NANCY

An Artificial <u>In</u>telligent <u>A</u>ided Unified <u>N</u>etwork for Se<u>c</u>ure Be<u>y</u>ond 5G Long Term Evolution

[GA: 101096456]

Deliverable 4.1

Computational Offloading and User-centric Caching

Programme: HORIZON-JU-SNS-2022-STREAM-A-01-06

Start Date: 01 January 2023

Duration: 36 Months

Co-funded by the European Union

6GSNS

NANCY project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101096456.

Document Control Page

Deliverable Name	Computational Offloading and User-centric Caching
Deliverable Number	4.1
Work Package	WP4
Associated Task	T4.1 Computational Offloading & User-centric Caching Functionalities
Dissemination Level	Public
Due Date	30 September 2024 (M21)
Completion Date	30 September 2024
Submission Date	30 September 2024
Deliverable Lead Partner	UMU
Deliverable Author(s)	Ramon Sanchez-Iborra, Rodrigo Asensio-Garriga, Gonzalo Alarcón Hellín (UMU), Daniel Casini, Luca Abeni, Alessandro Biondi, Mauro Marinoni (SSS), Tziouvaras Athanasios (Bi2S), Antonella Clavenna (ITL), Maria Belesioti (OTE), Javier de Vicente, Michele Massetti (NEC), Anna Panagapoulou, Alvise Rigo (OTE), Carolina Fortuna, Blaz Bertalanic, Shih-Kai Chou (IJS), Konstantinos Kaltakis, Alexandros Dimos (INNO), Georgios Katsikas (UBI), Grigoris Kalogiannis (DRAXIS), Hatim Chergui (I2CAT), Cristina Regueiro (TEC), Marco Tambasco (TEI), David Franco (EHU), Athanasios Liatifis (UOWM), Dimitrios Pliatsios (UOWM), Panagiotis Sarigiannidis (UOWM), Thomas Lagkas (UOWM), Sotirios Tegos (UOWM)
Version	1.0

Document History

Version	Date	Change History	Author(s)	Organisation
0.1	8/3/2024	Section 1 and Section 3.1	Ramon Sanchez, Rodrigo Asensio	UMU
0.2	30/05/2024	Section 2 and Section 3	Gonzalo Alarcón, Ramon Sanchez, Rodrigo Asensio, Daniel Casini, Luca Abeni, Alessandro Biondi, Mauro Marinoni, Carolina Fortuna, Blaz Bertalanic,	UMU, SSS, IJS, 8BELLS, DRAXIS, UBIT, i2CAT, TECN, NEC, Bi2S, VOS,

			Konstantinos Kaltakis, Alexandros Dimos, Georgios Katsikas, Grigoris Kalogiannis, Hatim Chergui, Cristina Regueiro, Francisco Javier de Vicente, Thanasis Tziouvaras, Anna Panagapoulou, Alvise Rigo	
0.3	15/07/2024	Section 4 and Section 5	Ramon Sanchez, Rodrigo Asensio, Gonzalo Alarcón, Antonella Clavenna, Maria Belesioti, Athanasios Liatifis, Dimitrios Pliatsios, Panagiotis Sarigiannidis, Thomas Lagkas, Sotirios Tegos, Marco Tambasco, David Franco, Anna Panagapoulou,	UMU, ITL, OTE, UOWM, TEI, EHU, VOS
0.4	26/7/2024	Internal review	Shih-Kai Chou, Konstantinos Kaltakis	IJS, 8BELLS
1.0	30/9/2024	Final version	Ramon Sanchez, Rodrigo Asensio, Gonzalo Alarcón	UMU

Internal Review History

Name	Organisation	Date
Shih-Kai Chou	IJS	26 Jul 2024
Konstantinos Kaltakis	8BELLS	26 Jul 2024

Quality Manager Revision

Name	Organisation	Date
Anna Triantafyllou, Dimitrios Pliatsios	UOWM	30 September 2024

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Explanation
QoS	Quality of service
CAPEX	Capital Expenditure
OPEX	Operational Expenditure
SLA	Service Level Agreement
SP	Service Provider
EU	End User
UP	User Plane
SLO	Service Level Objectives
KPI	Key Performance Indicator
IP	Internet Protocol
SoTA	State of the Art
UE	User Equipment
AI	Artificial Intelligence
DAC	Digital Agreement Creator
SDN	Software Defined Network
NFV	Network Function Virtualization
MDP	Markov Decision Process
ML	Machine Learning
RL	Reinforcement Learning
MEC	Mobile Edge Computing
ETSI	European Telecommunications Standards Institute
STEP	Standalone Explainable Protocol
MADRL	Multi-Agent Deep Reinforcement Learning
DQN	Deep Q-Network
VAE	Variational Autoencoder
IB	Information Bottleneck
DRL	Deep Reinforcement Learning
DNN	Deep Neural Network
SFC	Service Function Chain
QoE	Quality of Experience
RSU	Roadside Unit
DQL	Deep Q-Learning
DDQL	Double Deep Q-Learning
DDDQL	Double Duelling Deep Q-Learning
IoT	Internet of Things
PPO	Proximal Policy Optimization
AAC	Advantage Actor-Critic
PQC	Post Quantum Cryptography

SSI	Self-Sovereign Identity
DID	Decentralized Identifier
NLP	Natural Language Processing
RAN	Radio Access Network
B-RAN	Blockchain Radio Access Network
O-RAN	Open Radio Access Network
VM	Virtual Machine
Q-RAM	QoS-Aware Resource Allocation Model
OS	Operating System
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CFS	Compositional Scheduling Framework
ARM	Advanced RISC Machine
VOSyS	Virtual Open Systems
MEC	Multi-Access Edge Computing
SO	Service Orchestrator
NBI	Northbound Interface
BSS	Business Support System
TN	Transport Network
RIC	RAN Intelligent Controller
SMO	Service Management and Orchestration
NFVO	Network Functions Virtualization Orchestrator
DTs	Digital Twins
OBU	On-Board Unit
vOBU	Virtual On-Board Unit
V2X	Vehicle-to-Everything
API	Application Programming Interface
5GC	5G Core
OSS	Operations Support System
CI/CD	Continuous Integration/Continuous Deployment
VNF	Virtual Network Functions
CNF	Container Network Function
TL	Transfer Learning
FF	First Fit
BF	Best Fit
WF	Worst Fit
HBI	Highest Bandwidth Importance
HCI	Highest Core Importance
HBCI	Highest Bandwidth Core Importance
HBIcC	Highest Bandwidth Importance constrained to Cores
MMRB	Max-Min Residual Bandwidth

mmRB	Min-Min Residual Bandwidth
HRB	Highest Residual overall Bandwidth
LSF	Largest Service First
HQ	Highest QoS improvement
HQcC	Highest QoS improvement constrained to Cores
HQcB	Highest QoS improvement constrained to Bandwidth
HQcCB	Highest QoS improvement constrained to Cores and Bandwidth

Executive Summary

This document reports the work conducted in NANCY with a focus on the design and development of novel task offloading and user-centric data caching mechanisms for B5G infrastructures. Concretely, this deliverable presents the activities conducted in Task 4.1: “Computational Offloading & User-centric Caching Functionalities”.

The state-of-the-art for these technologies is given in Section 2, along with an explanation of the project-developed proposals that allow to fill the identified gaps in these fields. In order to achieve these advances, Section 3 presents the NANCY's offloading and caching architecture and provides a thorough definition of its constituent building blocks. Following this conceptual explanation, Section 4 discusses the integration of these assets in the various NANCY testbeds and demonstrators. Section 5 summarizes the key findings and facts, closing the document.

Therefore, this document provides an overall vision, from their design phase until their integration plans, of the different components and building blocks composing the NANCY computation offloading and data-caching schemes.

1. Introduction

Future 6G Networks are envisioned to behave as a unique system shared by different stakeholders. Virtualized and physical infrastructure will be prepared to host any kind of applications and services, changing the role from consumer to providers. By optimizing both resource and power efficiency, while keeping service and user requirements, shared infrastructures will bring CAPital and OPerational EXpenditure (CAPEX and OPEX) reduction at the time of advanced service provisioning. Led by virtualization means, especially in lightweight forms, agile resource handling, reallocation and migration are key enablers for these novel scenarios. Service provisioning will seamlessly leverage a wide range of infrastructure capabilities, self-adapting and self-configuring their components to enhance their KPIs performance thanks to intelligent resource and task management. The user-centric nature of 6G is envisioned to drive network service composition, shifting the traditional VNF-centric towards a more advanced view oriented to provide the best experience to customers.

In these dynamic scenarios, the nature and aims of the provided services vary depending on the field of application, e.g., the vertical under consideration. Considering the network performance and the capabilities of processing devices within the fog-edge-core-cloud continuum, as well as the Quality of Service (QoS) requirements of the provided service, computation tasks should be properly adapted or instantiated along this seamless chain. Besides, power and computational consumption are of special importance in constrained devices such as battery-supplied devices. In this regard, task offloading becomes a key enabler in 6G networks to address this challenge. Task offloading allows dynamic service handling by leveraging the possibility and flexibility of selecting the best node in which to execute a given task. In this sense, as mentioned above, the decision process driving these mechanisms depends on the nature of the service and its requirements, but also the characteristics of the available resources in the infrastructure. Following the user-centric nature of 6G networks, service consumers are the principal beneficiaries of task offloading in terms of improved experience. From the perspective of computational nodes, resource-constrained devices are beneficiaries of task offloading in terms of power consumption, while cloud nodes leverage task offloading to reduce network load and reduce end-to-end latencies.

Besides, to release the full potential of task offloading procedures, user-centric caching mechanisms should be considered as a highly valuable supporting scheme to reduce data access and local decision latencies. Indeed, caching mechanisms are crucial to provide contextualized information in a very short time, while reducing network usage hence releasing priceless network resources that may be used to transport big data for complex aggregation and analysis, for example, in the cloud. Caching mechanisms need to be designed carefully and adapted to each of the verticals and applications given that memory and storage are appreciated resources, especially in fog and edge computing. Therefore, a high caching hit rate is required to optimize these resources at the time of improving the end-user’s experience in terms of low latency service delivery.

In order to achieve effective task offloading and caching mechanisms, specialized AI engines are meant to take into consideration the peculiarities of data, services nature, requesters, and available resources. “What”, “where” and “when” to offload and cache are the three main rationales that these engines aim to solve. For instance, in low latency scenarios, proximity to the end-user is one of the main considerations; while for other capabilities, like intensive computation, affinity decisions should be considered to enable flexible orchestration placement for load balancing purposes.

1.1 Purpose of the Document

This document presents the NANCY's vision on advanced task offloading and user-centric caching in complex network environments where multiple stakeholders and domains are involved. Therefore, this deliverable mostly reports the activities conducted in Task 4.1: "Computational Offloading & User-centric Caching Functionalities". Along the following pages, the state-of-the-art of these technologies is provided together with a description of the proposals developed in the project that permit advancement in these fields. To this end, the NANCY's offloading and caching architecture is presented, and its building blocks are comprehensively defined. After this conceptual description, the instantiation of such components in the different NANCY's demonstrators and testbeds is discussed. The document is closed by summarising the most important facts and findings.

1.2 Relation to other Tasks and Deliverables

Although the work reported in this deliverable is mostly carried under the umbrella of T4.1 "Computational offloading and user-centric caching functionalities" it has direct links with the activities conducted in other WPs as well as other WP4's tasks. It has a direct connection with T4.2 "Resource elasticity enabling techniques" as certain elastic techniques developed in this task are being coupled with the offloading schemes addressed in T4.1. Besides, the data-caching mechanisms developed in this task will be exploited in T4.3 "Trustworthy grant/cell-free cooperative access mechanisms" to support the effective handling of user data in his/her registration or login process. Considering other WPs, as depicted in NANCY's Pert diagram, T4.1 has direct interactions with WP2, WP3, and WP6. Considering WP2, D2.1 "NANCY Requirements Analysis" provides the key NANCY requirements regarding the Key Performance Indicators (KPIs) to be considered during the service provisioning, which are materialized in T4.1 in the form of a well-defined Service Level Agreement (SLA) model. Regarding WP3, D3.1 "NANCY Architecture Design" defines the general NANCY architecture in which the offloading and user-centric data caching mechanisms developed in T4.1 should be accommodated. Finally, regarding WP6, D6.1 "B-RAN and 5G End-to-end Facilities Setup" presents the implementation plan that should be followed to integrate NANCY's offloading and caching schemes in the different demonstrators and testbeds of the project.

1.3 Structure of the Document

The rest of the document is structured as follows:

- **Section 2 – State of Art Overview** presents the state-of-art associated with service level agreement management, orchestration of offloading and caching, decision making, Blockchain-based transaction management, and resource handling at the edge.
- **Section 3 – NANCY Offloading and User-centric Caching** introduces the offloading and user-caching processes that are developed in the context of NANCY and documents the associated components.
- **Section 4 - Offloading and Caching in NANCY Demonstrators and Testbeds** describes the plans for the integration of offloading and caching mechanisms in NANCY's demonstrators and testbeds.
- **Section 5 – Conclusion and Outlook** summarizes and concludes the deliverable.

2. State of the Art Overview

2.1. SLA-based management

The Service Level Agreement (SLA) is a formal contract between the Service Provider (SP) (e.g., an Operator) and the customer (e.g., an End User (EU), another operator, etc.) where the high-level indicators and capabilities to be met are established, defined as Service Level Objectives (SLOs). It is a widely adopted method for agreeing on specific operational terms, such as the services to be used, their performance in terms of QoS, their availability, and any other relevant parameters that the provider commits to meet during the service provision.

As mentioned before, an SLA defines the precise metrics to measure service performance, which entails the tools to collect the relevant data [1]. These metrics may include network availability, bandwidth, latency, traffic management, computational efficiency and other Key Performance Indicators (KPIs). Some works stress the importance of defining network availability levels and specifying the minimum amount of bandwidth guaranteed for each user or service [2]. It is also necessary to define traffic management policies to prioritize certain types of traffic during network congestion as well as to specify maximum acceptable latency levels for different types of services. This should be done to ensure suitable performance, for example, thanks to the configuration of network slices [3]. In [4], it is stated that bit rate, jitter, packet loss rate and delay experienced by IP packets are the most useful parameters to describe how the network processes IP traffic, and therefore, the SLA should include them.

At the same time, an SLA permits the definition of a clear and hierarchical process for addressing any issue that arises during the provisioning of the service, guaranteeing efficient and faster problem resolution. SLAs not only cover the responsibilities of both the SP and the customer for the maintenance of the infrastructure and costs [5] but they may also include penalties and compensation in case of not meeting the agreed service levels [6]. To this end, network orchestrators make use of well-defined SLA schemes to drive their governance and manage network resources efficiently. When the orchestrator receives an SLA, it serves as an informational block for making well-informed decisions on network resource allocation and traffic management. The orchestrator analyzes the KPIs described in the SLA and translates them into concrete enforcement actions to optimize service and network performance. For example, the orchestrator can dynamically adjust bandwidth allocation to meet required service levels or prioritize specific types of traffic to ensure that critical applications receive special treatment during network congestion. Tightly coupled with SLA enforcement, orchestrators also monitor the system performance in real time to identify potential SLA breaches and take necessary actions to correct these breaches; this is crucial to keep the lawful status of the system. Traffic rerouting or applying service elasticity techniques, e.g., resource scaling or service migration, are typical countermeasures for keeping SLA assurance [7].

However, there are still some issues that need further research such as the SLA lifecycle management, which faces important challenges originated by the lack of a de facto standard for SLA definition or well-defined SLA-centric workflows. In this line, there are still gaps to be filled related to the diversity of protocols and processes in a complex network, the difficulties of modifying and negotiating SLAs between different administrative domains, the lack of well-defined service specifications and the need to clearly define the terms of the agreement to avoid confusion. SLA monitoring is also necessary to verify compliance with the agreement and ensure availability of services, as well as to monitor the connection and delivery of traffic [1].

Beyond SoTA

In NANCY, an SLA-centric workflow is proposed to ensure proper fulfilment of the requirements agreed in such SLA. A specific SLA model is used to shape the cooperation between different parties inside the 6G infrastructure. NANCY adopts a user-centric approach, hence from the user's perspective, the SLA management is shaped in different ways, as a proactive or reactive process. Regarding the former, the UE directly requests the home operator a certain service with an associated SLA presenting a set of well-defined KPIs. As a first step, the home operator attempts to handle the service request, providing all the needed resources. In the case that the home operator cannot comply with the SLA terms, it accesses an inter-domain marketplace to find a suitable external operator able to provide the requested service fulfilling its associated SLA. Regarding the proactive process, once a service is running, NANCY's monitoring and decision engine blocks in the serving operator may detect that the SLA terms cannot be guaranteed in the near future. Therefore, an alert is raised to the network orchestrator, which will look for a solution both internally, by reallocating the service and supporting resources, or in the inter-domain environment, where the marketplace module registers and finds the best provider to move the service and its SLA provisioning. Thereby, real-time monitoring guarantees compliance with the SLA by tracking each requirement based on data collected from the infrastructure. The collected data is deeply analyzed by AI-based decision engines to identify patterns, such as network anomalies or predictions regarding resource usage or mobility transitions. Besides, in inter-domain transactions, the role of the smart pricing module is crucial to find the most convenient price for the user considering the information published in the marketplace. Once the price of the service and its (re)allocation is set, the smart pricing module shares this information with the Digital Agreement Creator (DAC), which creates the final agreement between the parties involved. This digital contract is forwarded to the user and provider for its signing and further incorporation into the blockchain, prior to its final enforcement in the network by the orchestrator. This process, which will be comprehensively explained in the following sections of this document, increases the reliability of the system by providing transparency and integrity in resource-sharing agreements.

2.2. Offloading and caching orchestration

Resource offloading refers to the process of transferring resource-intensive tasks from one device to another, typically to enhance performance, save energy, or overcome hardware limitations or temporal events in the hosting system. To materialize this concept, different underlying technologies should be well orchestrated for their effective cooperation, e.g., Software Defined Networking (SDN), Network Function Virtualization (NFV), or network slicing, among many others. In the context of cloud-native network slicing, efficient resource offloading orchestration is crucial for optimizing network performance and meeting user requirements. Work in [8] introduces a fog-cloud container offloading system, treating the process as a multi-dimensional Markov Decision Process (MDP). They utilize a deep Q-Learning algorithm to generate efficient offloading plans, minimizing network delays and computation costs. Wang *et al.* [9] extend this approach to microservice coordination in edge-cloud environments, formulating the coordination process as an MDP model. They employ Q-Learning to find optimal solutions for service offloading while considering long-term performance metrics such as overall offloading delays and costs.

Various approaches have been explored for workload offloading in cloud environments. Authors of [10] propose a multi-component application placement algorithm to minimize application running costs across distributed edge servers. Work in [11] addresses the placement of AI functions using mixed-integer linear programming in federated learning settings. Reinforcement Learning (RL) methods, as demonstrated by Solozabal *et al.* [12], show superior performance in unknown environments compared to heuristic methods, although most existing RL works focus on centralized approaches, as seen in the work in [13]. Goudarzi *et al.* [14] explore distributed learning approaches for optimizing application placement in Fog/Edge servers.

However, existing works often overlook the complexities of multiple Mobile Edge Computing (MEC) systems with different orchestrators and fail to address standardization challenges related to the ETSI MEC architecture [15]. Work in [16]focuses on service placement load balancing in ETSI MEC architecture but neglects cross-orchestrator communication. Torres-Pérez *et al.* [17] propose a novel approach for MEC applications placement in highly distributed environments, aiming to minimize the number of active edge nodes while meeting user requirements. In this respect, efficient statistical multiplexing among slices is highly desirable in network slicing, primarily due to infrastructure costs and the dynamic traffic loads of each service [18]. While a slice is initially assigned a portion of distributed resources, it may be pre-empted by other concurrent slices if they remain unused. Without coordination, orchestrating agents for parallel slices might independently allocate more resources than their allocated shares based on their local observations, resulting in potential conflicts between slices. Thus, the main challenge lies in effectively managing resource allocation among concurrent slices to prevent inter-slice conflicts.

Beyond SoTA

NANCY goes beyond SoTA by designing a communication framework for future sixth-generation (6G) resource allocation, surpassing existing limitations. Referred to as the Standalone Explainable Protocol (STEP), this novel multi-agent deep reinforcement learning (MADRL) framework dynamically adapts communication messages to evolving system conditions, fostering the emergence of an on-the-fly protocol. This protocol facilitates conflict mitigation and minimizes resource contention among network slices while fostering offloading. STEP integrates concepts from Information Bottleneck (IB) theory with deep Q-network (DQN) learning principles. By incorporating a stochastic bottleneck layer inspired by variational autoencoders (VAEs), STEP imposes an information-theoretic constraint on emergent inter-agent communication. This ensures efficient exchange of concise and meaningful information among agents, preventing resource wastage and enhancing overall system performance.

