enable the orchestration of AIF MEC applications. An important requirement for these AIFs is the ability to deploy them in a distributed manner for federated learning. The federated learning paradigm requires a communication/transport channel between distributed AIFs to exchange data and model parameters throughout the lifecycle of learning. These channels need to be defined with the required constraints for each specific AIF application. Several AIFs also have requirements on the properties of the data sources that serve as their input. Time granularity and probabilities associated with predictions are some examples. The output from an AIF is also subject to some additional requirements that are necessary when their output is used as an input by other AIFs. Declaration of dependencies is important to identify loops where an AIF (A) could be using, as input, predictions from another AIF (B) that used AIF (A)s output as input. AIFs also need runtime attributes, such as a debug feature for debugging and a data augmentation feature for model updates (or retraining). In addition, the AIF should have an attribute that provides information of the actual model used. Considering all the points discussed above we have consolidated the additional attributes with a short description below.

- Model update requirements
    - Trainable: A flag to identify whether this AIF implements a retrainable model
    - Does the ML model in the AIF need to be updated, either periodically or on trigger?
    - How often/ when does it need to be updated? This is important for the orchestrator to know how often this MEC application needs the resources required for model updates
    - Compute and memory resources required while retraining, or updating the model
- Data source requirements
    - The list of data sources it needs
    - The list of data sources it shall use if they are available
    - This AIF requires input data sources to be of at least the specified time granularity to run.
    - The bandwidth required on the transport interface that connects this AIF to the data sources. This depends on the publish frequency and the size of the metrics in the data sources.
    - The latency requirement for the data sources. The values in the data source are only valid if they are consumed by the AIF within a certain time (so that the data is not stale).
    - If the input to the AIF is the output of another AIF then confidence intervals associated with them need to be above a certain value
- Transport channel requirements for distributed AIFs
    - Transport/communication channel requirements of this AIF with other AIFs that need to coordinate with it. This includes a list of other AIFs with which it needs to relate to, and the latency and bandwidth requirements for this communication channel. This is essential for AIFs that are part of a federated learning system. The information exchange between these AIFs can be data or model parameters.
- AIF output requirements
    - Configures whether the AIF output (predictions/decisions) is periodic, or event driven.
    - If it is periodic then configure the attribute to set the time.
- AIF dependency

    - Each AIF must include a list of other AIFs that it intends to use so that a dependency tree can be built to avoid loops of AIFs with dependency loops.
- Privacy domains for AIFs to restrict data and inference sharing between different verticals but allowing it within the same vertical.
- GPU requirements
    - GPU compute
    - GPU memory
- Debug features
    - Debug run: If set, run step by step for debug purposes at the cost of slower execution time.
- Augment data: whether to do data augmentation for this AIF
    - if the model is trainable, using this attribute to determine whether to use data augmentation, and also to choose which augmentation operations to use.
- Model information:
    - Model list: return the list of models.
    - Model summary: return the basic information of each of the models, like how many layers and how many parameters.
- Distribute strategy: it describes how this model should be distributed when implementing model training (e.g., data versus model parallelization)

6. Cross-layer, Multi-Connectivity Aggregation and Scheduling Technologies

Multi-connectivity (MC) is a key function for 5G networks and enhances the network performance (high reliability, low latency, and high throughput) in many use-cases (from eMBB that targets high-capacity applications to the mMTC and URLLC applications that requirements very low latency). By allowing the user to simultaneously use multiple connections, MC increases transmission mobility robustness, reduces handover (HO) interruption time, and increases the overall reliability of the network. The multiple connections can use multiple independent communication paths, nodes, access points (APs), or base stations (BSs) for data transmission to a UE.

Concerning standardization, MC has evolved from 3GPP Release 12 (with the LTE dual connectivity) to the 5G new radio (NR) multi-radio dual connectivity (MR-DC) paradigm in Release 16 and will be enhanced in Release 17. In Release 12, 3GPP introduced the Intra-E-UTRA Dual Connectivity (DC) which allows dual connectivity to two LTE BSs where both BSs are connected to the EPC (Evolved Packet Core). In Release 16, with the introduction of 5G NR, 3GPP introduced four configurations for Multi-Radio Dual Connectivity (MR-DC):

- MR-DC with EPC (E-UTRA-NR Dual Connectivity (EN-DC))
- MR-DC with 5GC configurations:
- NR-E-UTRA Dual Connectivity (NE-DC)
- NG-RAN E-UTRA-NR Dual Connectivity (NGEN-DC)
- NR-NR Dual Connectivity (NR-DC)

While EN-DC, NE-DC, and NGEN-DC fall under the 5G NSA architecture involving two APs of different RATs, the NR-DC represents the 5G equivalent of the LTE DC. In NR-DC, the EPS (Evolved Packet System) bearer split takes place at the PDCP layer, similarly as in LTE. Depending on where the PDCP layer is deployed, two sub-architectures are being explored. The first is the Xn-based NR-DC architecture where the PDCP sublayer is located in each gNB in a distributed fashion. The second NR-DC architecture is the fronthaul split-based approach, in that the PDCP sublayer is located in a centralized unit (CU) in the cloud while all (sub)layers below the PDCP are located in each gNB distributed unit (gNB-DUs). The 5G networks (and beyond) are poised to deliver enhanced quality of service through increased throughput and reliability as well as reduced latency, HO frequency, and probability of radio link failures.

Besides the multi-connectivity which uses 3GPP access only as above, some solutions try to combine using none-3GPP as well. While 3GPP access refers to the nominal UU New Radio (NR-Uu) interface with the RAN of a 5G network; non-3GPP access instead opens to the possibility of connecting a UE to the 5G CN via different access technologies, and in particular WiFi (WLAN). Since Release 16, it is possible also to enable the optional feature called Access Traffic Steering-Switching-Splitting (ATSSS), in which both 3GPP access and non-3GPP access are used simultaneously. In ATSSS, “steering” refers to the possibility of selecting for user-plane traffic, according to the service (QoS-type for a data flow), the best link to use, “switching” describes the possibility of performing handover without service interruption to the other link, when necessary, “splitting” means the simultaneous use (bonding) of the two links. By allowing the use of different RATs, ATSSS consolidates data at the transport layer by using Multiple Path TCP (MPTCP).

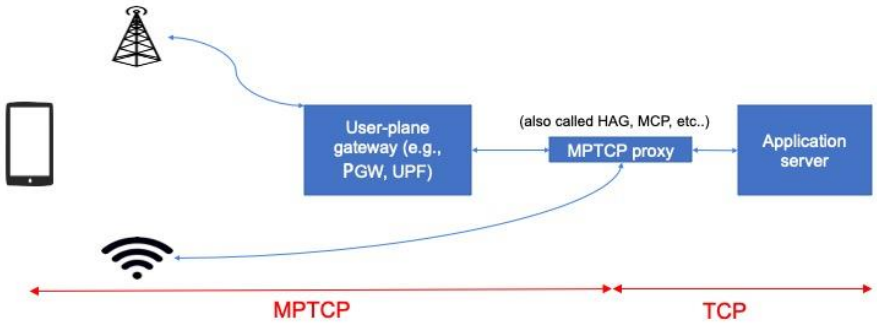
6.1 Multi-Connectivity Specific Subsystems

The Multipath TCP (MPTCP) [50] allows the Conventional MPTCP communication between MPTCP capable devices. In detail, the MPTCP protocol architecture allows packets of the same TCP connection to be sent via different paths to an MPTCP-capable destination. The MPTCP paths are called subflows and are defined by pairs of source and destination IP addresses or ports. Ideally, the number of sub-flows is equal to the full mesh interface topology; but not all subflows necessarily used (some of them can be used as backup). The scheduler is responsible for determining which subflow will be used to send data at each time. In case one of the terminals does not have MPTCP support, MPTCP still can be deployed thanks to MPTCP proxy function. It converts TCP traffic to MPTCP traffic if the endpoint supports MPTCP and converts MPTCP traffic to TCP traffic if the endpoint does not support MPTCP. In this section, we will a present multi-connectivity model using MPTCP proxy.

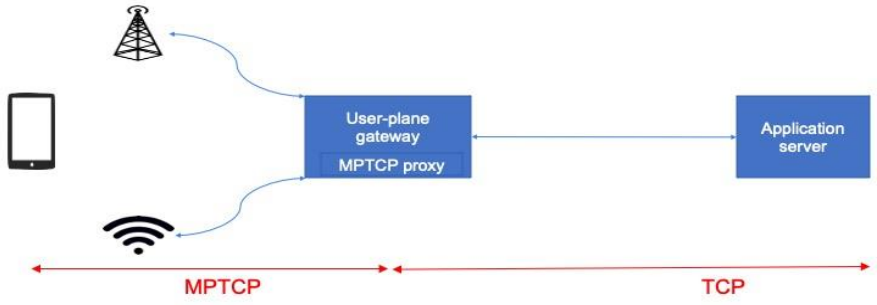
6.1.1 MPTCP Proxy Architecture

In this section, we will present three different models of MPTCP proxy, based on the position of proxy. The models are differentiated based on the deployment location of the MPTCP proxy.

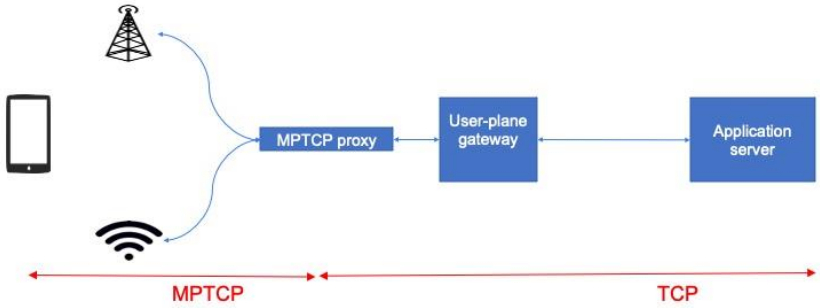
Off-path and On-path models



(a) off-path model



(b) on-path model



(c) on-path model variant

Figure 30 On-path and off-path models for MPTCP proxy usage in multi-connectivity scenarios

- Off-path model (*Figure 30*a): the MPTCP proxy sits after the user-plane cellular core gateway, i.e., the subflows join after the gateway. The position of the proxy can be anywhere along the IP path from the user-plane gateway and the server, and therefore the multiple RATs can be under the control of different access operators. The address of the proxy may need to be configured in the UE in case of additional paths not crossing the proxy in the uplink direction.
- On-path model (*Figure 30*b): the MPTCP proxy sits within the user-plane gateway and the subflows join at the gateway. Therefore, the multiple RATs are meant to be under the control of the same operator.
- On-path model variant (*Figure 30*c): a variant consists in having the MPTCP proxy before the user-plane gateway as a network function independent from the UPF (but it would be unfeasible in standard 4G/5G settings due to GTP tunneling).

ATSSS system

The integration of MPTCP proxies in 5G systems has been envisioned since Release 16 [51] following the on-path model. It is referred to as ATSSS (Access Traffic Steering, Switching and Splitting) and the MPTCP proxy is integrated within the UPF. A complete 5GC ATSSS system description is expected for Release 17. The current ATSSS specification is depicted in

Figure *31*; it encompasses the integration of an MPTCP proxy for multi-RAT bonding at the UPF level, with the exploitation of the PCF (Policy Control Function) to regulate the scheduling over the RATs, and the PMF (Performance Measurement Function) to gather real-time packet-level measurements to allow dynamic MPTCP scheduling update.

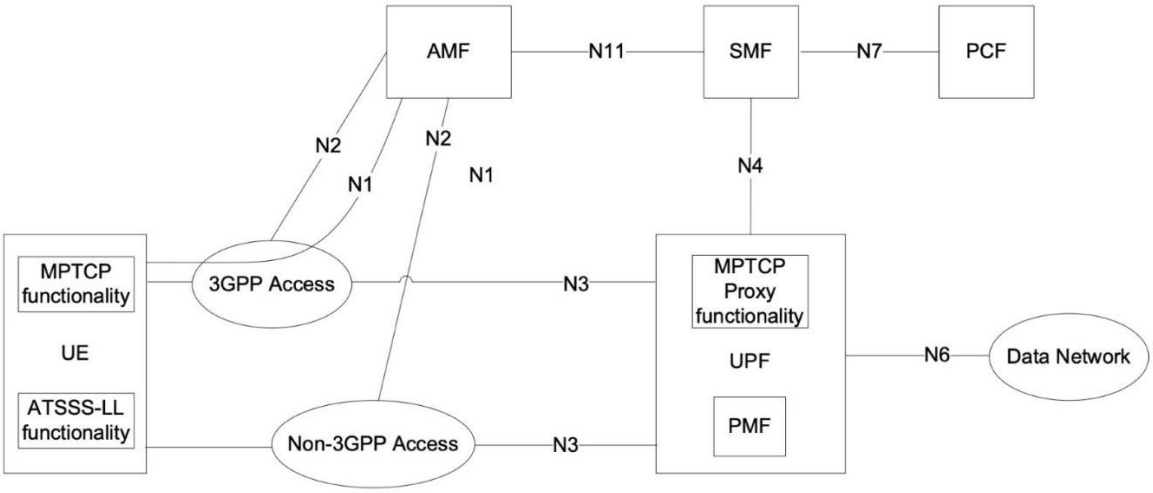


Figure 31 Access Traffic Steering, Switching and Splitting (ATSSS)-capable 5GC system. Source: [52]

In the framework of AI@EDGE, and in particular UC4 activities, we plan to design a gradual integration of ATSSS in the AI@EDGE platform and to design predictive scheduling algorithms for downlink communications at the UPF MPTCP-proxy level. Indeed, basic MPTCP schedulers presents in open-source implementations are reactive schedulers, changing the decision on which packet to send over which sub-flow upon sub-flow state changes. Available open-source implementations use packet-level latency and buffer occupancy measures collected in real-time at the socket level, as per the MPTCP standard [45]

6.2 Scheduling Challenges to Overcome

TCP has a limited inability to change connection parameters without severing the connection. The MPTCP protocol has been proposed to provide a considerable improvement by using multiple paths transparently to improve throughput, to support better use of multiple connectivity through capacity aggregation and seamless failover. However, capacity aggregation over heterogeneous paths, such as offered by cellular and Wi-Fi networks, is problematic. It causes packet reordering leading to head-of-line (HoL) blocking at the receiver, increased end-to-end delays and lower application goodput [53]

The role of a typical scheduler is to consider the following points:

- Hole blocking
- Connection failure
- Packet loss
- Delay variation

The scheduling algorithm can be based on the control information of a single layer or of several simultaneous layers. In the following, we will define single layer and cross-layer scheduling algorithms.

6.2.1 Single Layer Scheduling

In theory, the very good knowledge of the specifications and parameters of a network (like bandwidth, delay) allows the scheduler a perfect and optimal planning of the packets transmitted on the various paths, while avoiding the problems mentioned above. In a real implementation, the delay and the interference are variable as well as the bandwidth which varies according to the other TCP connections; the sought MPTCP scheduler must take into account these parameters.

MPTCP Scheduling

The MPTCP, improve exploiting multiple available network resources simultaneously for multi-homed devices. Several main benefits are brought: transferring data simultaneously through all the available paths, maintain connection if one of the path fails and also providing bandwidth aggregation.

However, the latencies heterogeneity of different paths, such as those offered by cellular, Wi-Fi and Ethernet, conducts to the existence of out-of-order packets at the receiver level, leading to head-of-line blocking. A large number of these packets exhaust the limited receiving buffer and make the receive window be stalled, which significantly degrade the throughput. Thus, an efficient scheduling mechanism will play an important role to keep in-order delivery [49]

The default MPTCP scheduler of the Linux Kernel implementation, min RTT (Round Trip Time), always starts by selecting the sub-flow with the smallest round-trip-time to send data. This scheduler is not sufficient to achieve good performance on memory-constrained devices that use a small receive window [47]. An ideal single layer MPTCP scheduling should consider the heterogeneous interface characteristics, i.e. latency, bandwidth, packet loss rate. Several researchers have proposed other types of schedulers for MPTCP. A Forward Prediction Scheduling [54] is a scheduling mechanism that predicts data packets reception times in advance, and then stripes packets onto multiple paths in a way that the packets are received in-order. Figure *32* illustrates the scheduling scenario, when the larger delay path *j* starts to send data, the MPTCP proxy predicts there could be five packets sent on other subflows (subflow i in this example). The sender keeps the numbers of the packets 1 to 5 for subflow *I* and the packets arrive in order to the UE.

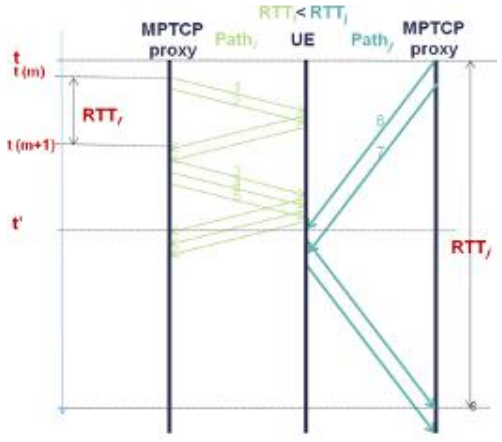


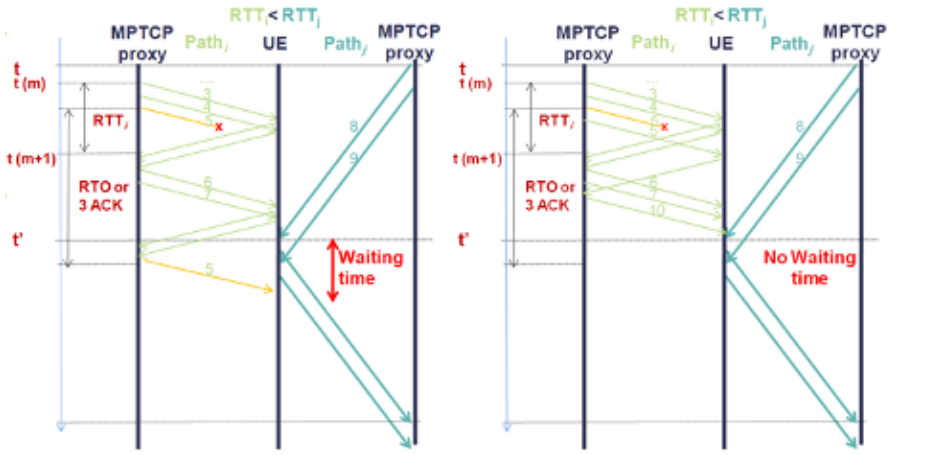
Figure 32 The scheduling FCS scenario

This algorithm takes into consideration the RTT difference between different paths without considering the dynamic effect of path changing and the packet loss probability into consideration.

The Forward Prediction based Dynamic Packet Scheduling mechanism (FPDPS) [46] is close to FPS, but is more robust in lossy heterogeneous networks. When a subflow is under-scheduling, the sender predicts the size varying of TCP's sending window for each faster subflow in the same connection. FPDPS utilizes maximum likelihood estimation to estimate the data amount (*N*) sent on them during one successful delivery time on the under-scheduling subflow. The FPDPS is a recursive procedure provides the estimation of scheduling value of the key parameter *N*. the estimation function of FPDPF algorithm as well as the different recursive situations are detailed in the reference [48].

The precision of the predicted value typically depends on the accuracy of the prediction algorithm input parameters, the RTT and the error rate often change in wireless networks which implies a deviation from the predicted value of N. Hence the use of feedback information collected by the proxy MPTCP allows, based on the Dynamics Adjustment with the feedback of SACK (DAF), to make an offset to eliminate the previous scheduling deviation for this round of scheduling.

Two situations can exist whether the estimated number N is too large or too small, in both cases based on the dual sequence numbers used by the MPTCP which are the DSN (Connection level sequence number) and the SSN (sub-flow level sequence number), the transmitter can know the actual number of the transmitted packets in the last round and infer the elimination deviation. The DPSAF [55], Dynamic Packet Scheduling and Adjusting with feedback based on the basic FPDPS packet scheduling mechanism and the DFA, Dynamic Adjustment with the feedback of SACK. This algorithm considers both packet loss and time delay, which can be more adapted to wireless packet loss.



(a) without data link synchronization (b) with data link synchronization

Figure 33 Scheduling scenario

The DPSAF protocol takes into account packet loss and gets feedback information from SACK options to fix the scheduling value, but the detection of packet loss and the decision to retransmit it will impose a delay time depending on RTO or acknowledgement delay time, as shown in Figure 33a where packet 5 of sub-flow I is lost. If the transmitter can capture information from the data link layer such as the channel quality indicator, the bitrates, the percentage of packet dropped and other data link parameter, it can predict the loss of packet 5 for example before it receives the SACK and then retransmit it directly, Figure 33b. This will improve the throughput and reduces cache occupancy at the receiver, hence the need to operate cross-layer scheduling.

PDCP Scheduling

When the user equipment (UE), accesses several networks with different radio access technologies (RATS), data can reach the UE from the primary base-station via multiple secondary base-stations. By taking advantage of the benefits of using such multiple paths, the Packet Data Convergence Protocol (PDCP) protocol duplicates the protocol data unit (PDU) and sends it to the user equipment via multiples and various paths. This PDU duplication is carried out at the data link layer implies that these various PDUs refer to the same packet having the same sequence number (SN). The PDCP is based on the duplication of the same PDU on various Radio Link Control (RLC) entities. Separate RLC PDUs can use the same MAC entity via carrier aggregation or go through several entirely separate MAC entities such as for example 5G and WIFI networks. The primary aim of duplication is to achieve reliability, among the challenges of duplicating PDCP data, is the selection of the right PDU on reception, the basic idea is to select the first suitable PDU received and to discard the other PDUs. This approach works as long as one of the paths succeeds in reliable communication and fails if all paths are in error, resulting in a forced retransmission of the entire block which increases the latency due to the RTT and what will degrade the throughput. The study in [56] proposes data duplication at the PDCP of the transmitter and combining at the PDCP of the receiver. The PDCP at the transmitter transmits duplicate copies of data through multiple paths, possibly through multiple RATs. If all individual paths fail in their checks at the receiver, multiple copies of data are combined. This procedure of combining data shows a significant improvement in the Block error rate (BLER). The improvement in BLER also results in throughput enhancement. These schemes will offer cell edge users better reliability offered by a different RAT. Since an improved BLER implies a reduced number of retransmissions, the average latency is also improved.

6.2.2 Cross-Layer Scheduling

MPTCP-PDPC

After having indicated the advantages of using the MPTCP scheduling as well as the contributions of the implementation of PDPC duplication, we find it necessary to exploit the possibility of the implementation of cross-layer scheduling based on MPTCP-PDPC. For each MPTCP subflow we can impose the applying of PDPC duplication on the various RLCs which will surely improve the reliability with respect to packet loss as well as connection failure.

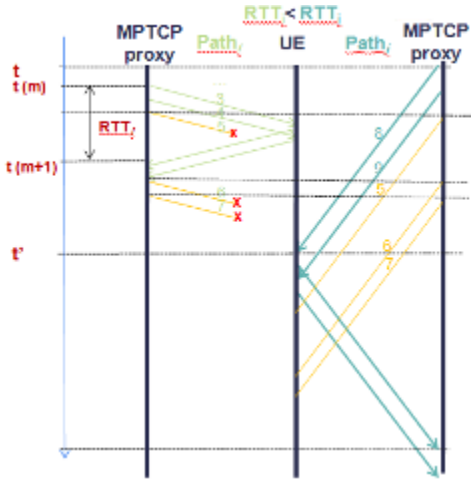


Figure 34 Duplicate scheduling scenario without data link synchronization

The fully duplication of MPTCP traffic via PDCP will certainly consume the bandwidth on the various interfaces. Good knowledge of a particular indicator of the link-layer by MPTCP layer, such as the CQI (Channel Quality Indicator) and percentage of packet dropped can allow synchronization between layers to implement a cross-layer scheduling. A proposed solution based on applying PDCP scheduling on a specific MPTCP sub-flow when a binding indicator appear. In the example of *Figure 34*, and after an interpolation phase of data link indicators we can estimated an degradation of path *i* when sending the packet number 5: in this case we can release the PDPC scheduler to duplicate the sub-flow *I* starting by packet 5 on the others interfaces in order to guarantee a good level of redundancy, and to minimize hole blocking time. This solution can be serviced in case of a strong service degradation or connection loss on one interface. In this case the pre-scheduled traffic on this interface is duplicated to another interface based on the PDCP.