2.3. Decision making

Decision-making in telecommunication networks refers to the set of processes used to manage and optimise the network performance, reliability and efficiency. Such techniques leverage the premises of optimisation theory in order to find the right balance between trade-offs such as latency-energy consumption [19]. In practice, several decision-making methods consider computation and data offloading scenarios, where a set of services or a collection of data is dynamically transferred between the UEs, the Cloud and the Edge of the network [20], [91], [22], [23]. In such cases, several constraints are in place such as the computational capacities of the involved devices, the network latency, the network throughput or the energy consumption, thus, researchers develop methodologies to optimise the service/data placement to maximise specific performance goals. With the advent of contemporary ML techniques, Deep reinforcement learning (DRL) has shown great promise in achieving significant improvements over the standard optimisation algorithms, while also increasing the automation levels within the network [24], [25]. Since the DRL approach is considered the current state of the art in decision-making scenarios, there are several models, techniques and architectures which are tailored to specific network types and application scenarios [26].

In DRL, one or multiple agents are trained to make decisions by interacting with their environment. DRL leverages the agent's knowledge and combines it with Deep Neural Networks (DNNs) to maximise a cumulative reward which represents the agent's goals. Under this premise, the agent takes actions which lead to changes in the environment's states and then, gets a reward based on how well these changes contribute to its objectives. Generally, DRL algorithms are distinguished between model-based, model-free, on-policy and off-policy. Model-based algorithms construct an internal representation of the environment (i.e., a model) which is used by the agent to predict the next state of the environment and thus, to estimate its reward. In contrast, in model-free DRL, the agent is unaware of the reward which will be obtained after performing a certain action. For this reason, model-free DRL encourages exploration and utilises the trial-and-error learning approach. On-policy techniques utilise a policy that governs the agent's behaviour and focus on evaluating and improving the same policy to select the optimal actions. On the other hand, off-policy DRL does not utilise a single policy to govern both the agent's behaviour and actions, instead, it allows the agent to learn from actions that were not chosen by the current policy. In the context of the decision-making process for telecommunication networks, most existing works fall into the category of model-free algorithms. Below, we discuss the most common approaches, models and use cases that stem from on-policy and off-policy model-free DRL methods.

The utilization of DRL for decision-making is also a very common approach in the latest innovations in communications. For example, the H2020 Predict-6G project [27] is utilised to resolve the traffic admission problems, while the H2020 6G BRAINS project [28] focuses on DRL resource allocation solution for the massive device-to-device connections in a highly dynamic cell-free network. Additionally, as shown in the H2020 5G-CLARITY [29] project, DRL can be successfully utilised for task offloading and resource reservation in B5G networks. Another focus of the 5G-CLARITY project was to examine the placement of Service Function Chains (SFCs) with DRL, for flexible service support and efficient resource utilization, so that the solution met Quality of Service demands, prevented congestion of edge resources, and enhanced the service acceptance ratio. DRL was also successfully deployed in the H2020 AI@EDGE [30] project in both improving the autonomous vehicle decision making among human drivers, to minimise traffic jams, while also optimising video analytics to dynamically optimise resource allocation by deciding the placement, migration, or horizontal scaling of serverless functions.

Model-free on-policy algorithms

Advantage Actor-Critic (AAC) is a synchronous DRL technique which employs multiple agents to update a central model. In terms of data offloading, AAC techniques have shown great potential in Internet of Things (IoT) ecosystems. In [31], the researchers develop a custom AAC model to cache IoT-collected data to the Edge. The conducted experiments show that AAC can maximise the long-term energy savings of the IoT devices, without any prior knowledge of the data popularity profiles. Proactive Edge caching is explored in [32], where a novel Proximal Policy Optimisation (PPO) DRL algorithm is proposed to enable Edge servers to perform data prefetch on each UE request. This model opens the pathway for collaboration between multiple servers which are in different areas, and it reduces the Edge-UE communication overhead. PPO DRL methods are also applied in [34] and [35], within the context of IoT-Edge networks to offload data from the IoT devices to the Edge. In such cases, DRL manages to optimise the energy efficiency and the IoT-Edge communication cost. A federated on-policy DRL approach is proposed in [33], tailored for data caching in vehicular networks. In this case, Roadside Units (RSUs) assume the role of Edge servers, which are trained in a distributed way to prefetch popular content in advance, while also taking into account each vehicle's position and moving direction.

Model-free off-policy algorithms

Q-Learning and more specifically Deep Q-Learning (DQL), which is a model-free and off-policy DRL technique frequently used for decision-making in telecommunication networks and, specifically, for data offloading and data caching functionalities. Previous work in [36] showcases the efficiency of DQL in proactive data offloading to the edge of the network. In this manuscript, authors employ DQL to decrease the energy consumption and increase the Quality of Experience (QoE) of the UEs. Cooperative edge caching is considered in [37], where a multi-agent DRL mechanism enables the data offloading from the Cloud to the Edge so as to relieve the network backhaul. Cooperation can also be achieved through a distributed training environment as previous work in [38] demonstrates. In this work, authors deploy a federated Duelling DQL network to improve the overall caching performance of the Edge. This ecosystem supports multiple Edge servers, with each one of them serving a distinct group of tenants. Federated DDQL is leveraged in [39] to support a cooperative proactive data caching technique in vehicular networks. In this work, multiple RSUs are located along the side of the road and serve incoming vehicle requests. The goal of the Federated DDQL is to reduce the response time and transmission delay of Road Side Units (RSUs) for data requests made by the vehicles.

Beyond SoTA

NANCY employs DRL to enhance and automate the decision-making operations (related to computation offloading and data caching) and more specifically, it leverages the Double Duelling Deep Q-Learning paradigm (DDDQL). The issue with the standard DQL is that it tends to be overoptimistic with its Q-value estimations. To solve this issue, the Double Deep Q-Learning (DDQL) is often employed. DDQL utilises two identical DNNs, one for predicting the Q-values and one for evaluating the predictions of the first one. In this way, the second network, which is usually referred to as the "target network", discourages the predictor (which is known as "Q-network") from generating erroneous predictions about each state's Q-values. This technique improves the network's convergence rate and significantly reduces its loss function. The "Duelling" part of the DDDQL comes into play by introducing a DNN split (both in target and Q networks) in the last layer of the model. By doing so, we separate the model output into two parts: The first part is referred to as "value", and the second part is referred to as "advantage". The "value" gives us information regarding the Q-value of the current state, while the "advantage" provides a quantitative evaluation of the action that led to this Q-value. This separation achieves two objectives: First, it allows the agent to focus on states where it's advantageous to act and differentiates the representation of the "state" and the "action" within the model itself. Secondly, the decoupling between the Q-values and the action advantages has the potential to increase the effectiveness of the agent's exploration. This has more impact in environments with complex dynamics (as in a UE-Edge-Cloud ecosystem) and can lead to a faster convergence rate and better stability of the learning process.

This DDDQL approach empowers NANCY's decision-making engines with the following benefits:

1) By incorporating robust and efficient data caching and computation offloading functionalities.
2) By increasing the applicability of the envisioned caching and offloading operations to different network types, topologies and devices.
3) By optimising the resource utilisation of network assets (through computation offloading), increasing the network throughput and reducing latency (through data caching).

For the reasons stated above, NANCY develops and implements two different DDDQL architectures for decision-making (one using a self-attention DNN and one using a dense DNN), and it performs several design-space exploration activities by changing their hyperparameters and modifying their characteristics.

2.4. Blockchain-based transaction management and accounting

Taking the blockchain as the central technology to achieve great levels of trustworthiness and transparency in NANCY's transactions, a complementary set of building blocks closely cooperate with each other to reach the desired automation and flexibility in resource and service management activities. These are the NANCY's marketplace, smart pricing module and Digital Agreement Creator (DAC). All of them are analysed in the following, one by one:

Blockchain

The NANCY Blockchain is based on Hyperledger Fabric [53] an enterprise-grade blockchain platform. In such a platform, data is stored in a distributed ledger that is maintained by each participating node, ensuring redundancy and resilience. Chaincodes, also known as smart contracts, form the backbone of Hyperledger Fabric's functionalities. They are located at the business logic layer, executing transactions and updating the ledger's state. A chaincode is essentially a decentralized application that encapsulates the rules governing how assets are managed within the blockchain network. These facilitate automated transactions upon meeting predefined conditions. Moreover, in Hyperledger Fabric, each ledger is treated as an independent subnetwork. This ensures data isolation and confidentiality, allowing organizations to transact securely without revealing sensitive information to unauthorized parties. Through these mechanisms, Hyperledger Fabric provides a robust foundation for secure, scalable, and privacy-preserving enterprise blockchain applications. Its modular architecture, pluggable consensus protocols, and customizable endorsement policies empower organizations to tailor blockchain solutions to their specific requirements, ushering in a new era of trust and efficiency in digital transactions.

As explained previously, an SLA acts as a formal agreement between a Service Provider (SP) and a user, which includes the SLOs that need to be met. Once a customer request triggers a service-launching process, e.g., a function offloading, the SLA must be converted into chaincode and deployed onto the blockchain by the responsible entity. Throughout this process, it is crucial to certify the trustworthiness of each entity's resources. Chaincodes are pivotal in this scenario, as they can either store this information directly or implement a scheme where data is stored off-chain with an on-chain integrity-checking process. In this line, the wallet entity plays a key role in enabling a secure connection and simplifying the process. It contains cryptographic materials necessary for securely connecting with the blockchain and acts as a "proxy" entity. When an entity needs to interact with the blockchain, it contacts the wallet to forward the requests, thus maintaining security and integrity throughout the transaction process.

Beyond SoTA

NANCY proposes several improvements over traditional blockchain implementation in three different domains: (i) PQC-enabled wallet and blockchain: NANCY proposes the use of physical PQC cards for the UE, for which an adaptor is being developed and will be integrated with the wallet. This will make it possible to use PQC materials (keys) and signature generation for secure interactions with NANCY's blockchain. To support PQC signature verification smart contracts are envisioned to be integrated into NANCY's blockchain. This module verifies the signed chaincode from the wallets at the blockchain's end. The registration to the blockchain using post-quantum cryptography material is an improvement over the state of the art that can help prevent malicious use of (previous non-quantum) crypto keys, e.g., stealing confidential product data or financial records, or manipulating service prices. In this case, both the end users (users of the blockchain) and the blockchain owner/provider benefit from this enhancement. (ii) Integrated blockchain-based marketplace and smart pricing components: NANCY proposes the integration of a marketplace and a smart pricing component on top of the NANCY's blockchain. The objectives of such a system are (1) to maintain trust for the marketplace and smart pricing components, (2) to maintain price fairness and (3) to keep indelible track of the agreement reached by all parties. This latest point, more specifically, refers to the ability of the system to negotiate a price between the service provider and the end user for a given resource with the conditions expressed in a publicly available SLA that is added to the NANCY's blockchain as a chaincode. By collecting performance (and other) KPIs, the system should be able to track down any deviations from the SLA, allowing NANCY to keep the accountability and reputation of the provider (operator). (iii) Self Sovereign Identity (SSI)-enabled wallet: Digital privacy prevents the illegitimate use of users' personal data and automatically improves the blockchain owner's reputation. On Web3, protecting a user from having unnecessary information shared with third parties without their consent or knowledge is of fundamental importance. In the case of NANCY, the system is equipped with decentralized identification features. A Decentralized Identifier (DID) [54] is decoupled from a formal identity and allows the user to fully control it. DIDs can be generated and used based on the user and service requirements independently from other, third-party identity providers. What this means is that, once the user is authenticated and granted specific verifiable credentials, it is able to present those to a service provider to obtain a given resource, with no need to reveal its identity or any other credential. In this case, both the end users (users of the blockchain) and the blockchain owner/provider benefit from this enhancement.

Marketplace

There are various studies that address the concept of telecommunication marketplaces [40], [41] where a marketplace serves as an intermediary between providers and customers. For providers, it extends the market reach without additional marketing costs and allows fair competition with larger providers. However, for customers, the marketplace provides a single location for comparison with transparency about availability and price. In the telecommunications ecosystem, marketplaces already facilitate the exchange of various telecom services and products between operators or end users [42]. However, these marketplaces are constantly evolving driven by technological advancements and changing consumer demands. On the one side, in recent years, the rapid growth of bandwidth and latency reduction demands as well as the network intelligence improvement have increased the pressure on mobile operators that sometimes experience a shortage of resources and require offloading of some tasks to other available resources. In this sense, telecommunication resources marketplaces are required with updated information about the availability status of existing resources at any time to correctly offload essential tasks while guaranteeing network correct operation [43]. On the other side, with the increasing threat of cyberattacks and data breaches, security has become a priority, so in this sense, Blockchain has started to be considered as a secure technology to backbone telecommunications marketplaces as it provides transparency, integrity and availability by design [44].

Beyond SoTA

A blockchain-based marketplace for telecommunications resources will be considered for making the offloading tasks easier through updated information about available resources as well as updated prices for fair competition at any time. This marketplace will be a central piece of the NANCY’s inter-domain environment together with the rest of the building blocks presented in this section, namely, the smart pricing and DAC modules which will closely work together with the NANCY’s blockchain. In this way, 6G networks will enhance their operation by avoiding bottlenecks while guaranteeing transparency, trustworthiness and fairness.

Smart pricing

In recent years, several innovative smart pricing policies have emerged within mobile content marketplaces and blockchain networks. S. Hosny *et al.* [45] introduced a mobile marketplace that enabled users to purchase digital content via smartphones, featuring a dynamic pricing model based on content popularity, quality, and demand. This model offered a superior user experience by providing convenient and accessible content anytime, anywhere. Work in [46] developed a social welfare maximization auction mechanism for edge computing in mobile blockchain networks. This mechanism optimized resource allocation by considering users' heterogeneous demands and computing resources, adjusting auction parameters accordingly, and showcasing increased performance in simulations compared to existing state-of-the-art mechanisms. Authors of [47] proposed an optimal pricing mechanism for data markets in blockchain-enhanced IoT networks, using a combinatorial double auction model to ensure fair compensation for data owners and efficient transactions, achieving high revenue for sellers and low cost for buyers.

Numerous studies have integrated economic and market factors into pricing algorithms. Authors of [48] and [49] explored strategic pricing in the telecom industry, where operators optimize prices to maximize profits, taking into account client preferences and competition dynamics. These studies examined scenarios in which pricing decisions are influenced by owners, renters, or regulators aiming to maximize profit or promote social welfare. Kumar *et al.* [50] focused on external factors such as demand-supply evaluations, network traffic, latency, end-user figures, and subscriber growth rates. They employed models like the Shapley value, bargaining games, and dynamic pricing to maximize revenue and distribute costs among stakeholders, demonstrating the importance of these external factors in shaping effective pricing policies.

These advanced methodologies underscore the evolving landscape of smart pricing policies, highlighting the need for strategic and adaptive solutions in diverse technological contexts. Researchers continue to explore auction techniques and game-theoretical models to maximize social welfare and optimize resource distribution, particularly in blockchain and IoT environments. The integration of external economic factors and advanced algorithmic approaches remains crucial in developing effective pricing strategies.

Beyond SoTA

NANCY's smart pricing module aims to provide several enhancements to existing state-of-the-art solutions, particularly by integrating dynamic pricing mechanisms within a Blockchain-RAN network (B-RAN). This integration is unique as it is one of the few, if not the only, such mechanisms designed to work seamlessly with task offloading processes to ensure efficient allocation of tasks to available providers and infrastructure within the network. By addressing the commercial and financial aspects, NANCY aims to create a unified network where providers can cooperate effectively, ensuring financial sustainability and operational efficiency. Technically, NANCY's integration of the smart pricing module within its general architecture is distinctive. By incorporating the module into blockchain mechanisms and smart contracts, NANCY ensures fast price calculations and a tamper-proof system, minimizing errors and enhancing reliability. This integration leverages the strengths of blockchain technology to provide a robust and secure pricing mechanism. In terms of implementation, NANCY is exploring an arsenal of novel tools through continuous experimentation. After a thorough review of relevant literature and prior works, it is evident that the most effective approach for optimizing bidding strategies, similar to those used in current state-of-the-art solutions, involves Stackelberg Games or Reverse Auction theory. Traditional theories like Bertrand and Cournot have inherent limitations, such as reliance on a scarcity of bidders or a focus on quantity rather than price. NANCY proposes using Reverse Auction theory implemented by Reinforcement Learning (RL) agents for two main reasons. First, RL offers excellent scalability, suitable for the complex dynamics of 5G and B-RAN networks with numerous users and MNOs. Second, the flexible nature of RL algorithms allows for continuous adjustments based on evolving datasets, providing significant adaptability to new situations. This approach ensures that NANCY's smart pricing module remains effective and responsive to changing network conditions and demands.

Digital Agreement Creator

Digital Agreement Creators (DACs) have become quintessential tools in the evolving landscape of business and legal technology [51]. Leveraging cutting-edge advancements in AI, blockchain, natural language processing (NLP), and cloud computing, these platforms are transforming how contracts are drafted, reviewed, and executed. On the other hand, in the rapidly evolving digital landscape, the advent of 5G technology has promised and delivered significant leaps in connectivity, speed, and latency. One area poised for transformation with the widespread adoption of 5G is digital contract creation. Therefore, one of the most profound impacts of 5G on digital contract creators is the enhancement of real-time collaboration capabilities. 5G's high bandwidth and low latency allow multiple stakeholders to work on contract creation, review, and editing simultaneously from various locations with virtually no delays and lags. In this line, this improved network performance offered by 5G significantly enhances the performance of AI and ML algorithms used in DACs. These technologies depend on vast amounts of data and computational power to learn from historical contracts, predict risks, and suggest language modifications. With 5G, AI-driven platforms can process larger datasets faster and run more sophisticated models, resulting in higher accuracy and more nuanced contract generation.

One of the hallmarks of 5G technology is its ability to facilitate edge computing [52], which allows data processing closer to the data source rather than relying solely on centralized cloud servers. This decentralization can be highly beneficial for digital agreement management, especially for contracts involving IoT devices and other smart applications. For example, a supply chain contract could leverage edge computing to autonomously monitor compliance with contract terms in real time, responding instantly to any discrepancies. Platforms that incorporate edge computing as part of their framework can offer enhanced reliability, lower latency, and increased data security, transforming how contracts are monitored and enforced. Besides, 5G technology enhances blockchain's potential in enforcing smart contracts, which are self-executing contracts with the terms directly written into code. These smart contracts can benefit from 5G’s enhanced connectivity, enabling more timely and reliable data feeds required for their execution. Industries using blockchain for real-time, automated contractual agreements will find that 5G allows for almost instantaneous data validation and transaction processing. This significantly reduces latency and increases the efficiency of smart contracts in sectors such as finance, healthcare, and logistics.

In summary, the integration of 5G networks into digital contract creation tools marks a significant milestone. By enhancing real-time collaboration, enabling immersive AR/VR experiences, boosting AI capabilities, facilitating edge computing, and improving security and compliance, 5G is set to revolutionize how contracts are created, managed, and executed. As DACs continue to evolve alongside 5G technology, businesses will find themselves better equipped to handle complex negotiations, streamline their workflows, and adapt quickly to changing legal landscapes.

Beyond SoTA

The rise of DACs, underpinned by blockchain technology, is revolutionizing the way agreements are formed and executed. By automating and securing transactions through smart contracts, these digital solutions enhance transparency, reduce costs, and minimize the risk of fraud, paving the way for more efficient and trustworthy digital economies. In addition, NANCY's DAC, due to its internal orchestrator for creating ad-hoc containers, responsible for the actual creation/deployment of the smart contracts, differentiates from the current SoTA with the advantage of being more flexible in smart contract creation. This provides great flexibility when it comes to handling heterogeneous service provisioning requests such as those envisioned in task offloading scenarios. Together, these advancements are propelling us toward more interconnected and intelligent infrastructures, in which different stakeholders can share their resources and offload processing functions to improve user experience.

2.5. Resource handling at the edge

In the rapidly evolving 5G infrastructure, NFV has now turned into a conventional paradigm. NFV effectively decouples the execution of network functions from dedicated hardware, as it provides the ability to run them on standardized, commercial off-the-shelf infrastructure [55]. Specifically, thanks to virtualization, network functions abstract hardware limitations and thus, are deployable or offloaded across all different parts of the network. This shift brings great flexibility to the 5G networks, which can now orchestrate the lifecycle of VNFs at different domains of the network, where the edge, given its distributed nature, allows service proximity to the users. Therefore, virtualization technologies at the edge of the network are key enablers for achieving efficient and flexible task offloading.