MPTCP-content distribution

Having the scheduler know more information from other layers provides an additional valuable source of data to support decision making. On that basis, we will coordinate among the content curation algorithms and the MPTCP scheduler to try to induce the best scheduling decisions on a per-content basis.

6.3 Experimental Evaluation

Our plan is to contribute integrating this multi-connectivity innovation in the AI@EDGE platform and experiment with novel predictive schedulers for the MPTCP proxy functionalities.

For the multi-connectivity activities, two evaluation modalities will be considered.

- First, whenever possible, a real integration with the WP4 testbed, with few real UEs to perform functional tests and low-scale experimental evaluation of the proposed features.
- Second, a large-scale mixed simulation experimentation environment with many dozens of simulated UEs, a dozen of eNodeB and few dozens of WiFi APs.

For both evaluation environments, we plan first to demonstrate the usage of the off-path model, with a programmable reactive scheduler exposing through a dedicated API its configuration. Then, we plan to move to the on-path model, using the project near-RT RIC to integrate (part of) the PMF functionalities and the MPTCP proxy as a function sitting before the UPF along the user-plane path. Possible evolutions then include the integration of the PMF and MPTCP proxy functionalities at the UPF, depending on its availability.

The testbed for the second evaluation environment, is in the progress of construction. Figure 35 represents this environment (with a simplified number of UE and ENodeB).

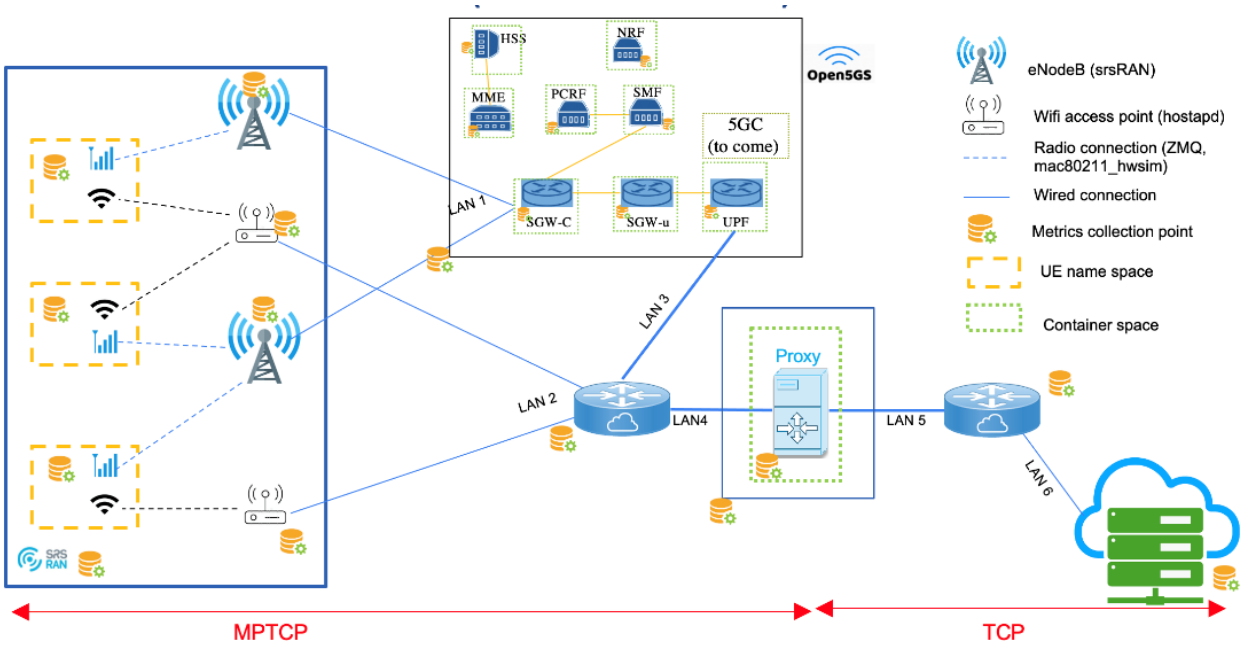


Figure 35 Proof of concept in off-path testbed

The simulation environment includes (see section 2.6):

- 75 UE nodes, each runs in an independent namespace and using srsRAN. Each UE haves two network interfaces, one WiFi, and one mobile network.
- 16 eNodeB/NSA gNodeB nodes run in independent namespace and using srsRAN. Each eNodeB has two UEs.
- Radio front-end links use the ZMQ Virtual Radios to transfer radio samples between UE and eNodeB.
- WiFi-AP using hostapd (HOST Access Point Daemon - a userspace daemon software enabling a network interface card to act as an access point and authentication server).
- Open5GS core runs as a 5G core part and is deployed in the Kubernetes system.
- MPTCP proxy runs in a container.
- Application servers emulation using NS3 and run in the independent server.

7. Hardware Acceleration Solutions for AI/ML

Endowing edge nodes with acceleration capabilities, i.e., with 10x additional compute power at the cost of a limited extra size/energy consumption, will be crucial for the success of 5G/6G applications. Decreasing the latency and increasing the throughput of AIF, either for network automation or user applications, requires some sort of parallelization inside purpose-built HW. The market provides a plethora of programmable HW acceleration devices: FPGA, GPU, TPU, VPU, VLIW CPU, multi-core DSP, etc. Furthermore, this pool of available devices has a wide range of power/size specifications to support servers ranging from big datacenters to small cloudlets, or even down to embedded systems and IoT boards. For the latter case, the accelerators are integrated inside very heterogeneous System-On-Chip architectures combining CPU+peripherals+accelerators, which allow building Single Board Computers with minimal power/size (e.g., one order of magnitude smaller than server-class devices). Such a variety of solutions creates a huge search space for AI@EDGE. Overall, the most prominent devices and vendors in the market today are the following: Xilinx for FPGA-based accelerators, Nvidia and AMD for GPUs, Intel for FPGA-/GPU-/VPU-based accelerators (e.g., Myriad VPU targeting mostly embedded applications), Google for TPU-based devices (Tensors parallelizing only AI layers, not general-purpose functions like in the case of FPGA/GPU), and TI for DSP/VLIW-based boards. As a proof-of-concept for the Connect-Compute platform, the AI@EDGE project has selected devices from the former two sets, i.e., Xilinx FPGA and Nvidia GPU, because they cover a vast range of applications and, historically, they provide the most representative results in terms of acceleration with programmable HW.

The current section will briefly describe which are the specific HW accelerators of AI@EDGE, their architecture, how they are introduced in servers/nodes at the edge, how they are programmed, and how they will be exploited by users. Additionally, for all the above, the paragraphs explain the choices taken in AI@EDGE. The first subsection describes the HW aspects and the acceleration testbed/servers used in AI@EDGE. The second subsection describes the SW aspects, i.e., how the accelerators are programmed and used/called, together with constraints & requirements pertaining to the choices made.

7.1 HW Architecture of Processing Nodes Including HW Accelerators

The HW accelerators primarily considered in AI@EDGE are daughterboard cards, i.e., FPGA or GPU microchips mounted on their own PCB card, which are placed inside a server node and connected to the main CPU via PCIe expansion slots. Such accelerators are the NVIDIA GPUs V100/P4/T4 and the Xilinx FPGAs Alveo U50/U200/U280. As a representative example, *Figure 36*, left, depicts a 1U-server supermicro 1029GQ-TRT at ICCS premises with two CPUs Intel®Xeon®Gold 6138 (in the center, under two coolers and between 12 DIMM RAM slots), together with an Alveo U280 card hosting a relatively big FPGA accelerator (upper side, inside the red chassis). Furthermore, the block diagram in Figure 36, right, shows the logical block diagram for a similar CPU+GPU server, which utilizes two PCIe switches and mounts up to 8 accelerator cards. In such setups, the power consumption of each accelerator card varies in the range of 70 to 300 Watts depending on active/passive cooling, interfaces, accelerator's VLSI area, etc. The form factor of such a card fits in 1U servers and its actual size lies in the area of 10x30cm (e.g., for Alveo U200/280 FPGAs, HxLxW(mm)=111.28x236x39.04- -111.15x242x39.04, Weight=1066-1130g).

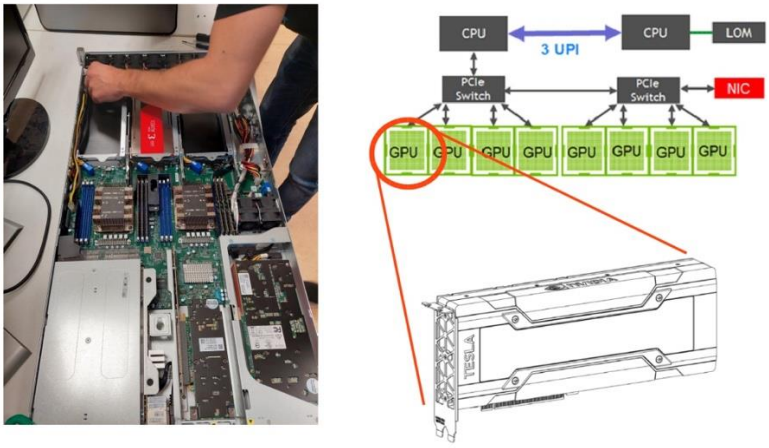


Figure 36 Left: ICCS' edge "supermicro" server with a Xilinx Alveo FPGA accelerator card. Right: block diagram of a server with PCIe (copyright supermicro.org) able to support multiple GPU cards (copyright NVIDIA).

Secondarily, AI@EDGE will consider smaller accelerators integrated inside embedded SoC processors, better suited to remote sites at the near edge (and IoT setups). Such cards combine CPU with FPGA or GPU accelerators inside the same main microchip, and hence, they are considerably smaller than the server+accelerator setup. Typically, the size of such a card is smaller than 10x10cm and can reach down to that of a USB stick. The power consumption varies in the range of 1-10 Watt. As representative examples, Figure 36, left, shows four embedded accelerators: two USB sticks with size comparable to that of a paper-clip (Google Coral TPU and Intel MyriadX), which can be attached via USB to a CPU motherboard, as well as two Single Board Computers with FPGA and GPU accelerators inside their main SoC (Xilinx Zynq from Xiphos and Jetson Nano from NVIDIA). Bigger, more powerful embedded systems include the NVIDIA Jetson AGX Xavier board and FPGA developer kits such as Xilinx RFSoC and Versal. However, the power consumption of these cards increases to 20-50 Watts.Figure 36, right, shows on VCK190 board the latest Xilinx Versal ACAP SoC, which is a very heterogeneous microchip embedding FPGA fabric, ARM CPU cores, AI engines, etc. Such Single Board Computers are self-contained, i.e., they can operate as a stand-alone near-edge node, but their size increases in the area of 30x30cm. We note that the embedded accelerators, especially the small microchips consuming 1-10W, provide one order of magnitude smaller computational power than the aforementioned server-class accelerators.

Figure 37 Left: small SoC devices with embedded FPGA/GPU accelerator (copyright Xiphos, NVIDIA, Intel, Google). Right: bigger FPGA SoC with development board (Xilinx ACAP Versal)

From the microarchitecture point of view, an FPGA fabric consists of a huge number of small blocks that each one covers a limited area, i.e., a DSP block with a single MAC unit, a RAM block of 36Kbits, a Configurable Logic block with a couple of look-up-tables implementing only 6-bit functions. Figure 37, left, depicts such a generic FPGA microarchitectural diagram regardless of device class (embedded or server). The computational power of the FPGA stems from the fact that these blocks can be interconnected at compile time to create very big and parallelized circuits/functions, as well as the fact that the number of today blocks reaches the order of one million (e.g., 1M LUTs and 10K DSPs for Xilinx Alveo FPGA). In contrast, the GPU microarchitecture includes much bigger cores that each one implements one ALU operation with integer or floating-point numbers (more like an ordinary CPU core). Instead of a million blocks, the number of CUDA cores in a GPU lies in a thousand (e.g., from 0.5K for AGX Xavier to 5K for V100). Alongside these CUDA cores, modern GPUs also include fewer Tensor cores (e.g., from 48 for AGX Xavier to 640 for V100), which are more customized for AI computations (each one performs a parallelized matrix multiplication). Figure 37, right, depicts the architecture of AGX Xavier with CUDA and Tensor cores grouped in Streaming Multiprocessors (SM), as well as the memory hierarchy. The methods to program such complicated microarchitectures as FPGA and GPU are explained in the following subsections, together with the relevant programming approach decided for AI@EDGE.

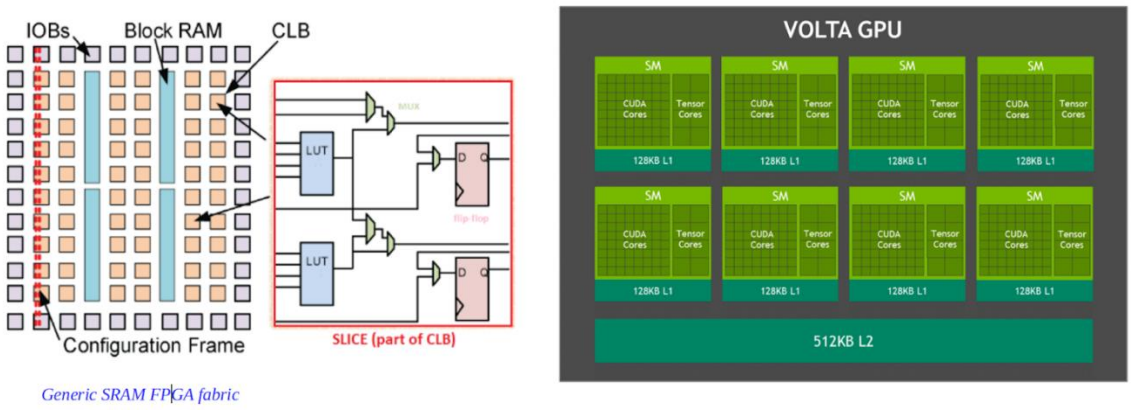


Figure 38 HW microarchitecture aspects of FPGA (left) and GPU (right) accelerators. FPGAs rely on significantly more, but smaller blocks to parallelize the computation compared to GPU cores (e.g., order of 1M vs 1K)

HW Acceleration in Heterogeneous Testbed for AI@EDGE

Based on the above landscape of HW acceleration, ICCS will build its own testbed-cluster at ICCS premises in order to study the accelerators' performance and develop custom solutions for AI@EDGE. The key idea is to utilize a number of servers and accelerators to emulate edge computing scenarios involving multiple nodes of diverse compute capabilities each, to test various integration approaches, to study orchestration techniques, measure AIF deployment efficiency, all while developing certain FPGA/GPU code to accelerate representative AIFs of AI@EDGE. *Figure 39*, shows a high-level description of the HW equipment to be assembled at ICCS. We will use three supermicro servers and two self-contained embedded processors. The latter two can emulate "remote" nodes at the far-edge, whereas the former three can emulate nodes at the near-edge with increased computational power (or even a single near-edge cluster/cloudlet). The Ethernet inter-connections of these nodes can vary in terms of speed/latency to support various network scenarios. The three supermicro servers will represent three different cases: a server with FPGA acceleration, a server with GPU acceleration, a server without acceleration. Furthermore, we will also use one of the servers to assume multi-card acceleration scenarios on a single node by equipping a server with 2-3 FPGA cards. Regarding the "remote" nodes, we will utilize an ultra-low-power GPU-based SoC and

the latest state-of-the-art FPGA-based SoC for diversity purposes. More specifically, the key characteristics of the 5 nodes in the ICCS testbed-cluster are:

- Intel Xeon E5-2658A without accelerators (8 CPUs, 8GB RAM). Also to be considered as the master of the local cluster.
- Intel Xeon Gold 6138 with 1 GPU (8 CPUs, 16GB RAM, Nvidia GPU Tesla V100). To be considered as a worker in the cluster.
- Intel Xeon Silver 4210 with 2 FPGAs (4 CPUs, 16GB RAM, Xilinx Alveo U200, Xilinx Alveo U280). To be considered as a second worker in the cluster.
- Xilinx Versal AI Core Series VCK190 Evaluation Kit. To be considered as a high-end remote node utilizing the latest FPGA acceleration technology (ACAP with AI engines)
- NVIDIA Jetson Nano Developer Kit. To be considered as a low-power low-end remote node with GPU acceleration.

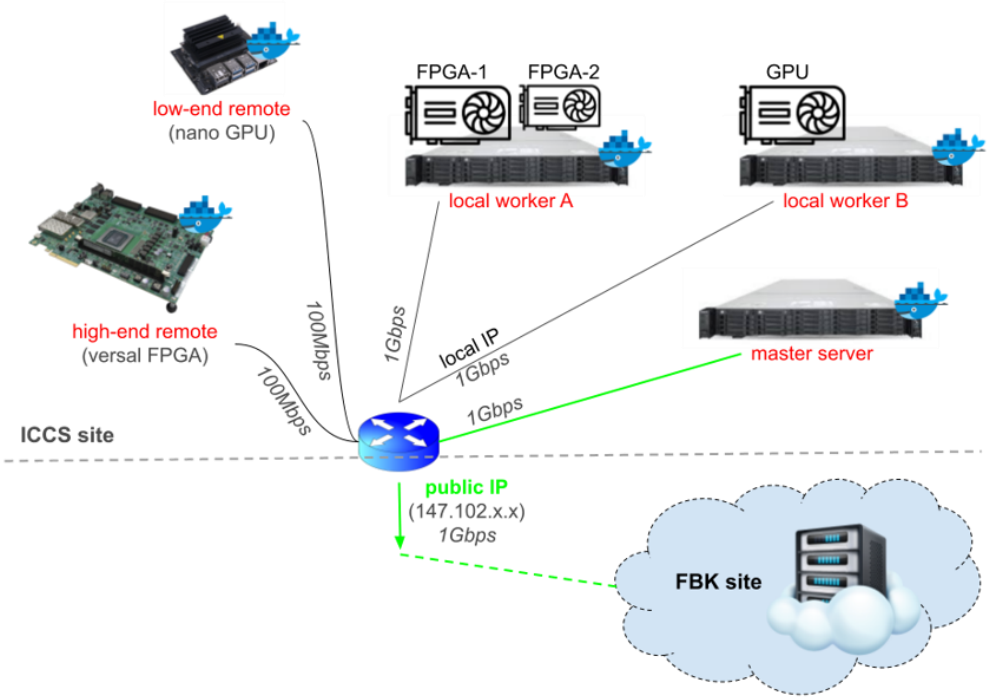


Figure 39 The testbed-cluster being assembled locally at ICCS for developing and testing the acceleration solutions of AI@EDGE

HW Acceleration with GPU for supporting distinct edge scenarios in AI@EDGE

Given the more general architecture of AI@EDGE, Italtel will examine how to leverage a multi-layer cloud/edge architecture spanning from end-user terminals to the centralized public/private cloud, capable of hosting AIFs and AI/ML tasks such as inference, local training, and global training across the different domain of the network. Indeed, the project promotes the vision of a new generation of AI-enabled applications obtained through the chaining of multiple AIFs across the converged connect-compute

platform. For this reason, when selecting GPUs for providing HW acceleration support for running AIF, it is essential to consider the different characteristics and constraints associated with each domain. NVIDIA provides GPU products capable of addressing the various computational environments in terms of power consumption and costs. Two possible architectures are considered in the project:

- Hosted on the PCI bus of a standard server (Table 5).
- Embedded with the CPU (SoC), typically ARM (Table 6).

Table 5 PCI based GPU - Main characteristics

	PCI based architecture	Notes
Domain	Cloud, Edge	
Hosting	Standard Server	The hosting serves must be validated
Power consumption	300W – 70W	Possible candidates for edge sites are P4 [57] (75W) and T4 [58] (70W) (Figure 40).
Dimensions	169,53 mm x 68,90 mm	Low-Profile PCI Express Form Factor
Performance	P4: 5.5 TFLOPS; T4: 8,1 TFLOPS [59]	Single-Precision Performance (FP32)

Figure 40 T4 and P4 GPU

Table 6 GPU modules - Main characteristics

	SoC based architecture	Notes
Domain	Near Edge, Far Edge	
Hosting	Modules	Jetson Family [60] (Figure 40)
Power consumption	5W to 50W	
Dimensions	69.6x45 – 100x87 mm	Possible hosting on end devices.
Performance	0.5 TFLOPs to 32 TOPS[61]	from Nano to AGX Xavier

Nano	TX2 series	Xavier NX	AGX Xavier

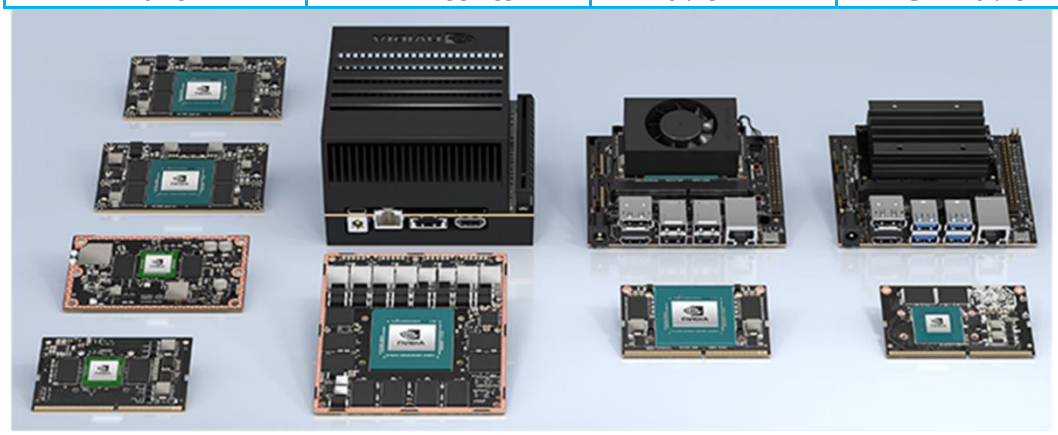


Figure 41 Jetson family modules

In addition, ATOS BullSequana Edge will be used to test and validate of the AI@EDGE AIFs within a commercial-ready edge node. ATOS BullSequana Edge is an edge computing server, with high storage and processing capacity, which aims to manage and process IoT environments where real time processing and security are essential. It can process IoT data, as well as analyze and run AI applications in real-time for tasks such as computer vision or immediate decision making. It also offers reduced dependency on data center and cloud connectivity and availability, ensuring that applications are not interrupted in the event of limited or intermittent network connectivity.

Figure 42 ATOS BullSequana Edge node

BullSequana Edge runs on a 16-core Intel Xeon CPU that can accommodate up to two Nvidia Tesla T4 GPUs or optional FPGA's which enable the inference of complex AI models right at the Edge with the lowest possible latency. In relation to the memory and storage capacity, BullSequana Edge uses a SSD of 480 GB or higher, and a memory of 32 GB RAM or higher.

BullSequana Edge is oriented to the following scenarios:

- Artificial Intelligence: ATOS Edge Computer Vision provides an advanced extraction and analysis of characteristics from people, faces, emotions, behaviors etc., to perform automatic actions based

on this analysis. In the case of video surveillance, it allows a large set of smart cameras to collaborate holistically in real time by monitoring operations without interruption.

- Big Data: ATOS Edge Data Analytics enables organizations to improve their business models with predictive and prescriptive solutions.
- Container: ATOS Edge Data Container (EDC) offers the all-in-one container solution in the place where data is managed and serves as a decentralized IT system. It can operate autonomously in non-data center environments, without the need for local on-site operations.