The virtualization technologies sitting behind the implementation of VNFs are either VMs, containers, or sometimes a hybrid approach that combines both [56]. Besides the advantages provided by these technologies, each of them also presents some limitations. In detail, VMs are realized with hypervisors, which can either sit directly on top of the hardware without requiring an underlying operating system (Type-1) or are deployed on top of an operating system (Type-2) and take advantage of its capabilities. In the context of NFV, the VMs are commonly implemented with some Type-2 hypervisor, like Linux Qemu/KVM or VMware vSphere [56]. VMs offer a more traditional approach to implement virtualization, by emulating the physical hardware to host multiple Operating Systems on a single physical server. Each VM runs its own dedicated OS and is both isolated and independent of others [57]. In this context, the key benefit of using VMs is that they guarantee strong isolation and security. However, this solution comes with a relatively high overhead, due to the necessity of emulating or virtualizing a hardware platform, which therefore impacts both performance and scalability [56]. On the other hand, containers are a modern solution to virtualize network functions in the 5G ecosystem [57]. There is currently a tendency to move towards the containerized, or also cloud-native approach because it is a very lightweight alternative to classic, hypervisor-based virtualization. By using containers, applications and their dependencies are encapsulated in the form of micro-services, which can be flexibly moved across environments. Moreover, virtualization with containers maintains a close-to-native performance. However, containers are only isolated at the OS level, and they share the same kernel, which introduces challenges in terms of security.

Regardless of the technology used to offload applications to edge or cloud nodes, a crucial issue is how to assign the nodes' resources to the virtual environments hosting the offloaded applications. In this regard, it is very important to find the correct trade-offs that allow for avoiding resource over-provisioning and bad performance due to under-dimensioned resources. The problem of optimizing resource allocation to maximize the utility of real-time applications has been previously considered in the literature. For example, the QoS-Aware Resource Allocation Model (Q-RAM) by Rajkumar and others [58] modelled the execution of multiple real-time applications on a single node. Each application is allocated to multiple resources and is characterized by multiple QoS dimensions that also depend on the resource-specific allocation. Although this model did not consider distributed or multi-core systems (one single worker node with one single CPU core was modelled), it resulted in being fairly generic and quite difficult to solve. Hence, simplified models (with one single QoS dimension and/or one single resource) were initially solved. The work was then extended to solve the optimization problem in more complex cases [59], even considering discrete CPU allocations [60]. Even if support for multi-core systems was later added [61], Q-RAM did not consider distributed systems. Methods for guaranteeing the timing constraints of real-time applications running on single nodes are generally based on the so-called Compositional Scheduling Framework (CFS) [62] [63] [64] [65], which can result in resource over-allocation. By specializing in resource allocation algorithms for a class of applications, it is possible to improve performance (at the cost of generality). For example, NFV has been considered [66]. By specializing in resource allocation algorithms for a class of applications, it is possible to improve performance at the cost of generality. For example, NFV has been considered in the work presented in [66]. Besides, Struhar et al. [67] targeted the allocation of containers on Linux, under SCHED_DEADLINE, but without considering mixed edge-cloud applications. Only the edge of the network is considered. Casini et al. [68] considered the local load balancing of real-time applications running on multicore embedded systems. However, edge-cloud is distributed. Struhar *et al.* [67] targeted the allocation of containers on Linux, under SCHED_DEADLINE, but without considering mixed edge-cloud applications. Only the edge of the network is considered. In turn, Casini *et al.* [68] considered the local load balancing of real-time applications running on multicore embedded systems.

Beyond SoTA

NANCY develops a novel technology to achieve virtualization, as an alternative to a VM or a container-based approach, aimed at VNF deployments at the network edge. This technology supports the ARMv8 architecture, which is a very popular choice for edge equipment suited for low-power consumption applications, without sacrificing performance. At its basis, the proposed virtualization technology resembles usual VM deployments, although it achieves bare-metal execution latency as well as stricter, hardware-enforced security guarantees. The ARMv8 architecture features a hardware technology known as ARM Trustzone. This technology allows the creation of secure enclaves within a device's processor, ensuring isolated execution environments for sensitive data and critical operations. Also, it has proven [69] to be suitable in mixed-critical scenarios, because it allows the concurrent execution of critical and non-critical activities on the same hardware platform. In the context of NANCY, VOSySmonitor [69] is employed at the ARMv8 edge servers, which is a software that creates on top of ARM Trustzone an environment capable of simultaneously managing and running whole, individual, bare-metal operating systems. In NANCY, the different operating systems that are deployed on the same hardware, and which either feature critical or non-critical characteristics, are called compartments. In most platforms, the non-critical compartments usually have access to most parts of the hardware and thus can be viewed as the “rich” ones. On the other hand, the critical compartments have a limited view of the hardware and are suitable for specific real-time applications.

As for the novel virtualization technology, NANCY designs it with the purpose of opening up the critical compartments to host offloaded VNFs, which will be deployed in a bare-metal fashion. Specifically, NANCY allows the exploitation of the ARM Trustzone hardware-enforced isolation, for the purpose of isolating the bare-metal, critical VNFs, from a pre-deployed resource-rich compartment. Each bare-metal VNF operates within its own compartment, with VOSySmonitor managing these compartments and orchestrating the allocation of primary resources such as CPU and memory to each of them. However, the big challenge behind this virtualization idea is that VOSySmonitor has to accommodate the level of abstraction that is required on the bare-metal compartments, to run the VNFs. In general, the VNFs interface with a generic view of the hardware, via specific virtual devices. Therefore, it should still be possible for the VNFs to be transparent in terms of the virtual devices and behave as if they are in a system with an underlying Hypervisor. NANCY meets the needs for deploying the abstracted VNFs with a software solution called cross-compartment virtio-loopback. This technology is a hypervisor-less virtualization technology, where the rich compartment behaves as the hypervisor and addresses the hardware requests, while on the non-rich compartments, the VNFs are deployed with all the views of the generic hardware. So, at the edge level, the resources become abstracted thanks to the cross-compartment virtio-loopback; thanks to it the para-virtualized resources are exposed to the VNFs that will access them with no limitation. The technical details of the cross-compartment will be further elaborated in the upcoming sections. More specifically, how NANCY achieves to provide a bare-metal, lightweight, edge-targeted virtualization solution based on it. Besides, considering data caching, compartments also permit to safely store information only accessible by authorized processes. This technology will be employed for this regard under the scope of the activities in T4.3: Trustworthy grant/cell-free cooperative access mechanisms; therefore, the developments in this regard will be reported in the corresponding deliverable D4.3.

Again, independent of the technology used to offload applications to edge or cloud nodes, the orchestration of virtualized software workloads to meet real-time requirements, e.g., bandwidth, and latency is crucial. To this end, NANCY also proposes new allocation algorithms for optimizing the QoS-to-cost ratio of edge-cloud distributed applications considering the SCHED_DEADLINE scheduler of Linux, which allows providing real-time guarantees and timing isolation between services that are allocated on the same node and cores. This provides substantial benefits in avoiding the underutilization of computational resources while still guaranteeing the required QoS even in the presence of contention for the computational resources. This is a key objective in massively distributed systems with task offloading demands such as those considered in NANCY.

While previous work addressed methods to maximize the QoS of real-time applications or proposed methods to guarantee the real-time constraints for single-node systems, none of them proposed allocation methods that jointly considered the usage of a scheduler with sound theoretical properties such as SCHED_DEADLINE, which allows providing guarantees on the CPU bandwidth and the CPU worst-case latency, while also considering distributed applications in the edge/cloud and focusing on optimizing a QoS-to-cost ratio.

3. NANCY Offloading and User-centric Caching

3.1 NANCY extended orchestration

In a previous document (D3.1), a series of baseline architectures defining the main NANCY's building blocks and their interactions were presented [70]. In the case of the NANCY's general architecture presented in D3.1, a functional and deployment view of its building blocks has been produced to contemplate new needed functionalities originated with the evolution of the project and represent them all together in a clear way (Figure 1).

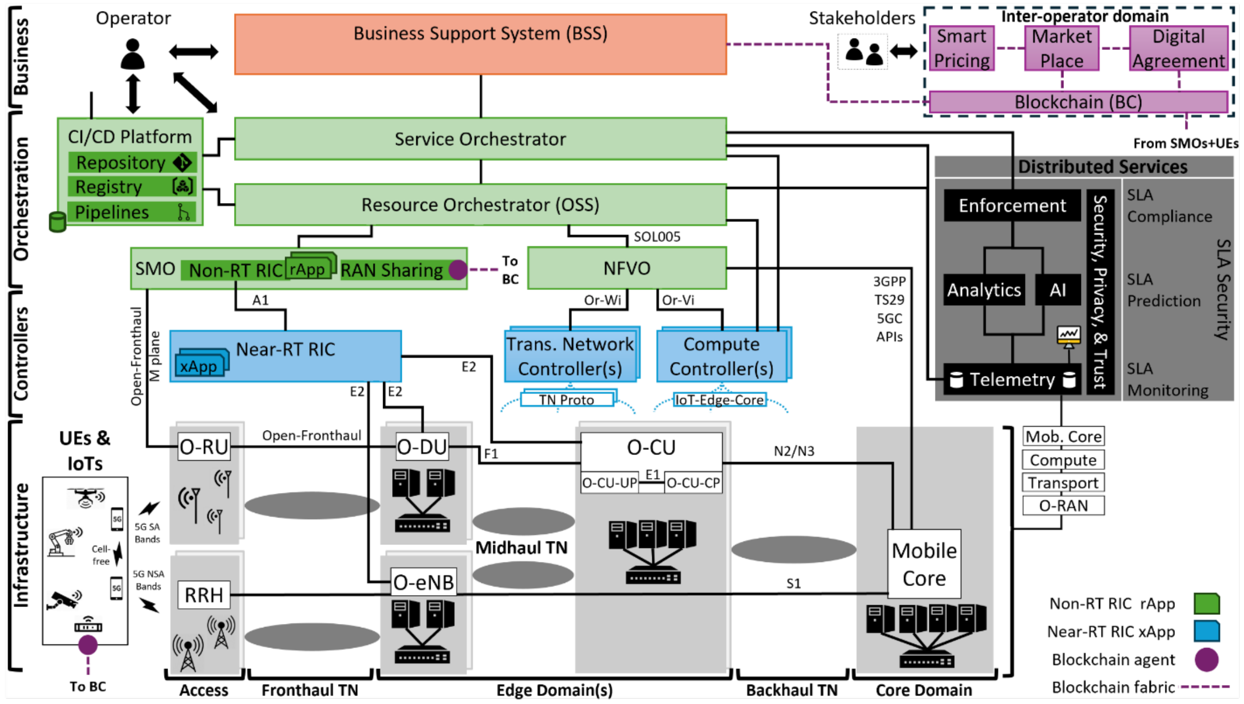


Figure 1. Functional and Deployment view of the NANCY Architecture.

Its structure is fully maintained, with two main operational domains: The inter-operator and the intra-operator domains. Considering the former, it refers to the domain that leverages the multi-stakeholder approach of the 6G networks to share and optimize resource usage while reducing the operating costs of infrastructure management. It is composed of the NANCY's marketplace, the smart pricing block, and the DAC that closely collaborate to handle and manage the pool of resources offered by the different operators. These elements are crucial to enable the cross-domain and cross-operator vision of NANCY. Besides, the presence of the NANCY blockchain in this domain ensures transparency, robustness and trust in all the transactions among multiple stakeholders such as operators, users, etc., as all of them will be recorded on it.

In turn, the rest of building blocks in the architecture belong to the intra-operator domain, which refers to the part of NANCY that is managed inside each operator; in this case, the different segments forming the network infrastructure, namely, UE and IoT devices, radio access, edge, backhaul transport, and core, are comprehensively monitored and managed by the Telemetry system. Taking into consideration the nature of both offloading and caching functionalities, monitoring in real-time the status of the infrastructure is crucial. This kind of exhaustive monitoring is represented by the individual and specialized monitoring tools, in which the key data to be under control is obtained. These are the O-RAN, Transport, Compute, and Mobile Core monitoring tools in the diagram. Besides, although a high level of automation is highly desirable, the Visualization of the gathered data in a comprehensive way becomes necessary to allow human-in-the-loop solutions. Thus, the Telemetry block is also important to dynamically improve and expand the creation of the datasets used to train the AI-based engines.

As can be seen, the principal consumer of the data generated by the Telemetry system is the Analytics and AI components. In these blocks is where the NANCY's main intelligence resides. Analytics engines are the first elements consuming the data monitored and extracted from the network infrastructure. With these data, this block continuously checks the SLA compliance and the resources allocated for its accomplishment. Pattern-based analysis is considered in this block as well for making different kinds of predictions. Accordingly, AI-based predictions regarding the future status of the systems considering both computation and network resources are obtained by the AI engines in the AI block. In the case of detecting that the SLA compliance is at risk, an alert is generated towards the Enforcement block, which will send a detailed alert to the Service Orchestrator to keep the SLA in fulfilment. Therefore, the Analytics and AI blocks process monitored data to determine the level of performance, security, available resources (non-allocated resources), underutilized resources (allocated but not used), and establish predictive behaviour.

In this line, upon the arrival of a service request or during the execution of a running service, NANCY's decision engines and analytics procedures continuously check if the SLA associated with such a service can be fulfilled by the operator serving it. In the case of inferring a possible SLA violation, an automatic and well-defined process (detailed in Section 3.3) is launched to solve the issue before it happens. Thereby, when the resources in the serving infrastructure are not enough to deal with the situation, an alternative solution is sought in the inter-operator domain, a process launched by the local Service Orchestrator through the Business Support System (BSS). At this point, the Marketplace and the smart pricing blocks autonomously look for the most adequate solution for the customer, which will involve the migration of the provided service and its state towards a different operator with available resources to accommodate the service while fulfilling the SLA terms. Therefore, a new digital contract between the former and the new service provider should be prepared and signed by both parties. This will be done in the form of a smart contract that will be produced and registered in the blockchain by the DAC for transparency and robustness purposes. Once this process is completed, the orchestrator in the new service provider will be informed about the new service to be onboarded together with its associate SLA, and the resource and service allocation and enforcement process will be launched through the different domain controllers (blue boxes in the architecture). On the other hand, the orchestrator in the former service provider will support the migration of the service and its associated data before liberating the previously occupied resources, again through the domain controllers implied in the service provisioning. This workflow will be exhaustively detailed in Section 3.3.

In this Section, starting from this NANCY architecture depicted in Figure 1, the focus is on delving into the specifications of the different modules and components that participate in the orchestration process of NANCY's offloading and caching mechanisms. Firstly, it is important to mention that both mechanisms are considered specific cases of orchestration, in which the concrete characteristics of the use case and scenario under consideration determine which decision modules participate and, therefore, what kind of data is monitored to feed these intelligent engines. Accordingly, the orchestration and enforcement of the needed actions will be applied to the different domains, namely, UE, RAN, MEC, and cloud, in different ways, depending on the scenario conditions and infrastructure characteristics. This flexibility is highly relevant to attending and handling a variety of requests in multi-stakeholder environments such as those envisioned for B5G systems.

In this line, offloading and caching mechanisms are key procedures to accomplish the KPI requirements, and consequently the SLAs, especially considering specific conditions or situations in the infrastructure such as high load or volume of traffic, data or user-centric services, mobility of the UE, resource-constrained devices, latency in critical applications, etc. A series of building blocks and processes implied on NANCY's offloading and caching processes have been identified: SLA-based management, Orchestration, Decision engines, Blockchain, Marketplace, Smart pricing, and Edge-resource management. The coordination among them is driven by automation loops that establish flexible flows to exchange data and commands. In the following, the role and interactions of these components involved in the offloading and user-centric caching processes within the NANCY's architecture are thoroughly described.

3.2 Offloading and user-centric caching building blocks

3.2.1 SLA-based management

Having a separate business model from the technical KPIs facilitates the dynamic nature of the service provisioning proposed in NANCY. Once the digital agreement has been signed by the parts and incorporated into the blockchain, while the SLA terms should be kept constant, the business part may change during the service life cycle. For instance, when the current service operator is not able to provide the signed SLA terms, the marketplace comes into play to provide seamless collaboration between infrastructure and service providers to accomplish the SLA in an optimal way. The incorporation of a new service provider is simplified as it is only needed to sign again the business part of the contract, linking it to the conditions established in the static SLA stored in the blockchain.

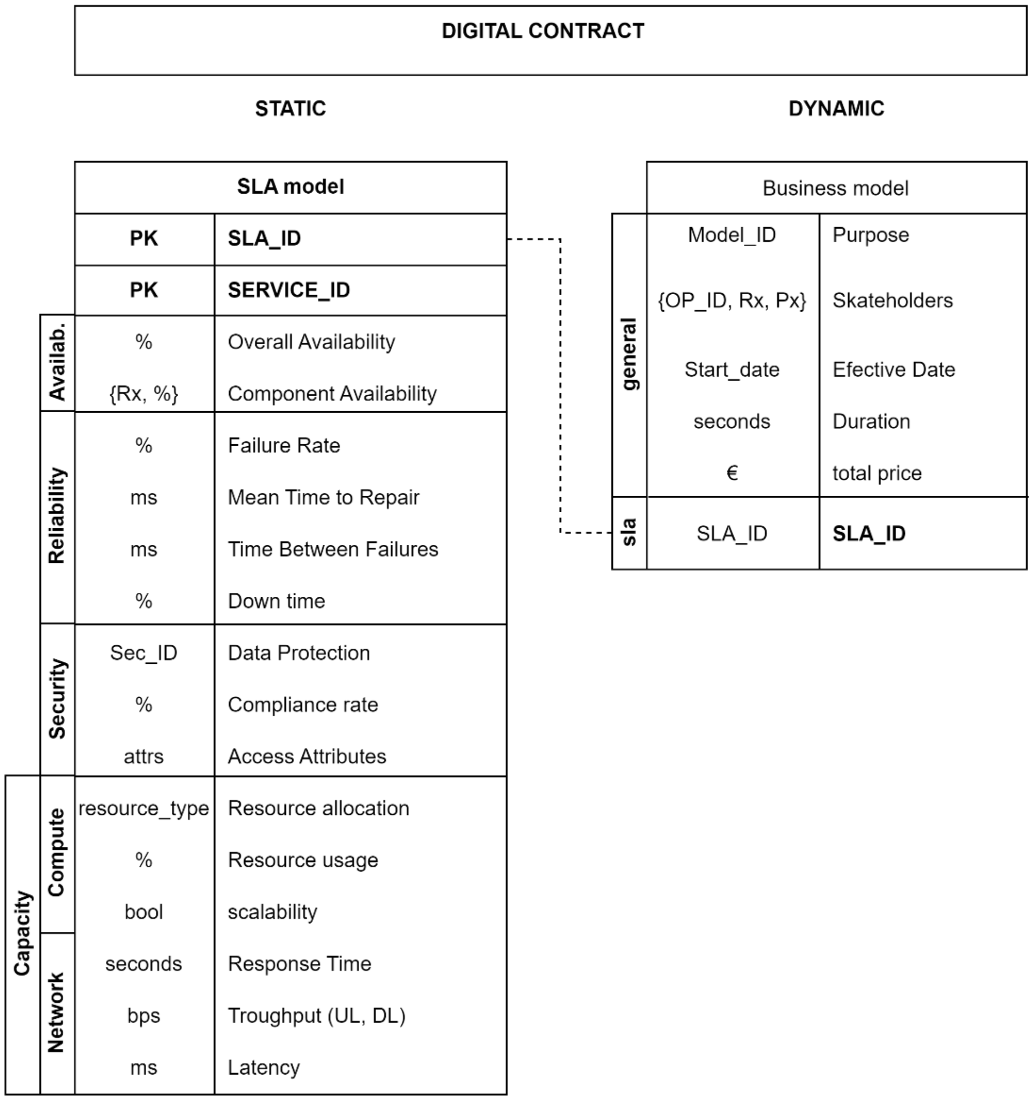


Figure 2. Digital contract (SLA + Business models) structure

Figure 2 depicts the digital agreement model, which has been grouped into different categories and information elements, subsequently described:

Static part (SLA model):

- SLA ID: It uniquely represents the static part of the contract (SLA).
- Service ID: It uniquely represents the provided service.
- Availability: Metrics specifying the percentage of time that the service or its components should be available:

- Overall Availability: Considering the service chain delivery, the percentage of time that the service should be available.
- Component Availability: For each of the resources provided by the different stakeholders, the percentage of availability during the service delivery.