HW acceleration for access network link monitoring and reconfiguration at the edge

Besides the above generic HW acceleration, in the framework of UC2, AI@EDGE aims to integrate Smart-NIC at edge servers to support runtime monitoring of the network infrastructure close to endpoints, sensors and actuators, and attack mitigation against detected attacks. We aim at doing that at the hardware level, using programmable Smart-NICs (Network Interface Cards), in order to (i) satisfy both real-time monitoring of network flows, (ii) guarantee the physical security of network link metrics then used for anomaly and intrusion detection, and (iii) exploit the reconfigurability of the Smart-NIC to mitigate attacks by blocking flows through an SDN approach. The metrics computed at the Smart-NIC level are meant to fuel the anomaly detection AIF framework (described in D3.1), as well as more specific collaborative intrusion detection systems such as the Split-and-Merge CIDS from [62]. A relevant challenge in taking in charge the real-time computing and update of metrics profiling network flows is (i) the limited number of primitives and operations that can be pushed at the Smart-NIC level, namely using the NetFPGA boards as Smart-NIC hardware and the P4 language and P4-NetFPGA environment for the SDN operations, and (ii) the limited time and memory budget at the Smart-NIC to scale with the line rate.

The NetFPGA-SUME board has been designed to allow the research community to access affordable evaluation, experimentation and demonstration environments for 10 and 100 Gbps operations [63, 64, 65]. The board consists in a PCIe adapter card with an FPGA fabric manufactured by Digilent Inc. The main component is a Xilinx Virtex-7 690T FPGA, complemented by five subsystems as depicted in *Figure 43*. The memory subsystem is composed of a DRAM section that contains two SoDIMM slots, supporting modules for up to 8GB each, and a SRAM section that includes three QDR-II+ modules of 72Mb each. The PCIe 3.0 subsystem is used to communicate with the host machine, allowing both register access and packet transfer. The storage subsystem supports up to two external disks through SATA interfaces and a MicroSD card. The configuration and debug subsystem provides additional storage in the form of two NOR FLASH modules of 512Mb each, connected to the FPGA through a CPLD. This storage is intended for the FPGA's programming file. The high-speed serial interface subsystem is composed by 30 serial links connected to Virtex-7 GTH transceivers, supporting operations up to 13.1Gb/s. These links connect the FPGA to the various ports on the board, such as the Ethernet Interfaces and the PCIe subsystem.

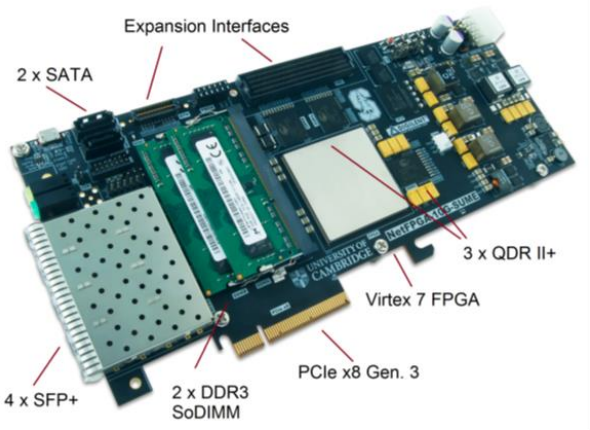


Figure 43 NetFPGA Sume Board [66]

7.2 SW Development of Accelerated AIFs on GPU & FPGA

The programming of very complex parallel architectures, such as those inside GPU and FPGA microchips, becomes very cumbersome when following the "classical" low-level optimization approach. Skilled programmers need to master parallelization techniques and low-level Hardware Description Languages, e.g., VHDL, which could require even months of designing-coding-debugging per AI function. Given the very short time-to-market targets and the proliferation of complex AIFs in the industry, it would become almost impractical to use HW accelerators in this "classical" way. To tackle this challenge, the vendors of FPGA and GPU devices are also providing high-level programming tools/frameworks for their devices, which allow ordinary SW developers and data scientists to exploit the underlying HW without delving into the low-level details of each device. Furthermore, these high-level tools/frameworks get constantly updated by the same vendors struggling to support the most recent techniques in AI/ML and to stay compatible with the most widely used AI/ML tools in the community (e.g., TensorFlow and PyTorch). Given the context and potential market of AI@EDGE, we adopt in the project such a tool-oriented and high-level development approach for accelerating AIFs, instead of performing time-consuming low-level coding & optimization (which would be more suitable, e.g., in a project involving smaller consolidated DSP functions and long-term development cycles). Furthermore, a considerable amount of development effort will be devoted to integrating the aforementioned acceleration tools to the Connect-Compute platform and to facilitate the future use of acceleration in the Connect-Compute platform, instead of merely populating a extensive AIF catalogue only with today's functions. The following two subsections describe more details regarding the tools selected for developing accelerators on GPU and FPGA for AI@EDGE, while the last subsection summarizes the constraints while doing so.

Besides developing accelerators, the second aspect of SW on GPU/FPGA regards the integration to the Connect-Compute platform, how ordinary users can exploit the already developed AIFs in the catalogue, and what acceleration factors should be expected at the end. The approach adopted in AI@EDGE for this purpose is explained in the third subsection below.

7.2.1 Developer tool-flow for FPGA

Employing high level approaches for FPGA design can enable a faster and more flexible development process compared to RTL, especially for complex DNN (Deep Neural Networks). AI@EDGE will leverage High Level Synthesis (HLS) tools as well as high level tools for AI inference in order to design accelerated

kernels for the computationally intensive tasks of the use cases. There exist multiple vendors that provide HLS tools for their own FPGA platforms, the most important of which are Xilinx and Intel. Intel (formerly Altera) with the Intel HLS Compiler which is included in the Intel Quartus Prime Design Software installation aims to optimise, verify, and simulate FPGA designs. Xilinx provides the Vivado HLS tool which is a compiler that enables C, C++ and SystemC programs to be directly targeted into Xilinx devices. Also, lately Xilinx provided a unified software platform called Vitis which comes with its own compiler and includes support for many accelerator cards, embedded platforms or FPGA instances in the cloud. Lastly, it is worth mentioning that these two companies have launched in 2019-2020 a new family of devices which feature AI-specific engines designed specifically for AI inference through an automated process of compiling DNN models to the device optimized blocks. Xilinx introduced the Versal AI Core family that enables adaptive, domain specific architectures while Intel introduced the Stratix 10 NX FPGAs. Xilinx also provides Vitis-AI which is a development stack for hardware-accelerated AI inference for these devices and previous FPGA families as well (such as Xilinx Alveo).

AI@EDGE will utilize high level approaches for designing accelerators specifically for Xilinx devices. The rationale behind this decision is the wide support of Xilinx devices and software, the quality and quantity of online resources, hardware capabilities and future prospects. Below we will analyze the design process using High Level Synthesis with Vitis as well as AI inference acceleration with Vitis-AI.

In terms of code, by adding different directives on a C/C++ or OpenCL file, users are able to direct the HLS compiler to synthesize kernels for FPGA. More specifically, through an Eclipse-based framework, ICCS will develop the kernels with the Vitis framework through a unified OpenCL interface for programming edge and cloud Xilinx FPGAs. The kernel acceleration process involves HLS pragmas being applied in the kernel C/C++ code (e.g loop unrolling, pipelining, etc.) in order to guide the HLS compiler to synthesize the code for the FPGA fabric efficiently. The process is illustrated in *Figure 44*:

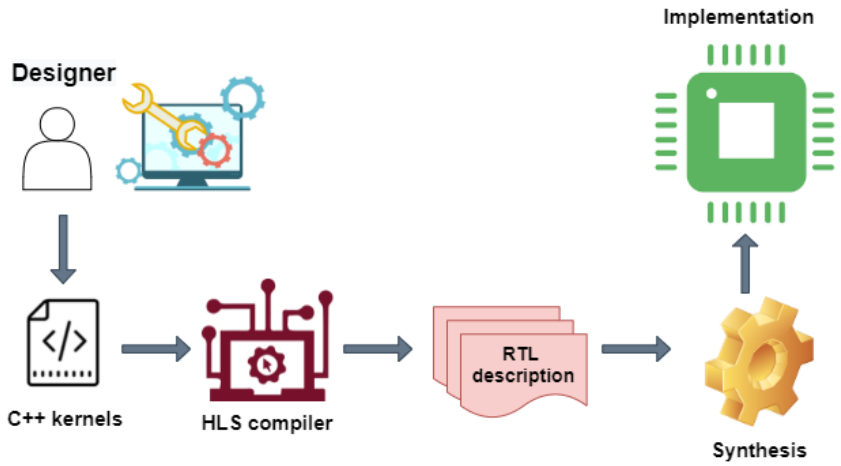


Figure 44 Illustration of HLS design process for FPGA using the Xilinx Vitis HLS compiler

The aforementioned hierarchy of abstraction layers of the FPGA design process includes even higher level layers such as automated compilers for accelerated DNN inference like Vitis AI. Vitis AI consists of optimized IPs, support for many deep learning frameworks (such as PyTorch or Tensorflow) and many DNN models (such as CNNs, MLPs, etc). It is designed with high efficiency and ease-of-use in mind supporting both traditional Xilinx FPGAs as well as Xilinx ACAPs that feature AI engines. First, ICCS with Vitis-AI optimizer will perform DNN model compression, reducing the computational complexity or

memory bandwidth by more than 4x with minimal accuracy impact. Along with Vitis-AI Quantizer, the AI model is converted from 32-bit floating-point weights and activations to fixed-point like INT8. FP32 arithmetic is native for CPU or GPU architectures while FPGAs can leverage INT8 arithmetic more efficiently. Consequently, the model requires less memory for the parameters, hence being faster and more efficient than FP32 equivalents. Lastly, with Vitis-AI compiler, the AI model is converted to an optimized instruction set and data flow in order to run on Xilinx Deep Learning Processing Unit (DPU) if the target platform is one of the Xilinx MPsoCs or Alveo, or on AI-engines if the target platform is one of the Versal AI Core Series platforms (such as VCK190). It is worth mentioning that all the flow works transparently also for MPSoCs, Alveos and VCK190 boards. Figure 45 shows an illustration of the Vitis-AI framework using a block diagram.

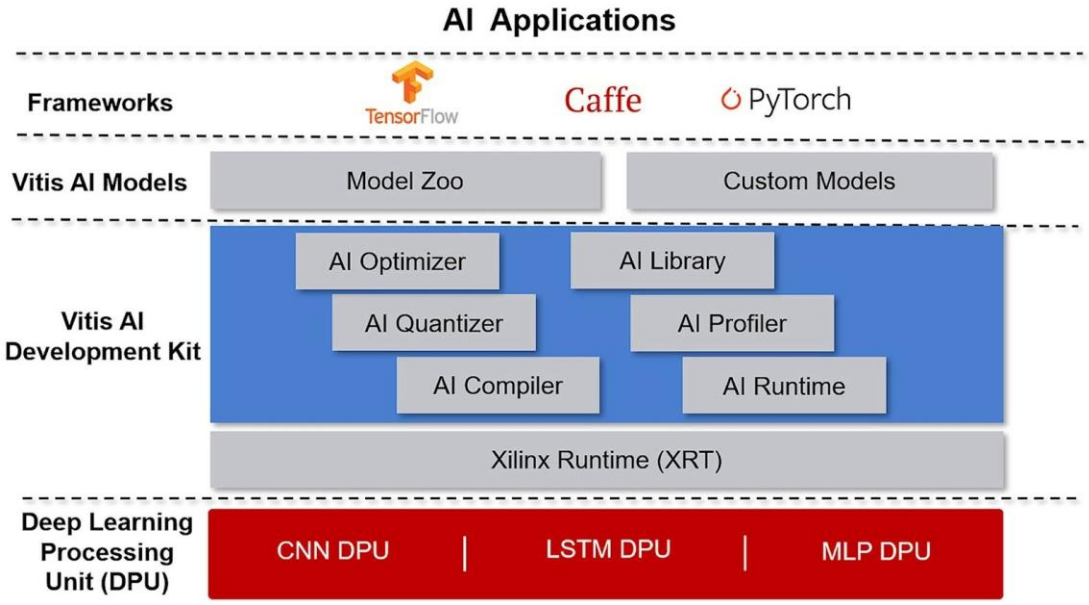


Figure 45 Illustration of Vitis-AI framework for FPGA using a block diagram [source: Xilinx]

Programming of NetFPGA accelerators

The P4-NetFPGA project [67, 68] offers a development environment based on the Xilinx P4-SDNet toolchain [69] and the NetFPGA SUME open-source code base. The goal is to make it easier to program using P4 without needing to use a Hardware Description Language.