- Reliability: Specifies the robustness to failures of the system:
  - Failure Rate: Maximum percentage of failures during the service provisioning
  - Mean Time to Repair: Time to recover defective components/resources and include reestablishment of the service provisioning.
  - Time Between Failures: Average minimum time elapsed between errors.
  - Down time: Percentage of time that the service is allowed to be inoperative.
- Security: Covers security aspects of the service:
  - Data Protection: Model representing several security capabilities required by the service (e.g., physical layer protection, confidentiality, authentication, etc.).
  - Compliance Rate: % of time related to the fulfilment of the security capabilities indicated previously.
  - Access Attributes: Which attributes are required to access the requested service.
- Capacity:
  - Compute: Defines the behaviour of the physical and virtual resources associated with the service:
    - Resource Allocation: Refers to CPU, GPU, memory, storage and network required by the service. Each of them has its own measurement unit.
    - Resource Usage: Minimum percentage of resource utilisation.
    - Scalability: Boolean parameter indicating the capability to scale up or down certain allocated resources.
  - Network: Focused on measuring different metrics of the data plane:
    - Response Time: Maximum allowance time elapsed since the service request is sent until the SLA has been enforced, hence the service is operative.
    - Throughput: Required bandwidth for uplink and downlink to keep a proper degree of data rate for service consistency.
    - Latency: Maximum round time trip for every packet of the service connection.

Dynamic part (Business model):

- General: contains information related to non-technical elements of the service model.
    - Purpose: Joined flags that contain bits representing the Service/Slice ID, Modes (e.g. Follow the UE, Scalable), and Behaviour (e.g., offloading).
    - Stakeholders: List of stakeholders participating in the agreement, with associated resources/components and the price for each of them. OP_ID, Rx and Px are the operator identifier, resource identifier, and price function associated, respectively.
    - Effective Date: Timestamp from which the SLA is operative.
    - Duration: Expiration time of the SLA.
    - Total Price: Summatory of the Px for every Rx in case of being fixed it will be a static integer.
- SLA: Links the business and technical parts of the digital contract.
    - SLA ID: couples this part of the contract with a specific SLA.

More details about the utilisation of this digital contract by different components identified in the offloading and data caching processes are provided in the following sections.

3.2.2 Orchestration

As mentioned above, we consider both offloading and caching mechanisms as specific processes in which an adequate overall system orchestration is needed. Therefore, NANCY envisions to provide an end-to-end platform for the provisioning and lifecycle management of generic services both within a single operator's realm, but also across multiple operators. To do so, NANCY advocates the need for an orchestration platform that can manage resources and overlay services across multiple heterogeneous domains within an operator's ecosystem as shown in Figure 3.

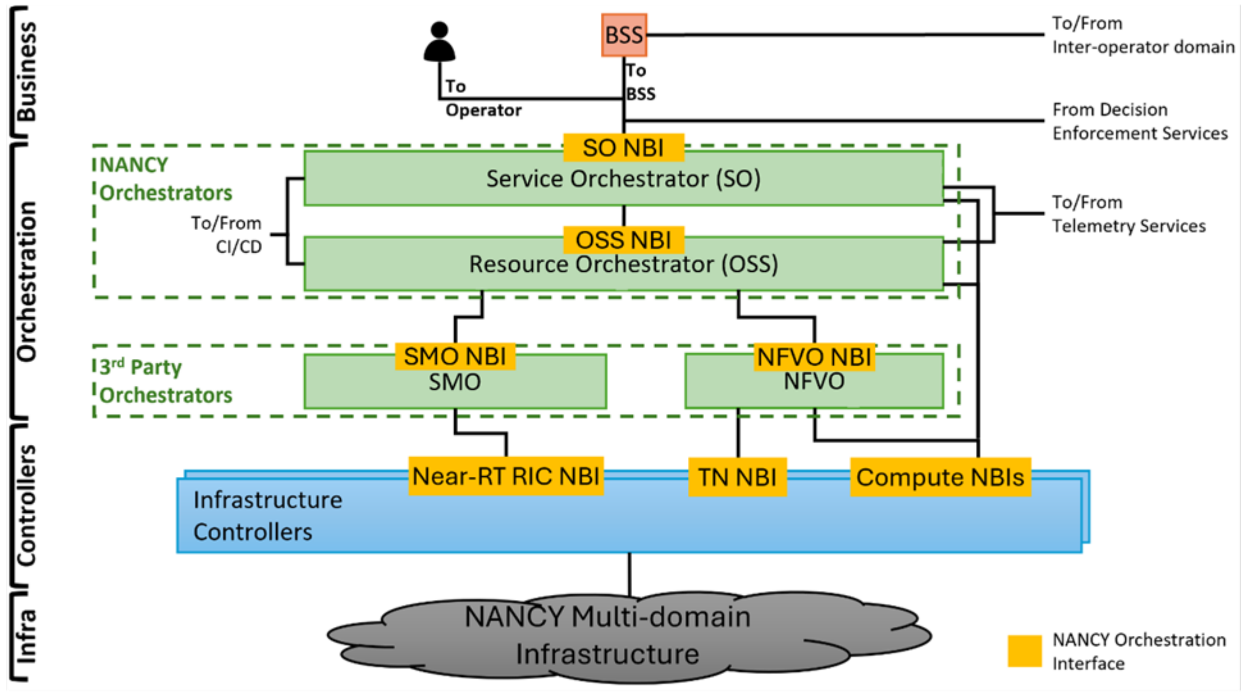

Business
Orchestration
Controllers
Infra
BSS
To Operator
To BSS
To/From Inter-operator domain
From Decision Enforcement Services
To/From Telemetry Services
NANCY Orchestrators
To/From CI/CD
SO NBI
Service Orchestrator (SO)
OSS NBI
Resource Orchestrator (OSS)
3rd Party Orchestrators
SMO NBI
SMO
NFVO NBI
NFVO
Near-RT RIC NBI
TN NBI
Compute NBIs
Infrastructure Controllers
NANCY Multi-domain Infrastructure
NANCY Orchestration Interface

Figure 3. High-level overview of the NANCY Orchestration platform.

The NANCY orchestration platform resides in the “Orchestration” layer shown on the left-hand side of Figure 3. This layer interacts with:

- Northbound stakeholders or systems are as follows:
  - The “Operator” oversees a certain domain through the Service Orchestrator’s (SO) north-bound API (NBI) and possibly a portal where important operations of the SO NBI are supervised.
  - The “Business” layer of the NANCY architecture through the SO north-bound API (NBI). In this layer, a Business Support System (BSS) consumes services from the NANCY orchestrators and maps these services to products and their billing operations. These products are sold to NANCY stakeholders either within the operator’s domain or advertised towards the NANCY inter-operator marketplace (to be consumed by 3rd party operators) as shown at the top part in Figure 3.
- Southbound controllers in the “Controllers” layer (i.e., the blue boxes in Figure 3) of the NANCY architecture are as follows:
  - A set of compute controllers which expose compute services from the underlying infrastructure, either by means of virtual machines (i.e., Infrastructure-as-a-Service) or containers (i.e., Platform-as-a-Service). These controllers may span across the IoT-to-Edge-to-Core cloud continuum.
  - A set of Transport Network (TN) controllers which manage connectivity services in the fronthaul, midhaul, and backhaul network segments.
  - A set of Near-Real-Time (RT) RAN Intelligent Controllers (RIC) which manage either O-RAN-based or legacy radio access networks (RAN).

<u>Internal layout of the NANCY Orchestration platform</u>

Within the "Orchestration" layer, NANCY introduces different orchestration entities for managing the underlying complexity of resources as shown by the green boxes in Figure 3. These orchestration entities are presented in a bottom-up fashion below:

Third-party Orchestration Entities (see Figure 3):

- A Service Management and Orchestration (SMO) automates radio resource management in O-RAN-based domains. The SMO specifications are defined by the O-RAN alliance [71]. Among others, the SMO employs a Non-Real Time RAN Intelligent Controller (Non-RT RIC) that interacts with the Near-RT RIC at the "Controllers" layer and the underlying O-RU, O-DU, and O-CU elements in the O-RAN domain as shown in Figure 3.
- A Network Functions Virtualization Orchestrator (NFVO) provides standardized APIs, such as the ETSI NFV SOL005 [72][73], for managing network functions (NFs) across the operator's network. These NFs mainly include, but are not limited to, the functions of the 5G core (5GC). In modern times, such functions are software-based, thus the NFVO can manage the underlying compute controllers to deploy these NFs either as virtual machines, i.e., vNFs, or containers, i.e., cFNs.

NANCY aims to leverage existing, standardized systems for SMO, e.g., The Aether Linux Foundation project [74] or similar, and for NFVO, e.g., the ETSI OSM [75] platform, thus it focuses on the design and development of high-level orchestration entities.

NANCY Orchestration Entities (see Figure 3):

- A Resource Orchestrator acts as the operator's Operations Support System (OSS). This platform interfaces with 3rd party orchestration components, such as the SMO and the NFVO, to manage compute and network resources within an operator's network in an end-to-end fashion. The mission of the OSS is to hide these resources behind resource-as-a-service services, thus exposing these resources to the upper layer, i.e., the Service Orchestrator described next, in a secure way.
- A Service Orchestrator (SO) manages end-to-end services within an operator's environment. To provision these services, the SO first consumes resources-as-a-service from the underlying Resource Orchestrator (RO) and then instantiates the service instances atop these resources, while providing lifecycle management APIs to the operator (and/or BSS) for managing the runtime behaviour of these instances.

NANCY promotes RO and SO solutions that are based on open and standardized interfaces too. Such interfaces will be based on ETSI [76] and 3GPP [77] specifications as well as the industry-grade open APIs introduced by TMForum (TMF) [78]. Specific NANCY partners, i.e., I2CAT and UBITECH, are in charge of the design and development of these components and their interfaces.
Additional details about the exact integration of the NANCY Orchestration platform with the rest of the NANCY ecosystem are provided in the NANCY D6.1 "B-RAN and 5G End-to-end Facilities Setup" [79].

In the rest of this section, we describe the basic operations of the NANCY orchestrators.

NANCY Service Orchestrator Operations

A summary of the NANCY Service Orchestrator's operations is provided in Table 1. The SO uses these operations to provide end-to-end services to an Operator or an overlay BSS. An end-to-end service (or slice) can be seen as a collection of sub-services (or sub-slices), each devoted to a specific domain. That said, an end-to-end service might comprise of (i) end-user service components residing in one or more compute clusters (compute slices), (ii) a 5G slice that comprises of (O-)RAN and 5GC (User Plane Function (UPF)) slices, and (iii) connectivity services that link the RAN, 5GC, and compute resources together.

Depending on the API used for provisioning end-to-end services, the data model for describing these end-to-end services might be different. For example, an ETSI OSM-based end-to-end slice might be seen as a collection of network service (NS) instances, while a TMF-based end-to-end service might be seen as a collection of service specifications, one for the end-user service, one for 5G, and another for the compute resources of the service.

Table 1. Summary of the NANCY service orchestration operations.

Orchestration Operation	**Description**	**Relevant APIs**
Service onboarding	An operator uses standardized means to describe a service to the NANCY SO (i.e., using service descriptors). This description is uploaded onto the SO's service catalogue, where the operator can pick the service for order/provisioning.	ETSI OSM NBI featuring ETSI NFV SOL005 [80] TMF633 Service Catalog Management [81]
Service ordering/provisioning	An operator browses the service catalogue of the NANCY SO, picks items from this catalogue, and composes service orders to initiate the service provisioning process.	ETSI OSM NBI featuring ETSI NFV SOL005 [80] TMF641 Service Ordering Management [82]
Service lifecycle management	Once a service order is completed, a service instance is available on a given domain. The operator uses a runtime API to manage the characteristics of this service.	ETSI OSM NBI featuring ETSI NFV SOL005 [80] TMF 638 Service Inventory Management API [83]
SLA management	An operator relates a given service with certain quality of service (QoS) criteria or a service-level agreement (SLA).	TMF657 Service Quality Management API [84] TMF 623 SLA Management API [85]
Party and party role management	An operator models the parties (individuals or organizations) that interact with his/her platform as well as the roles (e.g., service provider, infrastructure provider, etc.) of these parties in the operator's environment.	TMF632 Party Management API [86] TMF669 Party Role Management API [87]

	A party can also be a system, thus the NANCY SO can use this API to describe its affiliation with a specific NANCY RO instance or to peer with another SO east-west.	
Service artefacts management	After a service is described using the service onboarding operation above, links to the service artefacts are provided for the SO to know from which software repository and/or registry to pull the service artefacts (e.g., container images, service manifests, etc.). This is the link between the NANCY orchestrators and the NANCY CI/CD platform.	Popular software registry platforms, such as jFrog [93], GitLab [94], GitHub [95], Harbor [96], etc.

NANCY Resource Orchestrator Operations

The RO uses the same service-level APIs as the SO (see Table 1) to describe/order/manage resource-level services. However, the RO uses additional operations to describe/order/manage resources. A summary of these additional NANCY Resource Orchestrator operations is provided in Table 2.

Table 2. Summary of the NANCY resource orchestration operations.

Orchestration Operation	Description	Relevant APIs
Resource onboarding	An operator uses standardized means to describe a resource to the NANCY RO (i.e., using resource descriptors). This description is uploaded onto the RO's resource catalogue, where the operator can pick the resource for order/provisioning.	ETSI OSM NBI featuring ETSI NFV SOL005 [80] TMF634 Resource Catalog Management [88]
Resource ordering/provisioning	An operator browses the resource catalogue of the NANCY RO, picks items from this catalogue, and composes resource orders to initiate the resource provisioning process. Depending on the type of resource to be provisioned, the RO makes a connection of this resource with the respective controller in charge of managing this resource, thus delegating resource provisioning towards (i) the SMO for O-RAN-based resources, (ii) a TN controller for TN resources, and (iii) a	ETSI OSM NBI featuring ETSI NFV SOL005 [80] TMF652 Resource Ordering Management [89] Kubernetes API [91] OpenStack API [92] LF Aether project [74] (or a similar platform as an SMO)

	compute controller for compute resources.	
Resource lifecycle management	Once a resource order is completed, a resource instance is available on a given domain. The operator uses a runtime API to manage the characteristics of this resource.	ETSI OSM NBI featuring ETSI NFV SOL005 [80] TMF 639 Resource Inventory Management API [90].

Note that some of the relevant APIs (e.g., the Aether project acting as SMO) introduced in Table 2 are indicative as we are still in the process of investigating different options.

3.2.3 Decision engines

In this subsection, we elaborate on the characteristics of the AI models implemented for supporting the decision-making processes related to offloading and caching in the NANCY ecosystem. As mentioned in section 2.3, NANCY leverages the Double Duelling Deep Q-Learning paradigm (DDDQL) paradigm for decision making.

High-level overview of the AI-based decision-making engine

Figure 4 presents an overview of the DDDQL training framework. It contains an environment, which is formulated by the training dataset, that encapsulates all the possible states an agent can occupy. States are discrete time-dependent conditions, composed of several features such as *network latency*, *throughput*, *Edge storage capacity*, *Cloud processing capacity* etc. The number of features for each state depends on the dataset used for the training process. We should note that the training DDDQL framework for the NANCY project is designed with a major focus on interoperability and thus, users can easily change the training dataset (along with the number and type of features) without requiring any manual adaptation of the framework itself.

The general overview of the training process is considered as follows: An agent collects information about the current state of the environment. To accomplish this, the agent accesses the dataset and acquires the values of the corresponding features that represent its state. Subsequently, the agent takes an action upon the environment, e.g., “to offload a service to the Edge of the network”, and this decision forwards the agent into the next state. This transition will generate a reward based on its contribution to the agent’s goals, e.g., increasing the throughput of the targeted service. Then the NANCY framework collects the information related to the agent’s initial state, action, next state and reward and stores it in an experience buffer, which is a First-In-First-Out (FIFO) queue that temporarily stores the experience of the agent in order to enable the model to learn from its past behaviour. Afterwards, the model training process commences, during which the experience buffer is sampled to create a training batch and the DDDQL network is trained accordingly. This whole process continues until a predefined number of epochs passes. Below, we discuss the details of the DDDQL models, the training process, the decision-making operation, the datasets used and the application of the decision-making engine to the NANCY ecosystem.

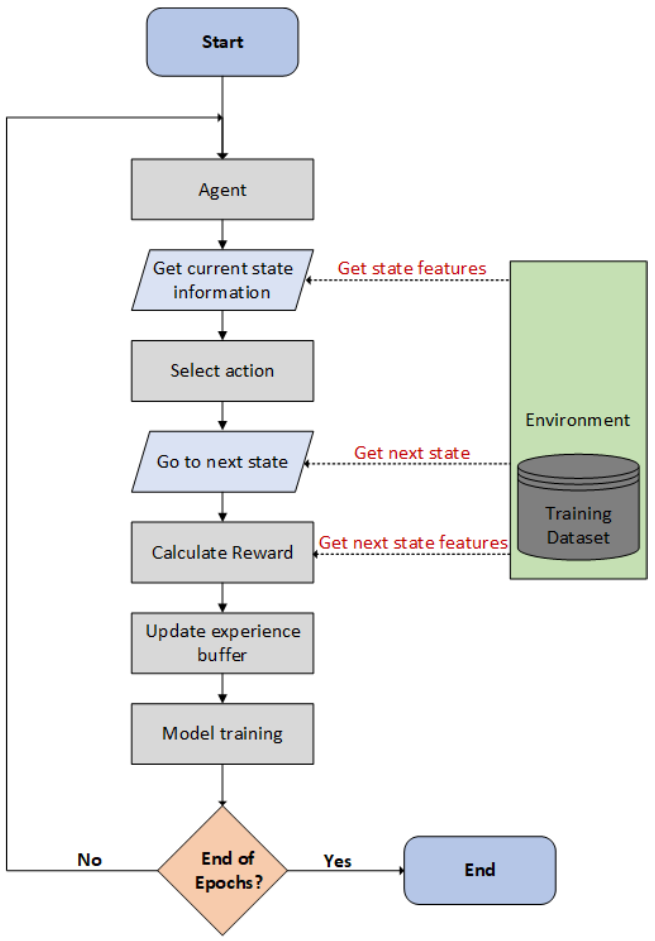


Figure 4. NANCY's training framework for DDDQL.

<u>DDDQL model architecture</u>

The models implemented for the decision-making process fall into the category of DDDQL and thus, they share some similar characteristics. First, the AI models utilise the Q-Learning (QL) algorithm, which is a model-free off-policy DRL approach that calculates the "Q-values" for each state. Q-values are used to analyse the expected value of an action/state, and they can be leveraged to look into future actions/states to evaluate their rewards as well. As a result, Q-values represent both "how good a state/action is" (given a certain goal) and "how good are the future states/actions". This gives the Q-learning the capacity to not only check for the near-future state rewards but also allows for long-term planning by checking the consecutive state rewards an action may lead to. Q-learning utilises the current state information, the next state information, the action and the reward to predict the Q-values of a state. We share more details on how this operation works during the training process, in the subsection named "DDDQL model training" below.

The DNN models used for the DDDQL framework are illustrated in Figure 5. The first model utilises 4 layers of 1-dimensional convolutions, accompanied by pooling and normalization layers to expand the dimensionality of the input data. Input data contain state information related to several network parameters at a given timeslot, such as *network latency*, *network throughput*, *RAM available to the Edge*, *computational unit availability on the Cloud*, *storage capacity of a UE* etc. Such data are convoluted with several filters to explore patterns that emerge between them. The pooling layers and normalisation layers are utilised to increase the convergence rate of the training process and to minimise gradient explosion issues. After expanding the dimensionality of the input data to 128 dimensions, a self-attention layer is leveraged to find the best spatial representation for this information. Self-attention is a layer that employs dot-product operations, followed by a SoftMax function in order to project the input data into a spatial coordinate system which maximises the distance between individual data points. This results in a lower mean square error (MSE) and in a clearer data representation format. We should also note that due to the several 1-D convolutional layers, the risk of gradient vanishing increases. To alleviate this risk, we utilise a residual connection from the output of layer 2 towards the input of the self-attention layer. Residual connections add older values, i.e., outputs of upper DNN layers, with new values obtained by lower DNN layers to preserve information throughout the model. Finally, the self-attention output is split into the "advantage" and "value" components, as described above.

The second model, which is referred to as the "dense model" utilises a different approach. It is composed of four dense feedforward layers which expand the input data dimensionality immediately to 128 dimensions. In this architecture, every neuron of a dense layer is directly connected with every neuron in the next layer and thus, a dense layer essentially performs several linear transformations to the input data. Dense layers can learn complex data patterns and can be used to aggressively increase the dimensionality of data. Contrary to the convolutional layers, dense networks can quickly make high-level associations between input data and layer weights. This can lead the network to generalise better, even with a lower amount of training data. In this architecture, we also employ residual connections to lessen the gradient vanishing, and we split the output into "advantage" and "value", similar to the self-attention model.

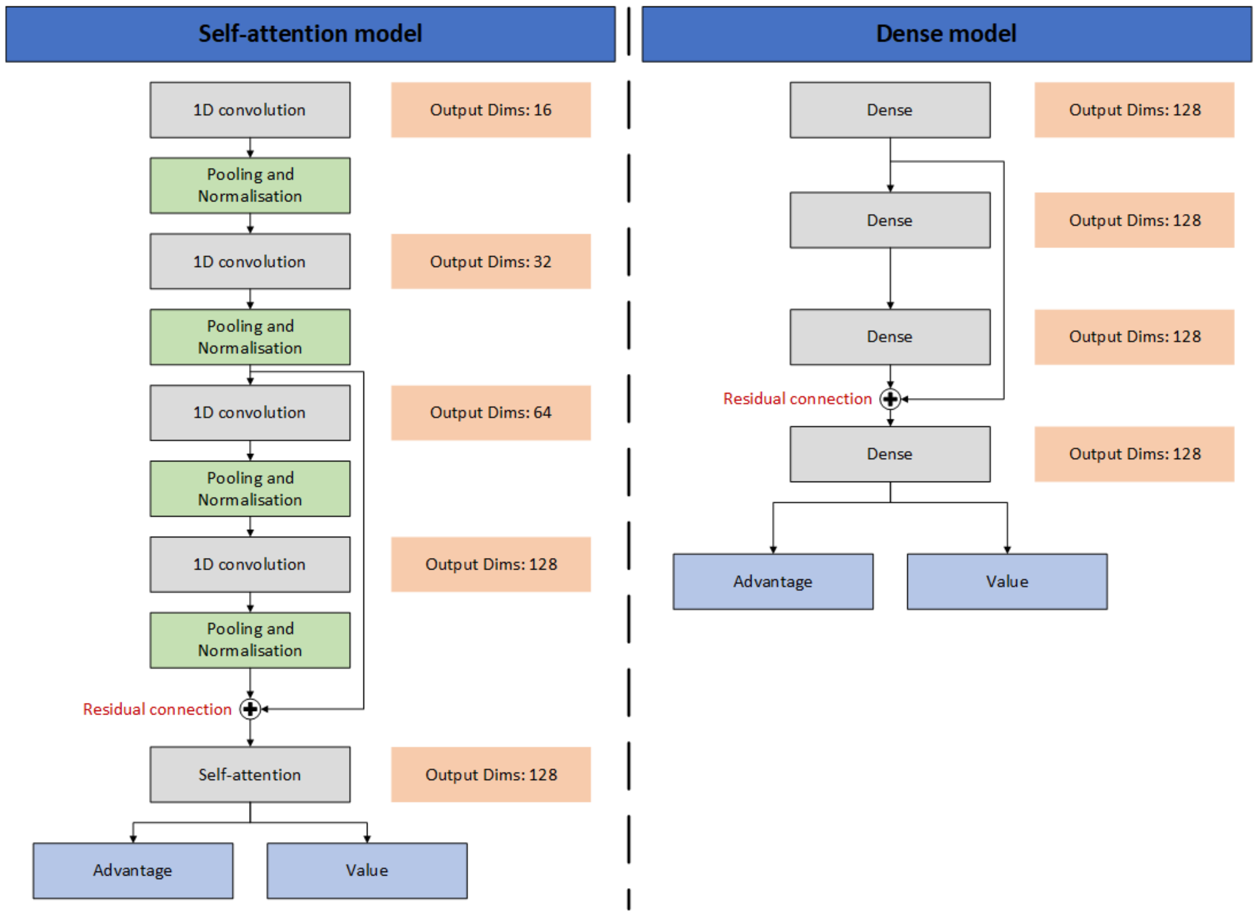


Figure 5. DNN models used for the DDDQL decision-making engine of NANCY.

DDDQL model training

Figure 6 depicts the training process of the DDDQL model. We should note that for a given decision engine we select: **(i)** either the Self-attention model as both "Target-NET" and "Q-NET"; or **(ii)** the Dense model as both "Target-NET" and "Q-NET". A performance evaluation of these choices can be found in "DDDQL application to the decision-making engine of the NANCY ecosystem" below.

During the training process, the experience buffer is sampled to obtain the current and next state information of the agent's experiences. Data related to actions taken by the agent in the past, and their corresponding rewards are also recalled during this phase. Then, the "next states" data are fed into the Target-NET for inference. Inference is the process of utilising a trained DNN to predict an output, given an input sequence. In this sense, during the inference process, no back-propagation takes place and thus, the model is not trained. The Target-NET's "value" and "advantage" are added together and are later used to calculate the Q-value of the sampled batch. We follow a similar approach for the Q-NET model, which is also used for inference with the "next state" data. Its "value" and "advantage" are added together and through an argmax operation, the action with the maximum advantage is selected.

Generally, in DQL the Q-Values are calculated using the following Bellman equation (1):

$$NewQ(S,A) = Q(S,A) + \alpha * r(S,A) + \gamma \underset{a}{(max)} Q\left(S^{'},A^{'}\right) \qquad (1)$$

Where ***New Q(S,A)*** represents the next Q-Value of the next state ***S*** and next action ***A***. Likewise, ***Q(S,A)*** refers to the current Q-Value of the state S and action A. Learning rate α is leveraged to weight the rewards ***r(S,A)*** which the agent obtains by selecting the action ***A*** in the state ***S***, and a discount rate γ is used to control and determine the importance of future rewards. The $\underset{a}{(max)} Q\left(S^{'},A^{'}\right)$ term is the maximum expected future reward of the agent.

For the DDDQL, the Bellman's equation is transformed in the following way (2):

$$NewQ(S,A) = Q_1(S,A) + \alpha\left(r(S,A) + \gamma Q_2\left(S^{'}, \underset{a}{(max)} Q_1\left(S^{'},A^{'}\right)\right) - Q_1(S,A)\right) \qquad (2)$$

The main difference in the updated equation is the introduction of two Q-Vales, namely the $\boldsymbol{Q1}$ and the $\boldsymbol{Q2}$. The value of the $\boldsymbol{Q1}$ is obtained by Q-NET, while the value of the $\boldsymbol{Q2}$ is obtained by the Target-NET.

After the Q-Value calculation which is referred to as $\boldsymbol{NewQ}(\boldsymbol{S},\boldsymbol{A})$, using the DDDQL Bellman's equation and the outputs of the Q-NET and Target-NET, the model training process kicks in. For this purpose, we utilise the "current state" information, sampled by the experience buffer. The "current state" data are packed into a training mini batch and they are fed into the Q-NET. The Q-NET initiates a forward pass of the training batch and then, it utilises the $\boldsymbol{NewQ}(\boldsymbol{S},\boldsymbol{A})$ as validation data in order to complete the back-propagation process. In this sense, the Q-NET is trained using ground truth data that stem from the Target-NET. This process allows the Target-NET to lower the overoptimistic Q-Value predictions made by the Q-NET, during the DDDQL decision-making process. Within periodic time intervals, a weight transfer operation takes place and updates the weights of Target-NET with the weights of Q-NET. This happens because the Target-NET is not trained (note that only the Q-NET is trained) and thus, the model should update itself with the newly acquired knowledge. The exact frequency at which this operation takes place is a hyperparameter of the DDDQL process and significantly affects the model convergence rate. We discuss this aspect in the subsection "DDDQL application to the decision-making engine of the NANCY ecosystem" below.

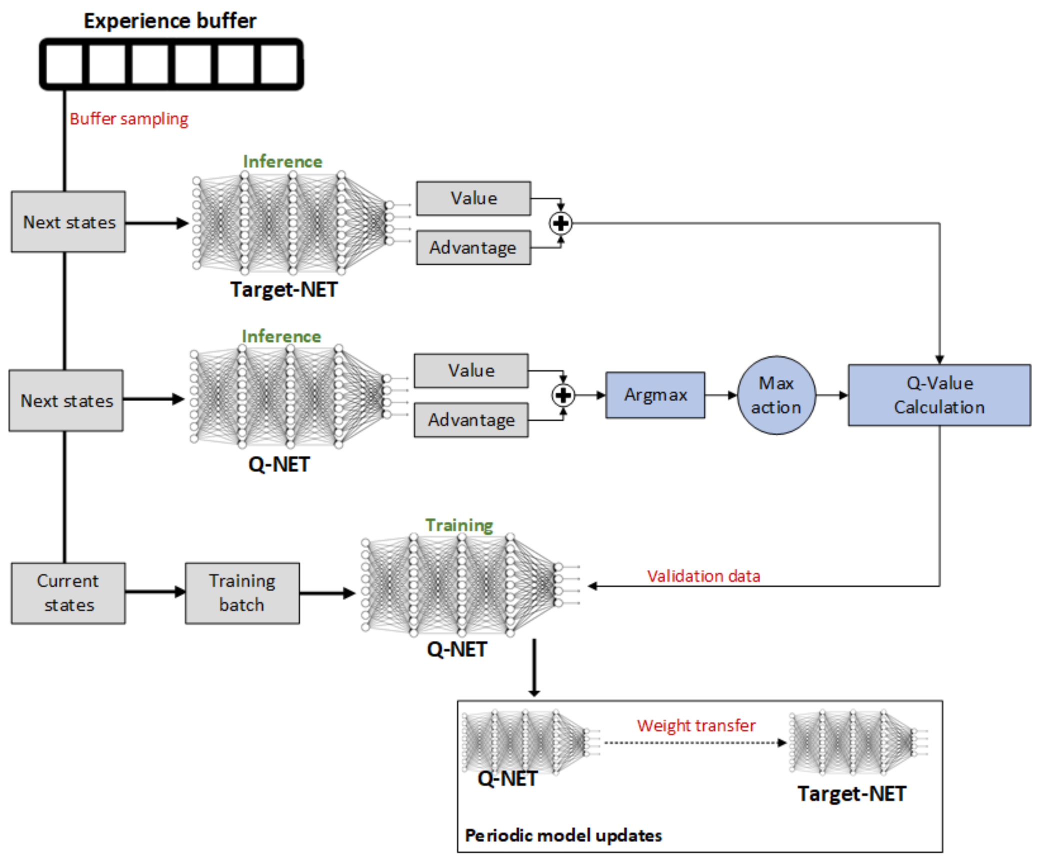


Figure 6. DDDQL model training process.

DDDQL decision-making

Figure 7 illustrates the decision-making process which is invoked when the model is required to perform an action that corresponds to a computational or data-offloading decision. Initially, current state information is fed to the system. Such information is then transferred to the Q-NET, which performs an inference operation. For the decision-making process, only the "advantage" is selected, and the "value" is discarded. An argmax function selects the action with the higher advantage and then forwards the result to the "exploration strategy" module. This module sets the strategy which the agent follows during the training process. An agent can either choose "random state exploration" (by performing random actions within a state) or choose "max advantage exploration" (by choosing the actions with the maximum advantage). "Random state exploration" is useful for the agent to discover new states, actions and rewards which are not dictated by DDDQL policy. As a result, for each training epoch, some random actions are foreseen in order to enhance the agent's experience. On the other hand, if the "max advantage" exploration is selected, the agent follows the DDDQL policy and selects actions that lead to the maximum rewards. Finally, a reward is calculated and attributed to the agent after considering the "current state" information, the "next state" information and the action taken. Such data is also saved to the agent's experience replay buffer and is used for the training process, as described above.

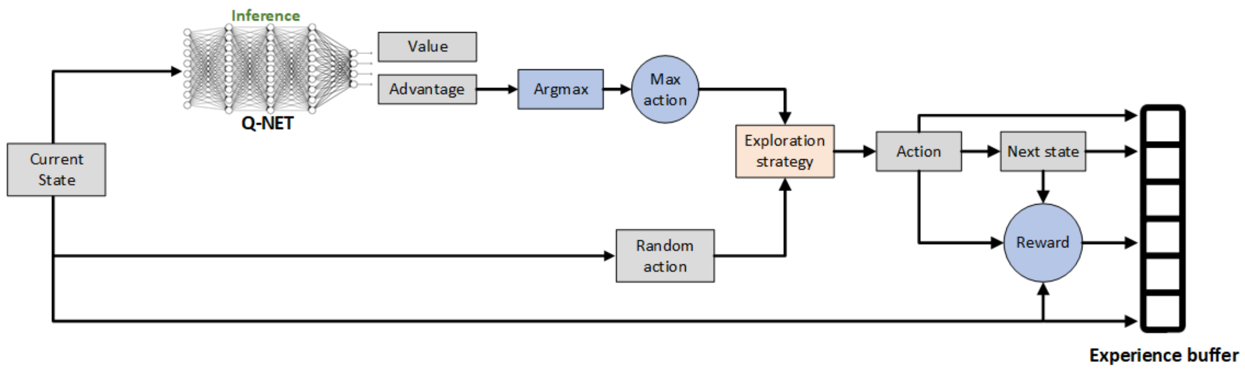


Figure 7. The decision-making process of the DDDQL.

DDDQL application to the decision-making engine of the NANCY ecosystem

DDDQL models are trained for either computation offloading, or for data offloading tasks. The details of each application scenario are depicted in Table 3 and are described below.

Computational offloading operations involve two distinct use cases: (i) the computational offloading of a service from the UEs to the Edge of the network; and (ii) the computational offloading of a service from the Cloud to the Edge of the network. NANCY addresses both scenarios by training the developed "self-attention" and "Dense" networks for each corresponding task. To achieve this, the reward functions are modified for each model in order to optimise a different use case, with different application constraints. This results in 4 models in total for the computational offloading operations: 2 "self-attention" and 2 "dense". For each model architecture, 2 model types are trained: 1 static and 1 adaptive. The terms static and adaptive refer to the reward types used by the agents. A static reward is calculated using a mathematical function that optimises a given problem. This function does not change over the course of the training process and thus, the agents converge quickly to the optimal policy. On the other hand, an adaptive reward scheme utilises several mathematical functions each one of which optimises a different problem. As a result, the way that the reward is calculated changes over the training course and thus, the agent converges slowly but the trained model accurately resembles the real-world conditions of a telecommunications network. Below we provide two examples of static and adaptive training for the DDDQL decision-engines.

Static training scenario: The model tries to optimise the Edge resource utilisation at any given time with respect to the network latency, by migrating services from the Cloud to the Edge of the network. The agent's rewards are analogous to the amount of computation resource utilisation of the Edge (the more, the better) and to the average communication latency between the UEs and the Edge (the less, the better). Nonetheless, the model should stop the service migration to the Edge, when the Edge is near its full computational capacity. The reward function for this scenario can be written as follows (3):

$$R(S,A) = \alpha * \frac{(Edge_{max})}{(Edge_{current})} - \beta * Latency_{avg} \qquad (3)$$

Adaptive training scenario: The model tries again to optimise the same problem, but as the services migrate to Edge, several constraints of the problem change. For example, if the Edge's resource utilization is almost 0%, then any service should be automatically transferred to the Edge. Otherwise, a priority hierarchy should be established among services, which changes according to the network latency and bandwidth. In this sense, multiple optimisation problems are formulated, each one for a specific network state. The reward function for this scenario can be written as follows (4):

$$
\begin{aligned}
R(S,A) &= \infty && , if \;\; 0\% \leq Edge_{current} \leq 20\% \\
R(S,A) &= 2\alpha * \frac{(Edge_{max})}{(Edge_{current})} - \beta * Latency_{avg} && , if \;\; 20\% < Edge_{current} \leq 60\% \\
R(S,A) &= \alpha * \frac{(Edge_{max})}{(Edge_{current})} - \beta * Latency_{avg} && , if \;\; 60\% < Edge_{current} \leq 90\% \qquad (4) \\
R(S,A) &= -\infty && , if \;\; 90\% < Edge_{current} \leq 100\%
\end{aligned}
$$

For the data offloading functionalities, NANCY foresees two scenarios, similar to the computation offloading: **(i)** the proactive data offloading of a file, or a set of files from the UEs to the Edge of the network; and **(ii)** the proactive data offloading of a of a file, or a set of files from the Cloud to the Edge of the network. Again, NANCY trains two model architectures, namely the "self-attention" and "Dense" networks for this scenario. This results in 4 models: 2 "self-attention" and 2 "dense". For each model architecture, NANCY provides a static and adaptive version of it, similar to the methodology followed for the computational offloading DDDQL models.

Table 3. DDDQL models for NANCY's decision-making engine.

Computation offloading		
DDDQL application scenarios	UE-Edge computation offloading	Cloud-Edge computation offloading
Model architectures	Self-attention	Dense
Number of trained models	2: (1 adaptive and 2 static)	2: (1 adaptive and 2 static)
Data offloading		
DDDQL application scenarios	UE-Edge data offloading	Cloud-Edge data offloading
Model architectures	Self-attention	Dense
Number of trained models	2: (1 adaptive and 2 static)	2: (1 adaptive and 2 static)

The models described above were trained using the Materna-Workload dataset [97]. Materna is a collection of real-world workload traces from various computer networks. The dataset contains information from telecommunication systems with various computing, memory and storage requirements and thus, it provides valuable data for the training process. Materna contains 5,329,729 data samples in total, which are adequate to train large DNN models, and 13 features as depicted in Table 4.

Table 4. Materna-Workload features and data types.

Features	Data types
Timestamp	Date-time
CPU cores	Integer (scalar)
CPU capacity provisioned	Double (MHZ)
CPU usage	Double (MHZ)
CPU usage	Double (%)
Memory capacity provisioned	Integer (KB)
Memory usage	Integer (KB)
Memory usage	Double (%)
Disk read throughput	Integer (KB/s)
Disk write throughput	Integer (KB/s)
Disk size	Double (GB)
Network received throughput	Double (KB/s)
Network transmitted throughput	Double (KB/s)

Before the deployment of the decision-making engine to the NANCY's Use Cases, real-world data stemming from NANCY's testbeds will be used to re-train and fine tune the DDDQL models. For this reason, the consortium will leverage the Transfer Learning (TL) technique illustrated in Figure 8. In TL, a model trained for a specific task is re-purposed for a similar task. TL enables the utilisation of pre-existing knowledge, i.e., a trained model, to produce a new model capable of performing well in a different scenario. Under this premise, data collected from the project's testbeds will formulate the training dataset. Then, a feature selection mechanism will choose the data features which are most appropriate for the re-training operation. During feature selection, subsets of the training dataset's features (variables and values) are selected according to their predicted contribution to the model's performance. In the sequel, the upper Q-NET's and Target-NET's layers will be frozen. Frozen layers do not utilise back-propagation and thus, their weights are not updated, even when data passes through them. Since the upper layers of a DNN model tend to learn general features, their existing knowledge will also be applicable to the new task. Afterwards, the re-training process will take place for a predefined number of epochs, to train the lower layers of the models that capture task-specific features. When this process is completed, the fine-tuning process will follow. During the fine-tuning, all the models' layers are unfrozen, and the learning rate is set to a very low value. This will enable the models to retrain all their layers, by leveraging the back-propagation operation to nudge their weights towards the right direction for convergence. The output of this process will be the final DDDQL models that will be used in the decision-making engine during the NANCY deployment scenarios.

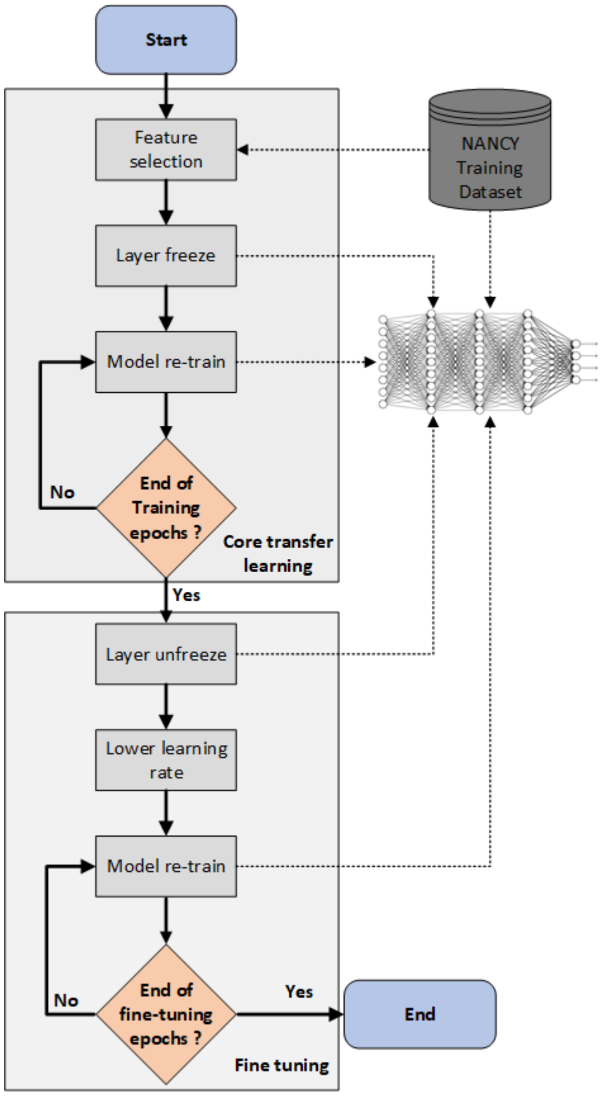


Figure 8. Transfer Learning workflow.