The P4 architecture model currently defined for the NetFPGA SUME is the SimpleSumeSwitch, consisting of just a parser, a match-action pipeline and a deparser, as shown in *Figure 46*. We are considering shifting to the SUME Event Switch, a new architecture model recently developed in the P4-NetFPGA project. The SUME Event Switch offers better flexibility regarding the events that trigger the pipeline to execute. A significant advantage is that it allows the generation of custom packets, a feature that can be useful to send information to the control plane.

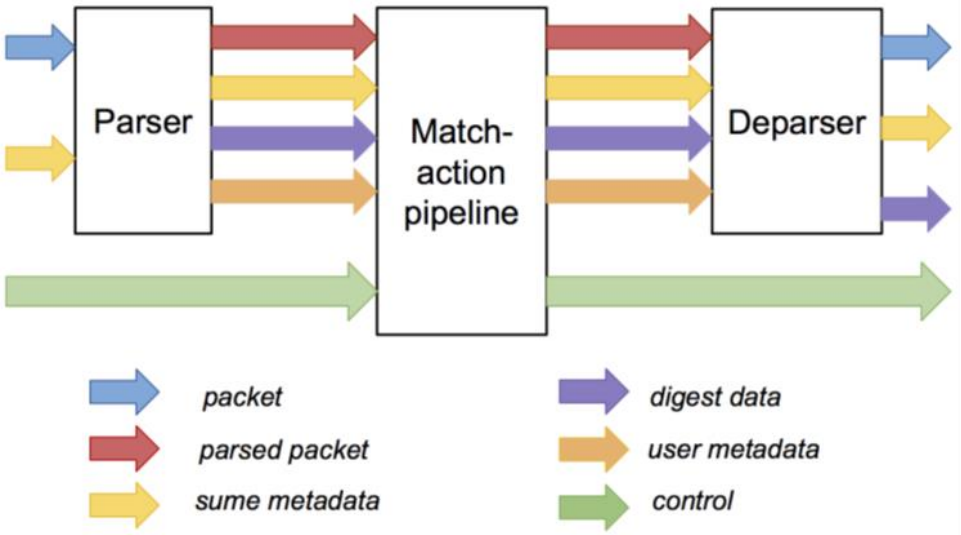


Figure 46 Block diagram of the SimpleSumeSwitch P4 architecture

The design is done according to the P4-NetFPGA workflow. First, the user writes a P4 program, a text file for populating the tables during the simulations and a python file used to generate the test data through the Scapy module [70, 71]. The P4-SDNet compiler then generates the resulting HDL instance of the SimpleSumeSwitch and an initial simulation framework, enabling the user to run a test-bench. The HDL is then wrapped and installed in the NetFPGA Reference Switch, replacing the output port lookup module. The second round of testing can now be made to verify that the design has been correctly integrated. After that, the bit-stream can be generated and the FPGA programmed. Finally, the design can be tested on real hardware through a command-line interface generated by the workflow.

In order to address some P4 limitations, such as lack of support for stateful operations, we need to use some target-specific HDL libraries, called extern functions, to implement custom logic within the P4 program.

7.2.2 Developer tool-flow for GPU

NVIDIA provide free access to the CUDA [72] Toolkit that enables developers to build NVIDIA GPU accelerated compute applications. It consists of the CUDA compiler toolchain including the CUDA runtime (cudart) and various CUDA libraries and tools. It provides a development environment for creating GPU-accelerated applications and can be downloaded from the NVIDIA website [73] The toolkit can be used on GPU-accelerated embedded systems, desktop workstations, enterprise data centers, cloud-based platforms and HPC supercomputers in order to build a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. CUDA is designed to support various languages and application programming interfaces, as depicted in Figure 47, as well different architecture and operating systems, as reported in Figure 48.

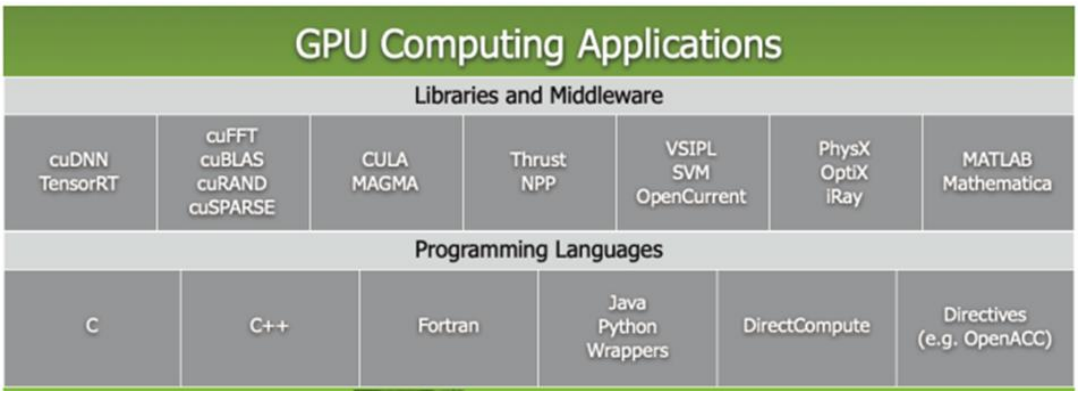


Figure 47 CUDA structure

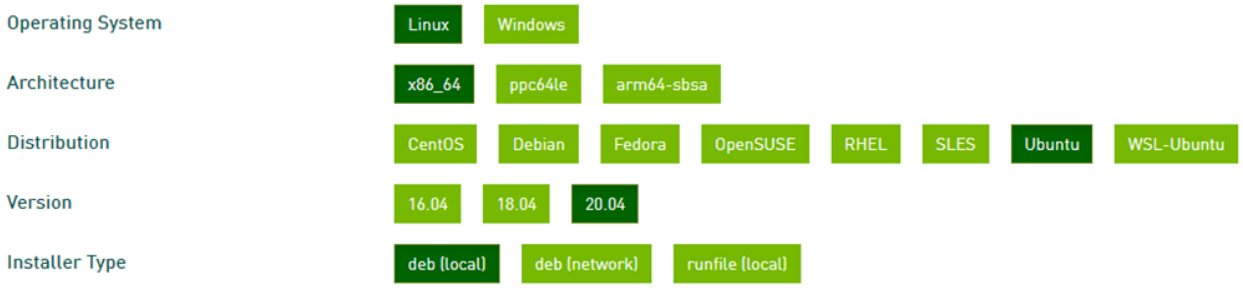


Figure 48 CPU and Operating System support

Figure 49 shows two possible deployments of GPU computing application (hosted in containers) on top of a CPU/GPU based architecture.

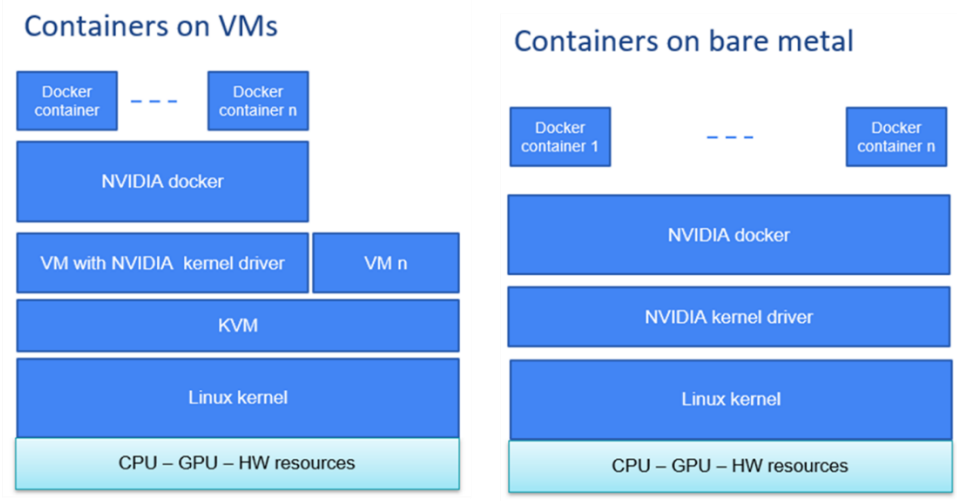


Figure 49 Application deployment scenarios

GPU support for AI applications

NVIDIA support AI application development over its GPUs providing CUDA-X, *Figure 50*, built on top of CUDA. It is a collection of libraries, tools, and technologies across multiple application domains, from artificial intelligence (AI) to high performance computing (HPC). NVIDIA libraries can run on resource constrained devices as well as on powerful devices hosted in cloud.

CUDA-X provides:

- a complete deep learning software stack [74] including AI libraries (Table 7);
- support for TensorFlow, PyTorch, MXNet, and other tools;
- support for single GPUs and multi-GPUs environments.

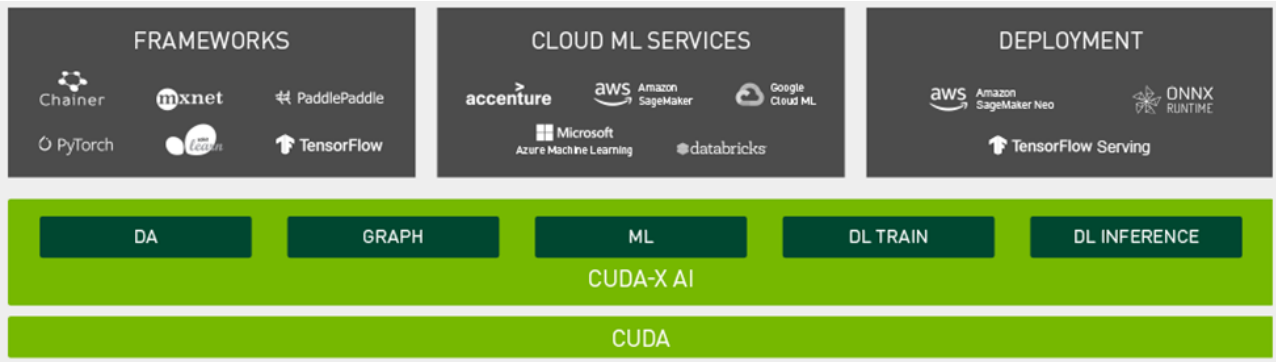


Figure 50 CUDA_X

Table 7 CUDA-X Deep Learning libraries

Library	Description	Documentation
NVIDIA cuDNN	GPU-accelerated library of primitives for deep neural networks.	https://developer.nvidia.com/cudnn
NVIDIA TensorRT	High-performance deep learning inference optimizer and runtime for production deployment.	https://developer.nvidia.com/tensorrt
NVIDIA Riva	Platform for developing engaging and contextual AI-powered conversation apps.	https://developer.nvidia.com/riva
NVIDIA DeepStream SDK	Real-time streaming analytics toolkit for AI-based video understanding and multi-sensor processing.	https://developer.nvidia.com/deepstream-sdk
NVIDIA DALI	Portable, open-source library for decoding and augmenting images and videos to accelerate deep learning applications.	https://developer.nvidia.com/DALI

The NVIDIA NGC catalogue [75] provides pre-trained models, training scripts, optimized framework containers and inference engines for popular deep learning models. NVIDIA AI Toolkit includes libraries

for transfer learning, fine tuning, optimizing, and deploying pre-trained models across a broad set of industries and AI workloads.