DDDQL simulation results

Table 5 depicts the hyperparameters of the DDDQL models, used for the in-lab evaluation process. The batch size, which refers to the number of samples used for a DNN training epoch, is equal to the experience buffer size for adaptive reward models, and half the experience buffer size for the static reward models. Adaptive reward models require more time to converge and thus, they benefit from using more samples for the DNN training, while the static reward DNNs can achieve good results even with a lower number of samples. Each DDDQL model is trained for 1,000 epochs and each epoch consists of 2,500,000 steps. The model replace rate represents the amount of DDDQL steps required to transfer the weights of the Q-NET to the target net, and it is set to 70. Gamma encapsulates the importance of future rewards for an agent. The higher the gamma (maximum is 1), the less important future rewards are for the agent's decision making. Lower gamma values complicate and slow the training process since the agent is looking far into the future to optimise its policy. For this reason, static reward models can afford a lower gamma, but adaptive reward models utilise higher gamma values. Epsilon designates the exploration strategy of the agent. More specifically, the value of epsilon represents the probability of an agent choosing random exploration over maximum advantage exploration. In our experiments, epsilon's initial value is 1, and for each step, it is reduced by 0.001 until it reaches 0.01. This means that for each training step, there is a small but measurable probability (between 1%-0.1%) for the agent to choose random exploration. That helps the agent to acquire new experiences, outside of the ones dictated by the current policy.

Table 5. DDDQL model hyperparameters.

Hyperparameter	Attention 1	Attention 2	Dense 1	Dense 2
DNN architecture	Self- attention	Self- attention	Dense	Dense
Reward type	Static	Adaptive	Static	Adaptive
Experience buffer size	128	128	128	128
Batch size	64	128	64	128
Environment steps	2,500,000	2,500,000	2,500,000	2,500,000
DDDQL epochs	1,000	1,000	1,000	1,000
Model replace rate (in steps)	70	70	70	70
Gamma	0.85	0.9	0.85	0.9
Initial epsilon	1	1	1	1
Epsilon decay	0.001	0.001	0.001	0.001
Minimum epsilon	0.01	0.01	0.01	0.01
DNN Loss function	MSE	MSE	MSE	MSE
DNN Optimiser	Adam	Adam	Adam	Adam
DNN learning rate	0.001	0.001	0.001	0.001
DNN dropout rate	0.2	0.2	0.2	0.2

Figure 9 illustrates the DNN model loss over the training epochs, for the implemented models. The loss is obtained after performing in-lab simulations for 1,000 training epochs. The static self-attention DNNs achieve the best loss (1.29), followed by the adaptive self-attention (2.04) and by the dense models (dense-static achieves 18.67 and dense-adaptive achieves 37.8). It is expected the static models to perform a bit better since their reward function increases their convergence rate compared to the adaptive models. On the other hand, adaptive models are expected to allocate resources more efficiently (via the data offloading and computation offloading operations) in real-world networks. Further, for most models, the loss converges after 600 or 800 training epochs since the agents require several traversals of the environment in order to fine tune their models.

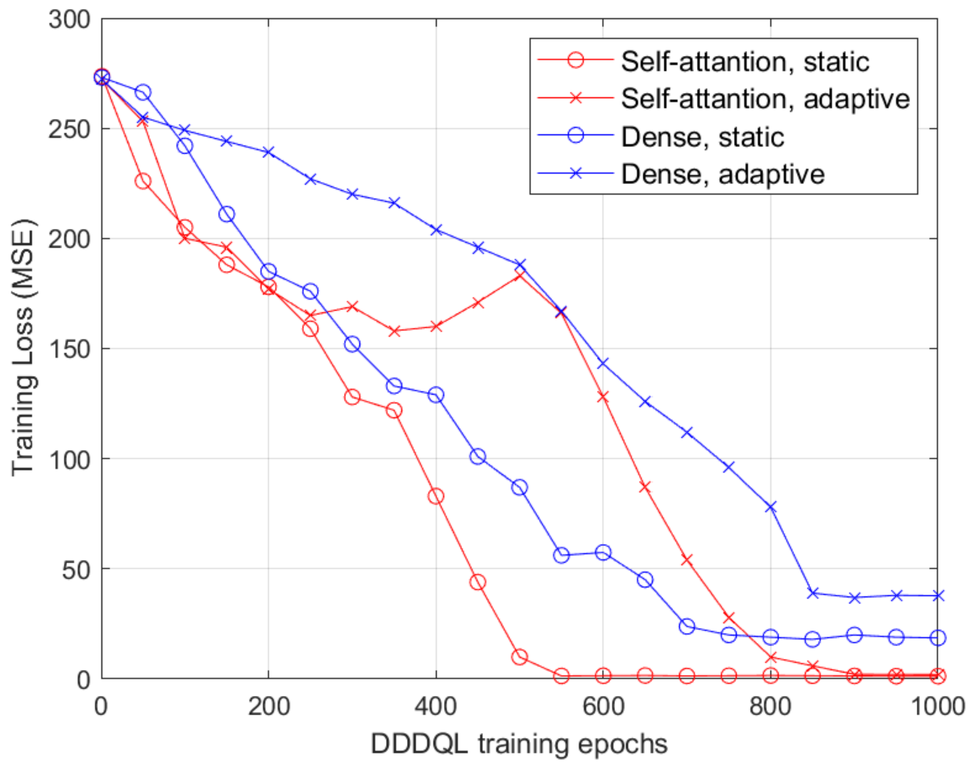


Figure 9. DNN model training loss.

Figure 10 demonstrates the cumulative rewards of the DDDQL decision-making engine over the training epochs. In all cases, the reward increases along with the training epochs thus, the agents learn to optimise their decision-making process. In this scenario, the best reward is achieved by the adaptive self-attention model (15.3), followed by the static self-adaptive model (11.13). The adaptive dense model obtains a cumulative reward of 8.88, while the static dense model obtains 6.55. It is evident that adaptive reward models outperform their static counterparts in terms of agent reward. For this reason, such models are expected to have a significant impact on the network's performance when computation and data offloading operations are considered.

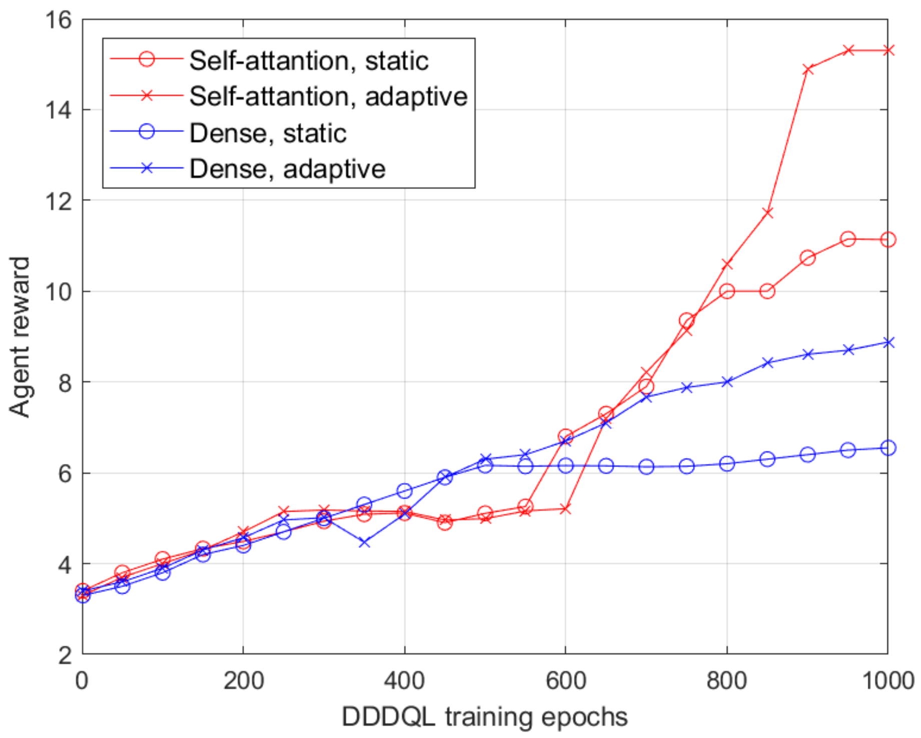


Figure 10. DDDQL agent reward.

Using the DDDQL framework described above, the decision-making engine of NANCY can decide "what to offload", as well as "where to offload" a file, a service, or some chunk of data. Below we describe the functionalities of each operation:

- **"What to offload"** refers to the specific data file (when data offloading is considered) or the specific service (when computation offloading is considered) which is offloaded. This operation decides the data stream(s), the file(s) and the service(s) which will be selected for the offloading process.
- **"Where to offload"** refers to the physical place, namely, UE, edge node or cloud, to which the offloaded element will migrate.

Table 6 below depicts the inputs and the outputs for each operation. For each decision, the DDDQL framework is invoked and is fed with the required data. When data offloading is considered, the decision-making engine requires as input the current state of the nodes, the data files to be offloaded (file names and sizes), along with the list of available nodes. When computation offloading is considered, the input describes the current state of the nodes, the computational and storage requirements of the services and the list of the available nodes. In all cases, the models generate a decision on which data/service(s) should be offloaded and where.

Table 6. The inputs and outputs of the decision-making engine, in terms of “what to offload” and “where to offload” choices.

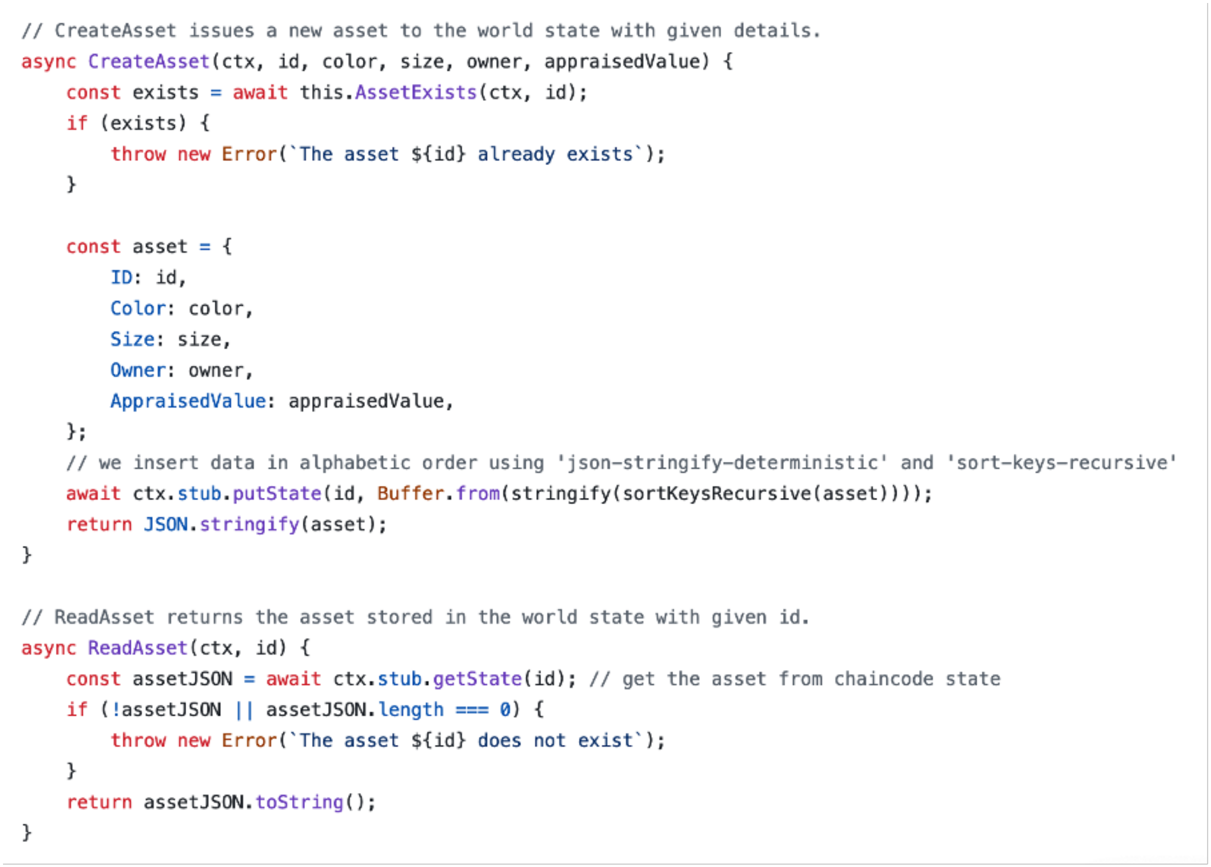
Data offloading	
DDDQL framework input	DDDQL framework output
Current state of UE and Edge nodes: *<CPU utilization, storage capacity, RAM utilization etc.>* **Data files to be offloaded:** <File_1, *MB; File_2, MB etc.>* **Edge nodes available:** *<Edge_1, Edge2, UE_1, UE_2 etc.>*	**What to offload:** Data files to be offloaded. E.g. <File_1, File_2> **Where to offload:** List of nodes where the offloading will take place E.g. < *Edge2, UE_1 etc.>*
Computation offloading	
DDDQL framework input	DDDQL framework output
Current state of UE and Edge nodes: *<CPU utilization, storage capacity, RAM utilization etc.>* **Service requirements:** <Service_1: *CPU, storage and RAM requirements;* Service_2: *CPU, storage and RAM requirements, etc.>* **Edge nodes available:** *<Edge_1, Edg2, UE_1, UE_2 etc.>*	**What to offload:** List of services to be offloaded. E.g. <File_1, File_2> **Where to offload:** List of nodes where the offloading will take place E.g. < *Edge2, UE_1 etc.>*

3.2.4 Blockchain

The NANCY Blockchain is based on Hyperledger Fabric 2.2. Its core elements are the ledger and the smart contracts. The ledger contains data about the current and historical state of different NANCY items, ranging from PQC public keys to DID material, among others. A smart contract defines the executable logic that generates new facts that are added to the ledger. Essentially, a smart contract is an application, and the many (or few) smart contracts on the NANCY Blockchain lay out the business model that governs all the interactions between the transacting parties, for example, the digital contracts between users and service providers. In this line, a smart contract can implement governance rules for diverse business objects, so that they can be automatically enforced when the smart contract is executed, while at the same time being more efficient than a manual human business process.

Hyperledger Fabric [98] users often use the terms smart contract and chaincode interchangeably. In general, a smart contract defines the transaction logic that controls the lifecycle of a business object contained in the world state. It is then packaged into a chaincode which is then deployed to a blockchain network. We can consider smart contracts as governing transactions, whereas chaincode governs how smart contracts are packaged for deployment. In other words, a smart contract is a domain-specific program which relates to specific business processes, whereas a chaincode is a technical container of a group of related smart contracts. In addition to this, only administrators deploy chaincode, and deploying a chaincode to a network makes all its smart contracts available to the organizations in that network.
A smart contract programmatically accesses two distinct pieces of the ledger – a blockchain, which immutably records the history of all transactions, and a world state that holds a cache of the current value of these states since it is the current value of an object that is usually required. Smart contracts primarily put, get and delete states in the world state, and can also query the immutable blockchain record of

transactions. At the heart of a smart contract is a set of transaction definitions. They are programmed in either JavaScript, Go, or Java.



```
// CreateAsset issues a new asset to the world state with given details.
async CreateAsset(ctx, id, color, size, owner, appraisedValue) {
    const exists = await this.AssetExists(ctx, id);
    if (exists) {
        throw new Error(`The asset ${id} already exists`);
    }

    const asset = {
        ID: id,
        Color: color,
        Size: size,
        Owner: owner,
        AppraisedValue: appraisedValue,
    };
    // we insert data in alphabetic order using 'json-stringify-deterministic' and 'sort-keys-recursive'
    await ctx.stub.putState(id, Buffer.from(stringify(sortKeysRecursive(asset))));
    return JSON.stringify(asset);
}

// ReadAsset returns the asset stored in the world state with given id.
async ReadAsset(ctx, id) {
    const assetJSON = await ctx.stub.getState(id); // get the asset from chaincode state
    if (!assetJSON || assetJSON.length === 0) {
        throw new Error(`The asset ${id} does not exist`);
    }
    return assetJSON.toString();
}
```

Figure 11. Issuing a new asset to the world state and returning an asset stored in the world state.

Associated with every chaincode is an endorsement policy that applies to all of the smart contracts defined within it [98]. An endorsement policy indicates which organizations in a blockchain network must sign a transaction generated by a given smart contract in order to declare that transaction as valid. It should be kept in mind, however, that all transactions, whether valid or invalid, are added to a distributed ledger, but only valid transactions update the world state. This ability of Fabric to enforce endorsement policies is reminiscent of the true world, where transactions must be validated by trusted organizations in a network (e.g. a bank or a public servant representing a regional authority).

In NANCY, the blockchain is used for various purposes, such as to store PQC public keys of, for example, UE wallets as well as storing DID-related data [99], and digital contracts containing providers, prices and SLA terms, which is particularly relevant for this report.

In a generic example, a service requester (e.g., a UE) would request a particular service, with an associated set of expected KPIs (e.g., latency, throughput, others…) represented in a static SLA (see section 3.2.1), from the NANCY network. This request would be handled by its domain orchestrator which may not be able to fulfil the demanded KPIs with its available resources, so it should forward this request to the inter-operator domain, concretely to the NANCY Marketplace. The NANCY Marketplace is a chaincode capable of providing lists of providers, prices, services, conditions and expected KPIs, similar to an online store. One must note that such providers (e.g., operators) must have previously registered themselves on the blockchain/Marketplace. It is precisely this type of organization the one that would have an endorser profile (see above). The Marketplace would also be able to receive pricing data from the smart pricing component, which is able to interact with the NANCY Blockchain, and eventually deliver sufficient data to the DAC for creating a digital contract (chosen provider, chosen service and KPIs, necessary conditions, price, etc.). This digital contract would be sent to the service provider and requester, who should sign it before it is added to the ledger and other enforcement and monitoring actions start. The Blockchain here acts as a central point of service request, bidding and registration, while other components of the system deal with service handling, enforcement and monitoring.

3.2.5 Marketplace

As mentioned in the previous section, the marketplace is a NANCY component based on blockchain which gathers all the information about the different stakeholders involved in NANCY, i.e., different operators, in a secure and transparent way. The blockchain-based marketplace can be considered secure because information recorded on the blockchain can never be removed, in this way, any modification to the existing information could be always traced and any uncontrolled changes would be detected. In addition, the blockchain-based marketplace can be considered transparent, as all the stakeholders will always have access to the same information. The marketplace includes two main components.

On the one hand, it includes a smart contract that allows the following functionalities: (i) It will store all the information related to the different operators and their available resources, providing required information to the smart pricing component to identify the most suitable price at a given time and also feeding the DAC with the required details about an operator required resources; (ii) it will also store the historical list of all the digital agreements signed in the NANCY environment, enhancing their security level.

In this sense, the marketplace allows the following functionalities:

- Register a new operator defining the specific details about it.
- Update any of the details of an already registered operator.
- Remove a registered operator.
- Apply filters by any property to obtain the list of operators fulfilling a specific condition.
- Retrieve any information about a given operator.

On the other hand, it includes a Blockchain monitor for enhanced usability and to allow human-in-the-loop solutions. It is based on the reception of Blockchain events every time any information of the marketplace is created or updated. This business logic is crucial to have a secure and trustable environment enabling resource sharing among stakeholders, therefore enabling cross-domain task offloading and data caching.

3.2.6 Smart pricing

The smart pricing module is a NANCY component developed by Eight Bells to cover both the user's as well as the provider's needs in certain offloading scenarios. In the scenario where a user ventures beyond the coverage area of their service provider, they might employ an alternative provider within range to manage the transfer of data or a given service itself. When faced with such situations, it is necessary to form a payment arrangement between the user's original provider and the alternative provider, following the parameters outlined in the SLA that was previously agreed upon with the user.
In areas with several capable data offloading providers, a Smart Pricing Policy coordinates an auction among these providers. Only providers who can fulfil the technical conditions specified in the service SLA are qualified to participate. These providers submit bids specifying the highest and lowest costs they are capable of incurring for the service. The successful bidder from this highly competitive auction will thereafter establish a new digital contract with the original supplier keeping the original technical terms in its fixed SLA part (see section 3.2.1).