7.2.3 User Exploitation of accelerated AIFs

After developing accelerated AIFs as explained above, the users of AI@EDGE will be able to exploit them seamlessly via the integration techniques of the Connect-Compute platform. The developer will put each AIF in its own docker container, almost as if it was an ordinary SW AIF, to be managed by Kubernetes with only a few additional modifications. Kubernetes is a widely adopted container orchestrator that already provides abstractions for resource management and simplifies distributed application deployment and management (see Section 3). In the ICCS acceleration testbed described above, we use device passthrough, a technology that allows the Linux kernel to directly present an internal PCI GPU/FPGA to a VM, in order to make the accelerator cards available to any VM and by extension to Kubernetes built on top of them. We utilized the device plugin framework, and the respective implementations from NVIDIA [76] and Xilinx [77] to advertise system hardware resources to the Kubelet, the node-agent that registers each node to the Kubernetes' API. Device plugins deploy a DaemonSet to the nodes with the specified accelerator. Using the **kubectl** CLI, we can get information about the pods, (the smallest unit of deployment in Kubernetes) responsible for accelerators registration to the Kubernetes API. Those pods are usually deployed in the **kube-system** namespace. Furthermore, the Kubernetes API is aware of the existence and the availability of those resources and can (de)allocate them depending on the requests of the incoming pods/applications.

The user can create/delete/configure resources through the Kube-API, which is usually accessed through the **kubectl** tool via the command line. We define an application (pod) through a configuration YAML file, as shown below, with an example GPU-enabled application: we include the resource **nvidia.com/gpu: 1**; thus Kubernetes scheduler will place this application on a node that has this resource available. In addition, Kubernetes API is now aware of the allocation of this resource, and since multi-tenancy in accelerator cards is not supported by default on Kubernetes, it will not allow other GPU-enabled containers to be placed in that node, unless the resource becomes available again.

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
    - name: digits-container
      image: nvcr.io/nvidia/digits:20.12-tensorflow-py3
      resources:
        limits:
          nvidia.com/gpu: 1 # requesting 1 GPUs
```

The accelerated AIF will be placed inside a container. When deployed directly via Docker, this container must become "privileged" with access to all devices or limited access to a specific device by using the *--device* flag (with their run command). Specifically, for Xilinx Alveo FPGAs, they have a management function and a user function to specify a device that will be accessible within a container, which can be found via commands like

```
/opt/xilinx/xrt/bin/xbmgmt scan
/opt/xilinx/xrt/bin/xbutil scan
```

to specify a device that will be accessible within a container. To execute the container, we then issue:
docker run --rm -it --device=/dev/xclmgmt[mgmt]:/dev/xclmgmt[mgmt] --device=/dev/dri/renderD[user]:/dev/dri/renderD[user] [TAG]

Similarly, via Kubernetes, for the deployment of an AIF on a Xilinx Alveo FPGA, by utilizing the respective device plugin, we are able to assign the task to the node that has this FPGA resource available. For the case of an Alveo U280 FPGA, we only need to request the resource:
xilinx.com/fpga-xilinx_u280_xdma_201920_3-1579649056: 1 in the YAML configuration file.