The smart pricing module is essential in this procedure. The network receives data about qualifying providers and the ranges of their bids from the marketplace. The module employs a reverse auction theory that relies on reinforcement learning to ascertain the winning bid. The low complexity of the RL method enables the smart pricing module to function within a blockchain framework, guaranteeing the transparency and security of the auction. Subsequently, the proposal from the provider who emerged as the winner is processed within the marketplace to complete the necessary procedures and officially sign a new digital contract (just the business part, as the SLA terms remain unaltered).

After conducting a comprehensive analysis of the pertinent literature and prior research, it is evident that the most efficient method for enhancing bidding methods, comparable to those employed by the smart pricing module, is to utilize either Stackelberg games or reverse auction theory [100] [101] [102] [103]. Stackelberg games are particularly effective in instances when cost considerations are of utmost importance, notwithstanding the vast study conducted in the literature. Stackelberg games are frequently employed in scenarios when there exists only a small number of dominating players in an industry, referred to as oligopolies. These players prioritize cost reduction over fairness promotion and network expansion. Unlike Stackelberg Games, the reverse auction theory we employ has not undergone thorough investigation or practical application to meet the specific problem our approach tries to resolve. This preference is emphasized by the intrinsic limitations of other theories, such as Bertrand's, which depends on a shortage of bidders, and Cournot's, which largely emphasizes quantity rather than price considerations. Nevertheless, it is imperative to acknowledge that these favoured approaches do possess their constraints. Conversely, our ecosystem benefits from the reverse auction technique since it encourages participation from a broader range of participants, including smaller and individual MNOs. This inclusivity fosters a fair and just allocation of network resources. Moreover, it fosters the growth of a more extensive and robust network by incentivizing a diverse array of participants, thus mitigating the disadvantages of an oligopoly where a limited number of dominant companies could wield influence. The reverse auction theory advocates for sustainability and competitiveness by giving priority to fairness and network expansion, aligning with our long-term objectives.

Our smart pricing module utilises the reverse auction theory, implemented by deep reinforcement learning (DRL) agents, as an alternative to standard game theory methodologies. This approach is preferred for two primary reasons. DRL exhibits exceptional scalability, making it very compatible with the requirements of 5G dynamics within the context of the B-RAN. The rise in the number of possible participants, encompassing numerous users and Mobile Network Operators (MNOs), gives rise to a more intricate system, emphasizing the suitability of Deep Reinforcement Learning (DRL). Moreover, the adaptable characteristics of RL algorithms enable continuous modification in response to dynamic datasets, leading to a notable degree of flexibility when confronted with novel circumstances.

In more detail, the RL algorithm we are currently experimenting with starts by evaluating the highest cost associated with each candidate provider, creating a hierarchical ranking based on cost allocation. The provider with the most economically beneficial proposal is given the top rank, and subsequent providers are positioned accordingly. The iterative bidding process then begins, with companies strategically adjusting their cost offerings to gain a competitive advantage. This iterative process allows providers to continuously refine their pricing strategies to compete for maximum gain. Each bidding round gives an opportunity for the providers to improve their cost proposals and enhance their competitive position. The bidding procedure involves infinite iterations but stops when the algorithm determines that players cannot improve their bids further. In the first round, each candidate provider bids their maximum, and rankings are assigned, with the highest rank going to the provider with the lowest bid. From the second round onwards, each provider checks their current rank and decides their next bid to improve their position. The SPP uses a decrement strategy to reduce the bids of low-rank providers. When the bidding reaches a state where no player can improve their position, the program stops and announces the winner with the winning bid.

3.2.7 Digital Agreement Creator

The Digital Contract Creator (DAC) is a component developed as an out-of-the-box solution for creating and deploying digital agreements in the form of smart contracts. DAC is a fully containerized component that has its own internal orchestrator which receives inputs from the marketplace concerning e.g., stakeholders (provider ID, consumer ID), service, price, conditions, etc. For each set/series of inputs received by the marketplace, the DAC orchestrator creates individual/separate ad-hoc containers based on the respective received set of inputs.

During this initialization-creation process of the ad-hoc containers, the DAC orchestrator produces a unique identification number that passes as a kind of hash of the smart contract. The overall information (received inputs and hash number) is represented as a JSON file, and it is stored also into the DAC orchestrator before it passes to the ad-hoc container as a configuration file.

These ad-hoc containers, which contain the latter JSON configuration files, are the ones that are responsible for creating the actual smart contracts and they do have the capacity to deploy them inside the Hyperledger Fabric Blockchain. Stakeholders initiate interactions with the smart contract by sending transactions from their wallets to the contract address. All these functionalities - actions, such as setting the parameters of the DAC orchestrator, creating the JSON configuration file, creating the ad-hoc smart contract containers, checking its syntax validation and deploying to Hyperledger are exposed via respectively Software Oracles (Open APIs).

3.2.8 Edge-resource management

NANCY introduces an alternative virtualization technology at the edge of the network, exploiting the capability of the ARMv8 architecture to co-execute in an isolated manner multiple bare-metal compartments on the same hardware. The core technology of this virtualization solution is the cross-compartment virtio-loopback, which enriches the edge servers with the ability to host VNFs at the individual, bare-metal compartments. The cross-compartment virtio-loopback creates the necessary level of abstraction for the edge resources so that the VNFs can interface with the abstracted resources to perform their functionalities. The cross-compartment virtio-loopback builds upon and extends the virtio-loopback technology, which is an open-source hardware abstraction layer built for AGL. The virtio-loopback provides a framework that aims to bridge the gap between virtualized and bare-metal deployments. Specifically, the solution allows software stacks that are typically suitable for virtual deployments to be deployed also in a bare-metal fashion.

Specifically, the solution borrows some virtualization mechanisms from the virtualized world and brings them to the bare-metal world, by re-engineering some of the fundamental components of these mechanisms. As a result, the resources become abstracted also in the bare-metal environment, and a workload interfacing these abstracted (virtualized) resources can now be executed there.

In its internals, the virtio-loopback is based on two main virtualization protocols that it brings to the bare-metal world. From one side, the virtio protocol provides all the mechanics on how to expose virtual devices, e.g., block, net, GPU, console, etc., into a system. The virtual devices implemented with the virtio protocol follow some specific conventions that make interfacing with them through corresponding virtio drivers an easy task. On the other hand, the vhost-user protocol provides the mechanics on how to realize virtual devices, by interfacing with the actual underlying hardware.

In the open-source community as well as in the industry, there is wide support for both virtio drivers and vhost-user devices implementations. By bringing these implementations into bare-metal environments, the virtio-loopback enables to reuse existing efforts while at the same time, providing homogeneity among the virtualized and the bare-metal deployments. The virtio-loopback itself introduces two novel architectural components, to connect the virtio drivers with the vhost-user device implementations in the same system.

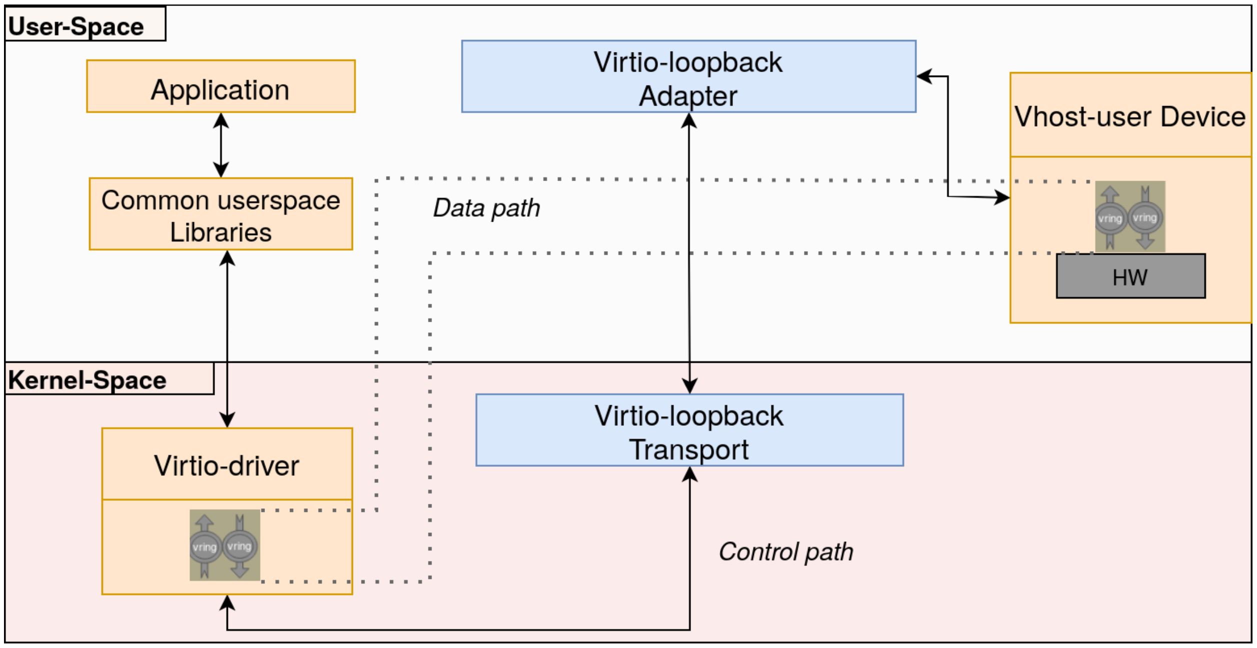


Figure 12. Virtio-loopback architecture in a bare-metal system.

Figure 12 demonstrates the overall virtio-loopback architecture deployed in a bare-metal system. Applications suited for virtualized deployments are able to interface with virtio devices, backed by usual virtio drivers. The novel components of virtio-loopback are the Transport and the Adapter, connecting the dots between the virtio drivers and the vhost-user devices. In detail, the virtio-loopback transport is a virtio transport which is similar to the virtio Memory-Mapped I/O (MMIO). While in the virtualized world, the role of the virtio transport is to invoke the hypervisor to carry out necessary actions, in the virtio-loopback case these actions are properly forwarded to the user-space virtio-loopback Adapter. As for the virtio-loopback Adapter, its role is to translate the incoming virtio requests to requests that the vhost-user device can understand. In essence, the virtio-loopback Adapter implements the vhost-user front-end to talk with the vhost-user back-end device.

The cross-compartment virtio-loopback extends the virtio-loopback solution, in a way it is ported across the bare-metal compartments of an ARMv8 edge server. The resource-rich, non-critical compartment provides the vhost-user device implementations that interface with the actual hardware. In this compartment, the virtio-loopback Adapter is ported unaltered. In the critical compartment, applications interfacing the virtio devices can be now supported. This way, VNFs can be hosted in this compartment and executed in an isolated and secure way. Regarding the virtio-loopback Transport, this component is split into two separate modules, following a client-server approach. Specifically, the client side generates virtio requests that the server side must serve. The whole architectural picture is depicted in Figure 13.

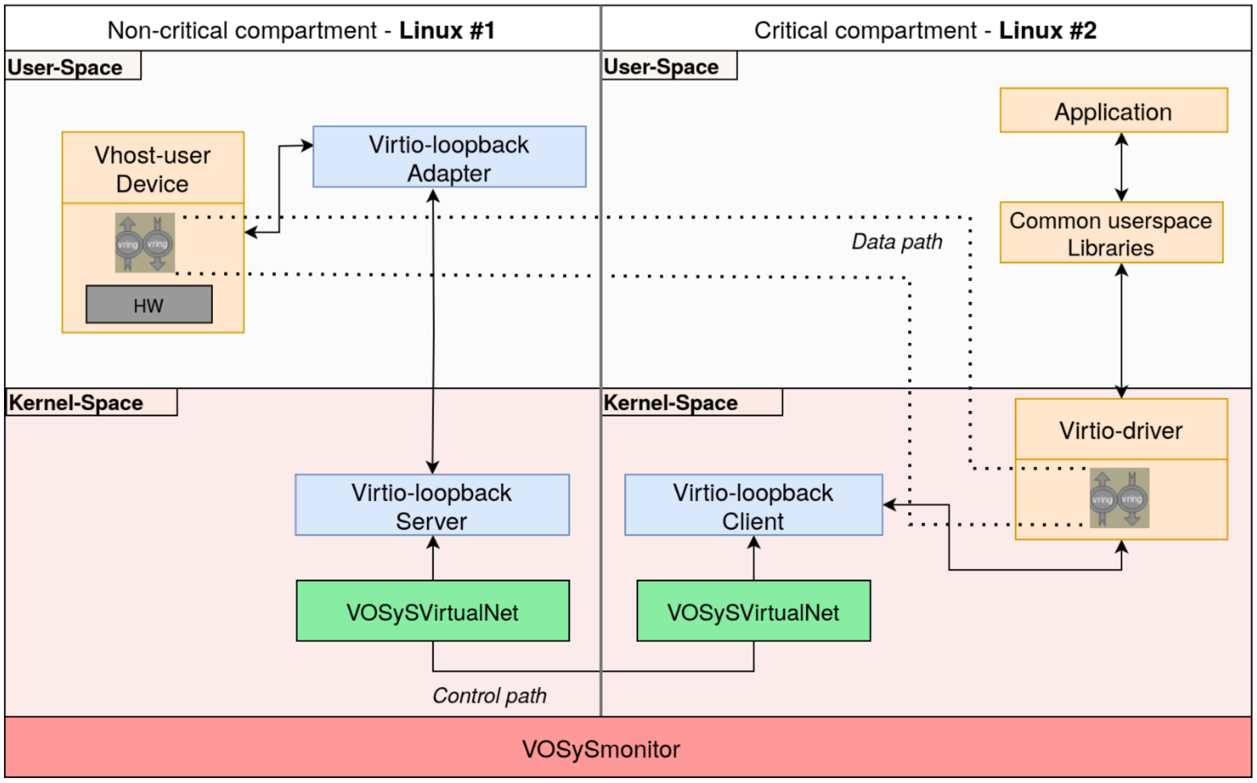


Figure 13. Cross-compartment virtio-loopback architecture.

Besides the virtio-loopback components that are adjusted in the cross-compartment solution, additional technologies such as VOSySmonitor and the VOSySVirtualNet are employed. Regarding VOSySmonitor, as already detailed in section 2.5, it plays the role of the system partitioner that consolidates the critical and the non-critical compartments. Regarding VOSySVirtualNet, it provides a communication link between the two compartments. The cross-compartment virtio-loopback interfaces with the VOSySVirtualNet layer for the exchange of control-plane virtio requests between the compartments. The mechanics of VOSySVirtualNet are simple: It retains a small shared-memory area for placing cross-compartment requests, which allows both sides to understand well how to consume. More specifically, when a compartment wants to notify the other side, VOSySVirtualNet passes the control to VOSySmonitor by issuing an SMC instruction and then VOSySmonitor generates an interrupt for the compartment to be notified. The data path does not follow this mechanism, instead, all the virtio data is placed in a specific shared memory region for the vhost-user device to process them directly. In order to place the virtio data of the critical compartment in this region, a bounce-buffering mechanism available from Linux is employed, making the data bouncing back-and-forth to their original location. This way, the critical data is protected and is not directly processed by the non-critical compartment.

Inter-compartment virtio-loopback technology demonstrates that it is feasible to deploy an abstract VNF that interacts with virtio devices in a critical compartment that is spawned on an ARMv8 architecture. In the context of computational offloading, the offloaded tasks with ultra-reliable and low-latency requirements can be deployed at the safety-critical bare-metal compartments of powerful ARM edge servers, enhancing the execution times and the security guarantees of the tasks. Considering data caching, as aforementioned, the use of compartment-based technology in this regard will be fully exploited in T4.3 so a detailed description of the related developments will be provided in D4.3

Besides, thanks to the work done in WP3, this technology will be configured and deployed dynamically, according to the requests forwarded by the orchestrator. Deliverable D3.4 will give more details about how a compartment can be instantiated and properly wired to its Adapter instance upon the orchestrator's request.

Resource Management of Linux nodes

Another important aspect of NANCY is the efficient resource management of computational resources. When dealing with Linux nodes, which are widespread, the SCHED_DEADLINE scheduling class of Linux [104] is an interesting option since it allows guaranteeing real-time constraints (CPU bandwidth and CPU worst-case latency) for virtualized software workloads (threads, VMs, and containers) and to guarantee temporal isolation between applications co-located on the same edge nodes and cores. In this context, a problem with key importance to avoid resource underutilization and guarantee timing constraints is to properly allocate the application corresponding to incoming service requests (e.g., from mobile IoT) to the available edge resources, possibly leveraging the cloud if needed (i.e., if the locally available resources are not enough to serve the application without compromising the real-time constraints). Next, we present algorithms to address this problem with a focus on SCHED_DEADLINE, with the purpose of optimizing the QoS-to-cost ratio of the applications.

System Model and Problem Definition

We consider a radio-enabled IoT-to-Edge-to-Cloud continuum where both Edge and Cloud nodes cooperate to offer services to *moving* IoT nodes by means of radio connectivity such as 5G. This work is focused on computing resource management in Edge and Cloud nodes to handle dynamic service requests issued by IoT nodes.

System definition. The system is composed of a dynamic set $\mathcal{C}$ of Cloud nodes and a dynamic set $\mathcal{E}$ of Edge nodes. A Cloud node is, in general, a virtual machine, while an Edge node can either be a whole computing (physical) platform or a virtual machine running on a shared cluster of Edge nodes (e.g., accessed under a rental/leasing scheme). Edge nodes are partitioned into n_d *domains*, where $\mathcal{E}_k$ denotes the Edge nodes within the k-th domain. The Edge nodes within the same domain share the same capabilities in terms of radio connectivity (e.g., they are connected to the same radio station), meaning that they can all communicate with the same set of IoT nodes. As IoT nodes move within the environment covered by the system, they can change the domain to which they connect. Furthermore, the fact that the set of Cloud nodes and the set of Edge nodes are dynamic means that Cloud nodes and Edge nodes can be powered on or off depending on the current service request by IoT nodes. For each domain, an orchestrator node is considered to determine a suitable allocation for each service requested by the IoT nodes. An overview of the system architecture is presented in Figure 14.

Workload model. Each node N_q includes M_q processor cores (either virtual or physical) and implements *partitioned* Earliest Deadline First (EDF) scheduling. A core of N_q is denoted with $c_{k,q} \in C_q$, where C_q denotes the set of cores in N_q. Both Cloud and Edge nodes run a dynamic set of applications composed of multiple tasks, each encapsulated within a reservation server (e.g., implemented by the SCHED_DEADLINE scheduler in Linux). Each application τ_i is modelled as a triplet (Q_i, P_i, m_i) and is served by m_i reservation servers, denoted with r_i, each allocated on a different core of the node, with the same budget Q_i and period P_i. The set of applications running on a node N_q is denoted by $\Gamma(N_q)$. The set of applications on each node is dynamic because applications can be activated or deactivated depending on the current service request by IoT nodes. The set of Edge and Cloud nodes is denoted with $\mathcal{E}$ and $\mathcal{C}$, respectively. Each node $N_q \in \mathcal{E} \cup \mathcal{C}$ is characterized by a corresponding set of allocated reservation servers $\mathcal{R}_q$. Similarly, the set of reservations allocated to a core $c_{k,q}$ of node N_q is denoted with $\mathcal{R}_{k,q}$.

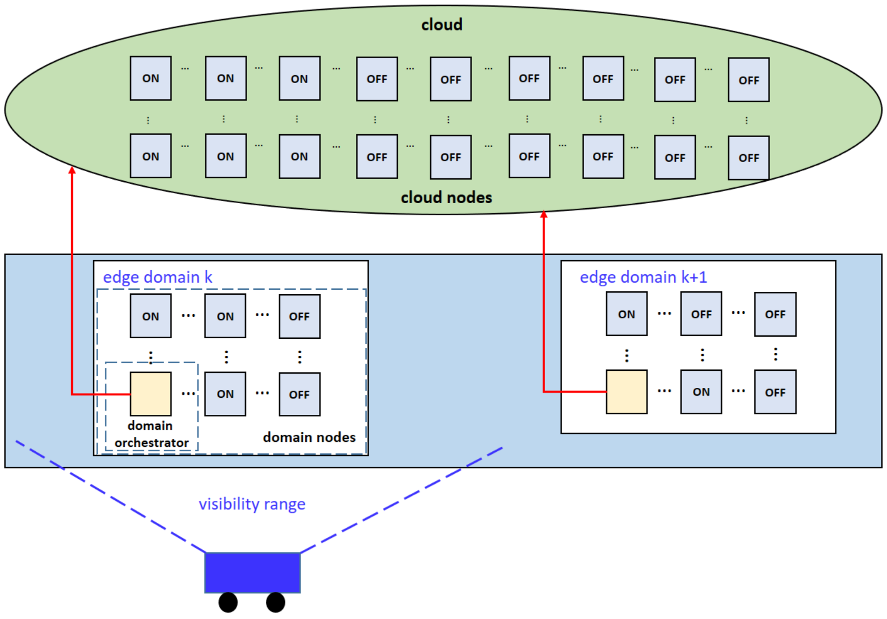


Figure 14 Offloading of computations from mobile nodes to the Edge/Cloud.