```
apiVersion:                                                                    v1
kind:                                                                         Pod
metadata:
  name:                                                                  lstm-pod
spec:
  restartPolicy:                                                            Never
  containers:
  -                                  name:                          alveo-container
    image:                                                         iwita/lstm_fpga
    imagePullPolicy:                                                  IfNotPresent
    resources:
      limits:
        xilinx.com/fpga-xilinx_u280_xdma_201920_3-1579649056:                     1
    command:                                                     ["./lstm_rom"]
    args: ["krnl_lstm.xclbin","10"]
```

For FPGA accelerators, the aforementioned Docker container image has an isolated runtime environment with pre-installed the Xilinx Runtime library (XRT) facilitating the communication between the application's SW code and the accelerated kernel, the SW executable for the host part of the accelerator, the *xclbin* file with the kernel's bitstream, and other dependencies that may be needed for the application at hand. The Docker container cannot access the host kernel of the system that runs the container, therefore there is the need to have the same XRT version installed on the system as a driver and use the XRT inside the container as runtime. Also, the FPGA should be flashed with a pre-specified Xilinx Shell, based on the selected XRT. We note that, when building the aforementioned container, the developer has specified the OS that is going to be used inside the container, downloaded and installed the selected version of the XRT library, copied the files required for the FPGA accelerated application, and prepared the environment variables inside the container so that everything will be ready after invoking the container. The server or host PC needs only the accelerator card (flashed with the predetermined Xilinx shell) and the XRT (same version as inside the container). We also note that, in order to apply accelerator-aware AIF placement in AI@EDGE, it might be needed to have multiple docker images prepared per AIF and uploaded in a shared registry, with each image built for a pre-defined combination of Alveo+XRT version; in such an approach, the operators will automatically edit the YAML file to request the required accelerator resource from Kubernetes API.

For GPU accelerators, the NVIDIA Container Toolkit [78] allows users to build and run GPU accelerated containers. In addition, it includes a container runtime library [79] and utilities to configure containers to leverage NVIDIA GPUs automatically. *Figure 51* shows the flow through the various components.

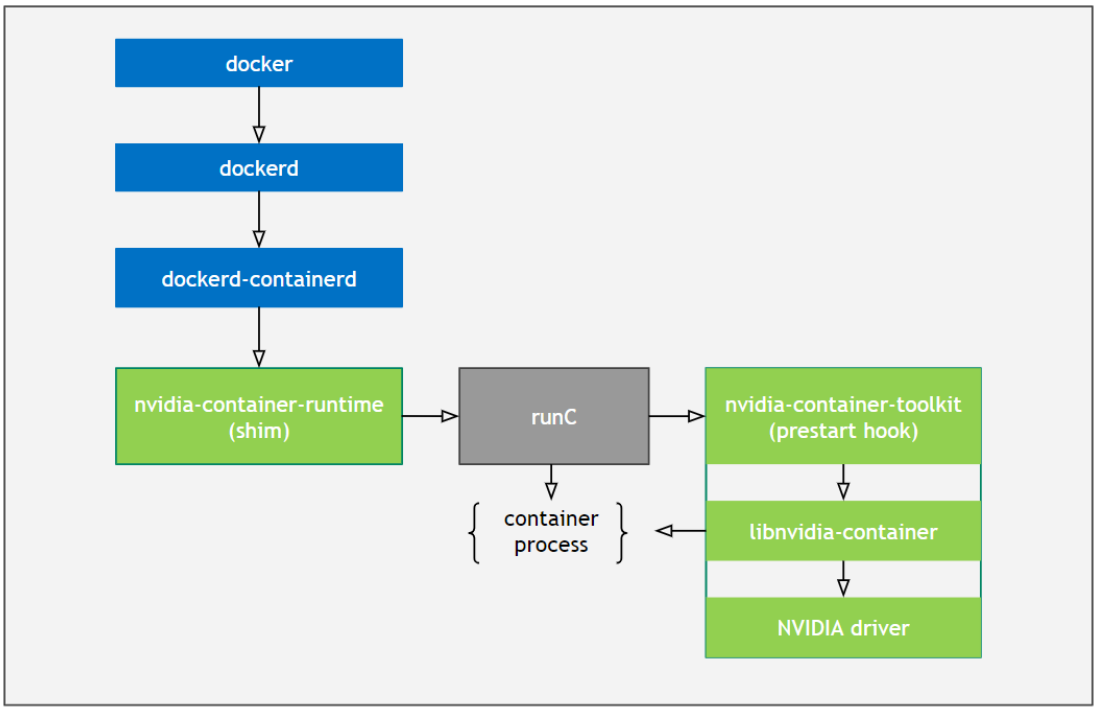


Figure 51 Running a container on GPUs

7.2.4 Accelerated AIF constraints and acceleration server requirements

In principle, any AI/ML function implemented by software can be accelerated on FPGA or GPU. However, for a considerable number of cases, the acceleration can prove impractical due to development time, implementation inefficiency (of certain AIF on specific computational models), or other HW/SW peculiarities. Given the aforementioned choices on HW devices and SW tools in AI@EDGE, we collect here a number of constraints regarding the AIF that are able to be accelerated during AI@EDGE.

First, with respect to FPGA, the **Vitis/HLS approach** allows for any AIF acceleration originally written in C/C++ (otherwise requires language conversion), but **is not recommended for** the following:

- training (mostly due to the development effort, extensive floating-point arithmetic, limited onchip memory) (overall GPUs are preferred for this task)
- large ANN networks (due to device memory size and inefficient partitioning of computation as a result), e.g., more than 10MB-weights ANN becomes challenging (order of magnitude)
- function recursion, dynamic memory allocation in AIF code (not supported by toolflows)
- general pointer casting (HLS supports only native C/C++ types of pointers)
- increased SW/function/branching complexity (due to inefficient mapping onto FPGA resources)
- many third-party library dependencies (due to development effort required for porting to FPGA) (only common libraries available now by Xilinx, e.g., math.h)

We note that, although quicker performance compared to "classical" HDL development, the HLS development still remains considerably high-level (e.g., compared to VitisAI or Python), especially when targeting a certain optimization level involving a lot of design space exploration. Hence, the HLS imposes constraints also with respect to time/budget and only a limited number of AIFs will be developed via HLS during AI@EDGE. The most prominent example is the LSTM-based structure, which has already consumed development effort in the order of 2 person-months per AIF (from scratch).

Second, also with respect to FPGA, the **VitisAI approach** starts from Python code (i.e., TensorFlow or PyTorch) and **supports only specific** sizes and kinds of layers, i.e., it can accelerate [80]:

vai_q_tensorflow2 Supported Operations and APIs

The following table lists the supported operations and APIs for vai_q_tensorflow2.

Table 17: **vai_q_tensorflow2 Supported Layers**

Layer Types	Supported Layers	Description
Core	tf.keras.layers.InputLayer	
Core	tf.keras.layers.Dense	
Core	tf.keras.layers.Activation	If 'activation' is 'relu' or 'linear', will be quantized. If 'activation' is 'sigmoid' or 'swish', will be converted to hard-sigmoid or hard-swish and then be quantized. Otherwise will not be quantized.
Convolution	tf.keras.layers.Conv2D	
Convolution	tf.keras.layers.DepthwiseConv2D	
Convolution	tf.keras.layers.Conv2DTranspose	
Pooling	tf.keras.layers.AveragePooling2D	
Pooling	tf.keras.layers.MaxPooling2D	
Pooling	tf.keras.layers.GlobalAveragePooling	
Normalization	tf.keras.layers.BatchNormalization	By default, BatchNormalization layers are fused with the previous convolution layers. If they cannot be fused, they are converted to depthwise convolutions. In the QAT mode, BatchNormalization layers are pseudo fused if train_with_bn is set to TRUE. They are fused when the get_deploy_model function is called.
Regularization	tf.keras.layers.Dropout	By default, the dropout layers are removed. In the QAT mode, dropout layers are retained if remove_dropout is set FALSE. It is removed when the get_deploy_model function is called.
Reshaping	tf.keras.layers.Reshape	
Reshaping	tf.keras.layers.Flatten	
Reshaping	tf.keras.UpSampling2D	
Reshaping	tf.keras.ZeroPadding2D	
Merging	tf.keras.layers.Concatenate	
Merging	tf.keras.layers.Add	
Merging	tf.keras.layers.Muliply	
Activation	tf.keras.layers.ReLU	
Activation	tf.keras.layers.Softmax	The input for the Softmax layer is quantized. It can run on the standalone Softmax IP for acceleration.
Activation	tf.keras.layers.LeakyReLU	Only 'alpha'=0.1 is supported on the DPU. For other values, the model is not quantized and mapped to the CPU.

Figure 52 Tensorflow2 APIs

We note that the VitisAI supported AIF should be coded in specific versions, as of December 2021: TensorFlow 1.x (v.1.15 without fast fine-tuning), TensorFlow 2.x (v.2.3), PyTorch (v.1.2--1.7.1).

Furthermore, we note that VitisAI is used exclusively for DNN inference (not training), while unsupported ANN layers might result in inefficient partitioning/acceleration (they will be executed on the CPU). In contrast to Vitis/HLS, the VitisAI supports large DNN structures by default, by following its own off-chip memory scheme.

To conclude, regarding **FPGA acceleration in AI@EDGE**, the AIFs to be accelerated should:

- be only for inference
- have a straightforward and widely-used DNN structure (e.g., supported by most popular DNN frameworks), not very large, and be coded on specific Python versions (see above for VitisAI)
- in certain exceptions, for AI/ML algorithms, be developed in C/C++ (see above for Vitis/HLS)

Regarding the **Server Requirements,** users of accelerated AIFs must have:

- *HW*: PCIe Gen3x16 cards (dual-slot), 225W extra power, >16 GB RAM (64-80GB preferred)
- *SW*: K8s version >= 1.17 (1.10 for GPU), K8s plugins for FPGA+GPU, Docker latest version (>= 19.03 preferred), drivers from NVIDIA (~384.81), XRT runtime from Xilinx (latest version, few MBs), Linux kernel 3.10+ 64-bit, GCC with C++14 features. No license needed.

8. Next Steps

During the first year of activity, the effort of WP4 was focused on designing the Connect Compute Platform main components and defining the relationships between them.

In this report, the AI@EDGE connect-compute platform (CCP) design activity results have been introduced. The CCP was presented as an extension of ETSI MEC and ETSI MEC in NFV architectures for multi-site and distributed environments. Based on the evaluation of benefits and drawbacks of both architectural visions, the Consortium will proceed with the integration of the components and the software artefacts in a way that aligns best with the system requirements of the CCP, as detailed in D2.1. The integration work will bring to an extended ETSI MEC/NFV based architecture with the inclusion of applications and models capable of providing the AI@EDGE platform with the context and metadata necessary to take automatically actionable decisions and to realize intelligent data and computation offload control and management of applications and services deployed over the decentralized and distributed AI@EDGE platform.

A roadmap for the integration of the Connect Compute Platform is divided into four main phases and has been defined in detail in Section 2.4. Broadly speaking, the initial phases one and two focus on using 4G/5G NSA networks, with the SGW+PGW at the MEC hosts. Later in the project, during phases three and four, 5G SA will be introduced, with SMF + UPF deployed at the MEC hosts.

The integration of each component in the platform is tested in the Integration Testbed as soon as it is available.

In parallel with the design activity related to the overall CPP architecture, the Consortium started to investigate solutions for implementing the various components of the system such as the integration of serverless platforms and the definition of support for AIFs application provisioning. Also, different hardware acceleration solutions (FPGA, GPU, CPU) to be employed at the Edge have been investigated, and dual-connectivity monolithic RANs have been analysed and compared with cross-layer multi-connectivity disaggregated RANs to see if dynamically adapting the network topology to the network conditions.

The next steps will follow the roadmap defined in Section 2.3 and will see the finalization of the 4G/5G NSA in the integration environment and the preparation for the 5G SA phase. In the next months. a consolidated design of the CCP is expected. As the development of the CCP components gets more mature, they will be gradually integrated into the Integration Testbed for initial validation. Following the CCP evolutionary roadmap, the serverless and function as service functionalities will be enhanced and integrated into the platform more tightly. MEC applications are expected to be deployed as required. A first multi-connectivity solution will be integrated and validated in the testbed. Preparation activities will also take place for migrating the Integration Testbed from 4G/5G NSA to 5G SA.

Bibliography

[1] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification", ETSI, GS NFV-IFA 013, V2.1.1, October, 2016.

[2] ETSI, “Multi-access Edge Computing (MEC); Framework and Reference Architecture”, ETSI GS MEC 003 V2.2.1 (2020-12)

[3] M. Mena, A. Papageorgiou, L. Ochoa-Aday, M. S. Siddiqui, y G. Baldoni, “Enhancing the performance of 5G slicing operations via multi-tier orchestration”, feb. 2020. Doi: 10.1109/ICIN48450.2020.9059546.

[4] Roberto Riggio, Bengt Ahlgren. D3.1 Initial report of systems and methods for AI@EDGE platform automation. AI@EDGE Deliverable 3.1 (H2020-ICT-52-2020)

[5] LightEdge Wiki. https://github.com/lightedge/lightedge.github.io/wiki (accessed on nov. 23, 2021).

[6] S. Dahmen-Lhuissier, “ETSI – Open Source Mano | Open Source Solutions | Mano NFV”, ETSI. https://www.etsi.org/technologies/open-source-mano (accessed on nov. 22, 2021).

[7] “OSM-MR9 Hackfest – OSM Public Wiki”. https://osm.etsi.org/wikipub/index.php/OSM-MR9_Hackfest (accessed on nov. 22, 2021).

[8] https://www.ettus.com/wp-content/uploads/2019/01/b200-b210_spec_sheet.pdf

[9] https://www.ettus.com/wp-content/uploads/2019/01/X300_X310_Spec_Sheet.pdf

[10] LightEdge website https://lightedge.io/

[11] Estefanía Coronado, Zarrar Yousaf, and Roberto Riggio, “LightEdge: Mapping the Evolution of MultiAccess Edge Computing in Cellular Networks” Comm. Magazine April 2020

[12] Nuclio website https://nuclio.io/

[13] CNCF WG-Serverless Whitepaper v1.0

[14] Nuclio website https://nuclio.io/

[15] O-RAN Alliance, “O-RAN Non-RT RIC: Functional Architecture v1.01”, March 2021

[16] O-RAN Alliance, “O-RAN A1 interface: Application Protocol v3.01”, March 2021

[17] https://wiki.o-ran-sc.org/display/GS/Near+Realtime+RIC+Installation

[18] https://accelleran.com/drax-2/

[19] Bonati, L., D'Oro, S., Polese, M., Basagni, S. and Melodia, T., 2021. Intelligence and learning in O-RAN for data-driven NextG cellular networks. IEEE Communications Magazine, 59(10), pp.21-27. Bonati, L., D'Oro, S., Polese, M., Basagni, S. and Melodia, T., 2021. Intelligence and learning in O-RAN for data-driven NextG cellular networks. IEEE Communications Magazine, 59(10), pp.21-27.

[20] Schmidt, R., Irazabal, M. and Nikaein, N., 2021, December. FlexRIC: an SDK for next-generation SD-RANs. In Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies (pp. 411-425).

[21] O-RAN Alliance, “O-RAN A1 interface: Application Protocol v3.01”, March 2021

[22] Conrado, E, Riggio, R. et al, 2019, 5G-EmPOWER: A software-defined networking platform for 5G radio access network. IEEE Transactions on Network and Service Management PP(99):1-1, DOI:10.1109/TNSM.2019.2908675

[23] Tunability: Importance of Hyperparameters of Machine Learning Algorithms

[24] A Survey of the Recent Architectures of Deep Convolutional Neural Networks

[25] Deep learning: yesterday, today, and tomorrow

[26] Snoek et al., 2012

[27] Rasmussen and Williams, 2006

[28] Hutter et al., 2011

[29] Breiman, 2001

[30] Hutter et al., 2014

[31] Bergstra et al., 2011

[32] Meta-Learning: A Survey, Joaquin Vanschoren

[33] Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization

[34] BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters

[35] Meta-Learning: A Survey, Joaquin Vanschoren

[36] Metalearning: Applications to Data Mining. Brazdil et al., 2009

[37] Ho and Basu, 2002; Orriols-Puig et al., 2010

[38] Alleg, Abdelhamid, et al. "Delay-aware VNF placement and chaining based on a flexible resource allocation approach." 2017 13th international conference on network and service management (CNSM). ieee, 2017.

[39] Zhang, Qixia, Fangming Liu, and Chaobing Zeng. "Adaptive interference-aware VNF placement for service-customized 5G network slices." IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, 2019.

[40] Yang, Song, et al. "Delay-sensitive and availability-aware virtual network function scheduling for NFV." IEEE Transactions on Services Computing (2019).

[41] Behravesh, Rasoul, et al. "Joint user association and VNF placement for Latency sensitive applications in 5G networks." 2019 IEEE 8th International Conference on Cloud Networking (CloudNet). IEEE, 2019.

[42] Harutyunyan, Davit, et al. "Latency and mobility-aware service function chain placement in 5g networks." IEEE Transactions on Mobile Computing (2020).

[43] Behravesh, Rasoul, et al. "Time-Sensitive Mobile User Association and SFC Placement in MEC-Enabled 5G Networks." IEEE Transactions on Network and Service Management (2021).

[44] ETSI GS MEC 010-2 V2.1.1 (2019-11). Multi-access Edge Computing (MEC); MEC Management; Part 2: Application lifecycle, rules and requirements management.

[45] *Mobile Edge Computing (MEC); Mobile Edge Management; Part 2: Application lifecycle, rules and requirements management, ETSI Group Specification MEC 010*

[46] Ref. https://helm.sh

[47] ETSI GS MEC 010

[48] ETSI GS MEC 010-2

[49] ETSI MEC 010-2

[50] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," in RFC 6824, 2013

[51] 3GPP TS 23.501. System architecture for the 5G system.

[52] 3GPP TS 23.501

[53] S. Ferlin, Ö. Alay, O. Mehani, R. Boreli. (2016). BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. IFIP Networking Conference (IFIP Networking) and Workshops.

[54] F. H. Mirani, N. Boukhatem, M. A. Tran. (2010). A Data-Scheduling Mechanism for Multi-Homed Mobile Terminals with Disparate Link Latencies. IEEE 72nd Vehicular Technology Conference.

[55] Kaiping Xue, Jiangping Han, Dan Ni, Wenjia Wei, Ying Cai, Qing Xu, Peilin Hong. (2018). DPSAF: Forward Prediction Based Dynamic Packet Scheduling and Adjusting With Feedback for Multipath TCP in Lossy Heterogeneous Networks. IEEE Transactions on Vehicular Technology, 67, 1521-1534.

[56] A. M. Ahmed, A. Patel, M. Z. A. Khan. (2021). Reliability Enhancement by PDCP Duplication and Combining for Next Generation Networks. EEE 93rd Vehicular Technology Conference (VTC2021-Spring).

[57] Please see: NVIDIA Tesla P4 GPU Datasheet

[58] Please see: t4-tensor-core-datasheet-951643.pdf (nvidia.com)

[59] TFLOPS: one trillion floating-point operations per second

[60] Please see: Jetson Modules | NVIDIA Developer

[61] TOPS: one trillion operations per second

[62] Agathe Blaise et al. “Detection of zero-day attacks: An unsupervised port-based approach”. In: Computer Networks 180 (2020).

[63] NetFPGA. URL: https://netfpga.org/.

[64] NetFPGA-SUME. URL: https://github.com/NetFPGA/NetFPGA-SUMEpublic/wiki.

[65] Noa Zilberman et al. “NetFPGA SUME: Toward Research Commodity 100Gb/s”.

[66] NetFPGA-SUME Reference Manual. URL: https://digilent.com/reference/programmable-logic/netfpga-sume/reference-manual.

[67] Stephen Ibanez et al. “The p4-> netfpga workflow for line-rate packet processing”. In: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

[68] P4-NetFPGA. URL: https://github.com/NetFPGA/P4-NetFPGA-public/wiki.

[69] Xilinx. P4-SDNet User Guide. 2018. URL: https://www.xilinx.com/support/documentation/sw _ manuals/xilinx2017_4/ug1252-p4-dnet.pdf.

[70] Philippe Biondi. Scapy. 2018. URL: https://scapy.net/.

[71] Welcome to Scapy’s documentation! 2021. URL: https://scapy.readthedocs.io/en/latest.

[72] Please see: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[73] Please see: https://developer.nvidia.com/cuda-toolkit

[74] Please see: https://developer.nvidia.com/deep-learning-software

[75] Please see: https://ngc.nvidia.com/catalog

[76] Please see: https://github.com/NVIDIA/k8s-device-plugin

[77] Please see: https://github.com/Xilinx/FPGA_as_a_Service.git

[78] Please see: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/overview.html

[79] Please see: https://github.com/NVIDIA/libnvidia-container

[80] UG1414 (v1.4), “Vitis AI User Guide”, Xilinx Inc., July 22, 2021, https://www.xilinx.com/support/documentation/sw_manuals/vitis_ai/1_4/ug1414-vitis-ai.pdf