Dynamic scaling. IoT nodes dynamically issue service activation and de-activation requests to the domain to which they are connected. Each service S_i can be provided by the system with two quality-of-service (QoS) modes:

- *Standard*: The service S_i is entirely provided by an application τ_i running on an Edge node. This mode allows offering the service with the highest QoS as no extra latency is introduced due to the involvement of Cloud nodes.

- *Reduced*: The service S_i is cooperatively provided by both an Edge and a Cloud node, by means of two corresponding applications τ_i^E and τ_i^C running on the two nodes. τ_i^E, running on an Edge node, serves the functionality to implement connectivity with the IoT node that requested the service and offloads the compute-intensive part of the service to τ_j^C, which is instead running on a Cloud node. This mode offers the service with reduced QoS due to the extra latency introduced by the offloading process.

Satisfying a service activation request implies finding first an Edge node to which a new application τ_i can be allocated to, passing an *admission test*. If the allocation succeeds, the service is provided with Standard QoS. Otherwise, three options are available:

1. Offer the service with Standard QoS by powering on a new Edge node within the domain of interest (if available), which allows allocating τ_i to execute the service.

2. Offer the service with Reduced QoS by both finding an Edge node and a Cloud node to allocate τ_i^E and τ_i^C, passing two corresponding admission tests.

3. Offer the service with Reduced QoS by finding an Edge node to allocate τ_i^E and powering on a new Cloud node to allocate τ_i^C.

Conversely, satisfying service de-activation requests implies de-allocating the corresponding application(s). These events may in turn trigger a re-allocation of applications to either reduce the number of nodes powered on or improve the QoS of some services.

Service migration. Whenever an IoT node leaves the area covered by a domain D_a (since it exits from its visibility range) and enters the one covered by a domain D_b, it disconnects from D_a to connect to D_b. As a result, all services are activated by the IoT node on D_a need to be *migrated* to D_b. Migration consists of (i) issuing a service de-activation request to D_a and (ii) issuing a service activation (or re-activation) request to D_b. Note that the migration process can entail improving or reducing the QoS of a service.

System cost and overall QoS. Powering on a new Cloud or Edge node has a monetary cost, e.g., charged by the Cloud/Edge providers. This work considers the instantaneous cost of the nodes that are powered on at any time by means of two parameters: $\gamma^C(m)$, denoting the cost for each Cloud node with m cores, and $\gamma^E(m)$, denoting the cost for each Edge node with m cores. The overall system cost is hence given by (5):

$$\gamma^{\text{tot}} = \sum_{N_q \in \mathcal{E}} \gamma^E(M_q) + \sum_{N_q \in \mathcal{C}} \gamma^C(M_q). \qquad (5)$$

Cost is meant to be related to the overall QoS of the services activated by IoT nodes. Each service S_x is weighted by an *importance factor* $\beta_x \in [0,1]$ and assigned a QoS value W^{Std}, if running with Standard QoS, or W^{Red}, if running with Reduced QoS. Given a set of services $\mathcal{S}$ that are active at a certain time, the overall QoS at that time is defined as (6):

$$W^{\text{tot}} = \sum_{S_x \in \mathcal{S}} \beta_x \times W_x, \qquad (6)$$

where W_x is either equal to W^{Std} or W^{Red} depending on the QoS of the service.

Problem definition. The objective of this work is to design algorithms to handle dynamic scaling and service migration that aims at maximizing the QoS-cost ratio $W^{\text{tot}}/\gamma^{\text{tot}}$.

QoS and reservation parameters. Resource reservation servers in Linux are implemented in the SCHED_DEADLINE scheduling class and provide a valuable means to collocate lightweight applications or network functions in the same processing cores to avoid underutilizing the computing platform while still providing the required temporal isolation. Reservation servers are realized using the Constant Bandwidth Server reservation algorithm [1] and work by reserving Q_i time units every period P_i. Reservations are scheduled according to the EDF scheduling algorithm.

The budget and period parameters are equivalently mapped to two other parameters: the CPU bandwidth of the reservation $\alpha_i = Q_i/P_i$ and the worst-case service latency $\Delta_i = 2 \cdot (P_i - Q_i)$ of the workloads running inside the reservation.

Therefore, different QoS modes also correspond to different bandwidth and latency values since they correspond to potentially different budget and period pairs.

Algorithms

Admission test for Edge and Cloud

When a service request S_i is received, the domain-specific orchestrator node is in charge of finding a suitable allocation for the corresponding application, which allows satisfying an *admission test* required to satisfy the schedulability conditions.

The incoming service request S_i for τ_i is characterized by three tuples of reservation parameters. The first one, (Q_i, P_i, m_i), refers to the budget, period, and the number of cores required if the application τ_i runs at the edge in standard mode. The orchestrator then first tries to allocate in standard mode: if it cannot find a feasible solution in the current edge domain, then it tries to find a solution involving the degraded execution involving both the Edge and the Cloud.

SCHED_DEADLINE reservations are based on the EDF schedulability theory [105], which is capable of providing the guaranteed bandwidth and worst-case service latency to the workloads they serve if the sum of the requested bandwidths of all the reservations served on each core is less than or equal to one.

Since each application is composed of a triplet (Q_i, P_i, m_i), which maps to m_i reservations with bandwidth $\alpha_i = Q_i/P_i$, the application τ_i corresponding to the incoming service request S_i can be feasibly allocated in a given domain D_k if and only if (7):

$$\exists N_q \in D_k \mid \forall c_{k,q} \in C_q{}' \sum_{r_j \in \mathcal{R}_{k,q}} \alpha_j + \alpha_i \leq 1, \qquad (7)$$

with $C_q{}'$ being an arbitrary subset of C_q such that $\left|C_q{}'\right| = m_i$. Note that this holds assuming that the node was not overloaded before the additional service was requested.

When a feasible allocation cannot be found on the edge domain, a split solution that includes both the edge and the cloud can be considered. Otherwise, horizontal scaling at the edge can be applied, paying a monetary cost. When an edge-cloud split solution is applied, S_i is applied at a reduced QoS mode, which is characterized by two applications τ_i^E and τ_i^C with two sets of SCHED_DEADLINE parameters: $\left(Q_i^E, P_i^E, m_i^E\right)$ and $\left(Q_i^C, P_i^C, m_i^C\right)$. The first set of parameters refers to the part of the application running on the edge τ_i^E with the purpose of coordinating the work occurring in the Cloud. Clearly, the parameters $\left(Q_i^E, P_i^E, m_i^E\right)$ are characterized by a much lower budget and possibly a lower number of required cores, as the actual computation occurs in the Cloud. This allows favoring the allocation at the edge: indeed, if (7) is not satisfied for a given domain D_k, it could be satisfied with the reduced QoS mode. Therefore, the admission of the service in the reduced QoS mode, is composed of two applications τ_i^E and τ_i^C, must pass two admission tests: **(i)** the admission test for τ_i^E at the Edge, reported in (7) applied using $\alpha_i^E = Q_i^E / P_i^E$ instead of α_i; **(ii)** a similar admission test for τ_i^C in the Cloud, reported in (8):

$$\exists N_q \in \mathcal{C}: \forall c_{k,q} \in C_q{}' \sum_{r_i \in \mathcal{R}_{k,q}} \alpha_j^C + \alpha_i^C \leq 1, \qquad (8)$$

in which $\alpha_i^C = Q_i^C / P_i^C$. Clearly, the admission test in (8) may fail if the computing resources are not enough to accept the new application. In this case, it is either possible to turn on a cloud or edge node (at a financial cost) or to try to reallocate the pre-existing workloads in already-used nodes to allow the allocation of new applications. Both options are discussed later in this section.

Arrival of a new service request

When a service request arrives, the orchestrator needs to find the most suitable allocation (either in standard or reduced QoS mode).

Allocation on the Edge. The allocation produces first tries to allocate the application in standard mode considering the triplet (Q_i, P_i, m_i). To this end, the algorithm leverages the *residual per-core reservation bandwidth* as a metric, which is defined as (9):

$$U_{k,q} = 1 - \sum_{r_j \in \mathcal{R}_{k,q}} \alpha_j . \qquad (9)$$

First, the algorithm filters out all the nodes $N_q \in D_k$ that do not satisfy (7), i.e., which are characterized by $U_{k,q} < \alpha_i$ for more than $M_q - m_i$ cores. The remaining nodes are denoted in a set $D_k{}' \subseteq D_k$. If $D_k{}' \neq \emptyset$, the application can be allocated in standard QoS mode with at least one of the nodes in the current edge domain. However, there may be multiple choices for allocating τ_i, both in terms of nodes and cores within the nodes. Different choices lead to a different system configuration, which may be more or less prone to host the allocation of future workloads. We first consider the order of nodes in $D_k{}'$ by which to perform node selection, and we propose the following heuristics:

- **Max-Max Residual Bandwidth (MMRB).** $\max_{N_q \in D_k{}'} \max_{c_{k,q} \in C_q{}'} U_{k,q}$: this heuristic tries to select first the nodes with at least one residual reservation bandwidth $U_{k,q}$ (among those in the set $C_q{}'$ of those satisfying the admission test), with the rationale of privileging nodes having cores with less load.
- **Min-Min Residual Bandwidth (mmRB).** $\min_{N_q \in D_k{}'} \min_{c_{k,q} \in C_q{}'} U_{k,q}$: this heuristic tries to select first the nodes with at least one residual reservation bandwidth $U_{k,q}$ (among those in the set $C_q{}'$ of those satisfying the admission test), with the rationale of trying to fill the cores in a node.

Once a node is selected, cores are ordered according to standard bin-packing heuristics: **first-fit (FF)**, which considers cores in order of index; **best-fit (BF)**, which considers cores ordered by $U_{k,q}$; and **worst-fit (WF)**, which considers cores ordered by $1 - U_{k,q}$.

Mixed Edge-Cloud allocation with available nodes. If a service request cannot be served by using the standard QoS mode, there are two options: **(a)** Using the reduced QoS mode for the incoming service request, allocating two applications τ_i^E and τ_i^C, one at the edge and one in the cloud, respectively, or **(b)** reallocating another, already allocated, service S_j, possibly involving reducing its QoS mode, to make room for τ_i. Each option has its own advantages and disadvantages in comparison with the other one: **(a)** allows to not interfere with the execution of pre-allocated services, thus avoiding any reallocation delay; instead, **(b)** allows to achieve higher values of the QoS-per-cost ratio $W^{\text{tot}}/\gamma^{\text{tot}}$ by moving a pre-existing service with a lower importance factor to a reduced QoS mode. Applying (a) is straightforward if τ_i^E and τ_i^C pass the corresponding admission tests. Instead, (b) requires applying a heuristic: Several proposals are reported below. First, it is worth noting that not all the reallocations of other services necessarily cause a degradation of their QoS mode. We propose three reallocation schemes: the *intra-node reallocation*, the *intra-domain reallocation*, and the *edge-cloud split reallocation*. The first two reallocation methods do not involve a reduction of the QoS of S_j but are characterized by an increasing reallocation delay. Next, we present several heuristics to select the service S_j to be reallocated, among those in a given node $N_q \in D_k$. In their presentation, we use the parameter σ_j, which is equal to 1 if the reallocation method does not cause QoS degradation (intra-node and intra-domain) and to $(1 - \beta_j)$ otherwise (edge-cloud split).

- **Highest Bandwidth-Importance service (HBI).** This heuristic reallocates the service characterized by the highest $\alpha_j \times \sigma_j$ product, meaning that the service which is reallocated has high bandwidth (i.e., it is helpful to perform the reallocation since it frees a large amount of computation capacity) and low importance factor (considered only if reallocation causes QoS reduction), thus improving the value of the optimization function $W^{\text{tot}}/\gamma^{\text{tot}}$. This heuristic considers the bandwidth required by a service on each core but not the number of cores.

- **Highest Core-Importance service (HCI).** This heuristic reallocates the service characterized by the highest $m_j \times \sigma_j$ product, meaning that the service which is reallocated occupies many cores and has a low importance factor (considered only if reallocation causes QoS reduction), thus improving the value of the optimization function. This heuristic considers the number of cores required by S_i, but not its required bandwidth.

- **Highest Bandwidth-Core-Importance service (HBCI).** This heuristic combines the previous two by considering the product between the bandwidth and the number of cores. However, there is no guarantee that the overall bandwidth $\alpha_i \times m_i$ is distributed on enough cores to allow accommodating the incoming service request.

- **Highest Bandwidth-Importance, constrained to Cores, service (HBIcC).** This heuristic extends the first one by selecting the service with the highest $\alpha_j \times \sigma_j$ that occupies at least m_j cores. It overcomes the disadvantage of the first heuristic, but it is possible that no other reservation requires at least m_j cores exist, leading to a failure of the heuristic.

When a service S_j is selected to be reallocated, first the algorithm checks if there exists another set of cores within the same node that allows to let both S_j and S_i pass the admission test. This is done by first trying to allocate S_i and then S_j using the algorithms previously discussed. This is called *intra-node reallocation*. If intra-node reallocation fails, S_i is tentatively allocated on N_q after removing S_j. The procedure then succeeds if S_j passes the admission test on another edge node in the same domain D_k. This is called *intra-domain reallocation*. The third variant consists instead in reducing the QoS of S_j, checking if it is possible to allocate both τ_j^E and τ_j^C in such a way that both pass the admission test. This is called *edge-cloud split reallocation*.

The previous heuristics assume that the node to search for a service to be reallocated is given. Again, several heuristics can be used to make this choice. To this end, we consider the following heuristics:

- **Highest residual overall bandwidth first (HRB).** This heuristic selects first the node with the highest $\sum_{N_q \in \mathcal{E}_k} U_{k,q}$, with the rationale of privileging nodes with the largest residual bandwidth.

- **Largest service first (LSF).** This heuristic selects the node that has the largest $\alpha_i \times m_i$, with the rationale that selecting that node would allow gaining a considerable amount of bandwidth, useful to allocate S_i.

Powering on edge or cloud nodes. Powering on a new edge or cloud node comes with a monetary cost, and hence with a negative effect on the objective function $W^{\text{tot}}/\gamma^{\text{tot}}$. Nevertheless, allocating a new service increases the objective function value. Therefore, choosing whether it is more convenient to perform a reallocation, power on a new edge node, or power on a new cloud node is based on the variation in the objective function.

If a new edge node is powered on (i.e., rented) for serving S_i, then the new objective function value becomes:

$$\frac{W^{\text{tot}} + \beta_i \times W_i^{Std}}{\gamma^{\text{tot}} + \gamma^E\left(M_q\right)} \qquad (10)$$

If a new cloud node is rented for S_i, the new objective function value becomes:

$$\frac{W^{\text{tot}} + \beta_i \times W_i^{red}}{\gamma^{\text{tot}} + \gamma^C\left(M_q\right)} \qquad (11)$$

Clearly, powering on a new cloud node is a viable option only if the edge part of S_i in reduced mode (i.e., τ_i^E) can be allocated at the edge (i.e., passes the admission test).

Finally, reallocating a service S_j in the same domain $\mathcal{E}_k$ to the cloud to allow the new service request S_i to be served at the edge leads to the following tentative objective function value:

$$\frac{W^{\text{tot}} + \beta_i \times W_i^{std} - \beta_j \times W_j^{std} + \beta_j \times W_j^{red}}{\gamma^{\text{tot}}} \qquad (12)$$

Comparing these new objective function values allows for understanding the best choice for allocating the new service request.

The allocation procedure is summarized below.

Algorithm 1 allocate(S_i, D_k)

```
1:  function ALLOCATE(S_i, D_k, realloc)
2:      while true do
3:          if (AllocateEdge(S_i, D_k)) then
4:              return ALLOCATED
5:          if (realloc) then
6:              for each N_q in D_k do
7:                  if (intra-node realloc(S_i, N_q)) then
8:                      return ALLOCATED
9:              if (intra-domain realloc(S_i, D_k)) then
10:                 return ALLOCATED
11:         if (Eq. (6) $\geq$ Eq. (7) and Eq. (6) $\geq$ Eq. (8)) then
12:             if (scale_edge(S_i, D_k)) then
13:                 return ALLOCATED
14:         else if (Eq. (7) $\geq$ Eq. (6) and Eq. (7) $\geq$ Eq. (8)) then
15:             if (scale_cloud(S_i, D_k)) then
16:                 return ALLOCATED
17:         else if (realloc) then
18:             if (edge_cloud_split(S_i, D_k)) then
19:                 return ALLOCATED
```

Figure 15. Allocation procedure.

Exit of a service request

When a service S_i exits an edge domain D_k, the corresponding reservations are deallocated. Furthermore, it allows for improving the QoS by reallocating a service that is allocated in reduced QoS mode to standard QoS. To this end, we consider the four following heuristics:

- **Highest QoS improvement (HQ):** Selects the service S_j with the highest QoS improvement (subject to the importance), i.e., with the highest $\beta_j \times W_j^{std} - \beta_j \times W_j^{red}$.
- **Highest QoS improv. constrained to Cores (HQcC):** This heuristic extends HQ but only considers services that use fewer cores than those freed by S_i, i.e., m_i.
- **Highest QoS improv. constrained to Bandwidth (HQcB):** This heuristic extends HQ but only considers services that use less bandwidth than S_i, i.e., α_i.
- **Highest QoS improv. constrained to Cores and Bandwidth (HQcCB):** It combines the previous two heuristics by considering the constraints on both cores and bandwidth.

Services are considered only if allocated in reduced QoS mode. The selected service S_j is then tried to be allocated to the edge with the algorithms already discussed.

Furthermore, every time an edge or cloud node becomes empty due to a service exit, the corresponding physical machine can be turned off, reducing the overall cost γ^{tot}.

Service Migration

The migration of a service S_i from an edge domain D_k to another domain D_e can be managed as: an exit event from D_k and an arrival event to D_e. Note that a migration can produce an increment or a decrement of the overall QoS-to-Cost ratio, depending on the pre-existing services allocated to D_k and D_e, and the edge nodes in the two domains.

Runtime monitoring and vertical scaling

Since the computing nodes involved in the edge-cloud architecture are most commonly heterogeneous, it is possible that the initial Q_i estimate provided by the mobile device that requests the service can be either too large or too small, depending on the computing power of the nodes.

Therefore, the system must be coupled with a monitoring mechanism to report to the local orchestrator (of the edge domain) periodically if there is any amount of unused budget in the reservations of each service.

Given an estimate $\overline{Q}_i$ of the budget that is typically used in a reservation period, obtained by monitoring the system, a vertical scaling of the reservation can be performed. There are two cases: **(i)** the budget allocated to a reservation is too high, thus causing a underutilization of the node, or **(ii)** the budget allocated is not enough, not allowing to guarantee the CPU bandwidth and worst-case CPU latency constraints.

In case (i), the system can be configured with a threshold F, expressed as a percentage of the reservation period, such that if $\overline{Q}_i < F \cdot Q_i$ then a vertical scaling decision is taken by reducing the budget to the maximum budget that has been observed to be used within a past reservation period.

In case (ii), the budget is increased by a percentage G of the reservation period. Then, the admission tests are executed again, considering the same allocation. If the admission test succeeds, then the allocation is left unaltered. Otherwise, an exit event is followed by a new arrival event of the same service S_i are triggered so that a new feasible allocation can be found.

3.3 Offloading and user-centric caching workflow

Offloading and user-centric caching mechanisms aim to significantly improve the user experience through the correct preparation of the infrastructure and data necessary for fast and efficient user access to them; this implies caching data and applications in the network nodes that are closer to the user to reduce latency as well as using predictive models to anticipate the user's needs. One of the most critical issues to consider in the design of these mechanisms is user mobility. In a realistic scenario, users are not static, so such mechanisms must adapt to this behaviour.

To improve user experience and mitigate operational problems, NANCY proposes innovative offloading mechanisms, which leverage the user profile to efficiently and dynamically predict his/her future movements and requirements. This approach allows anticipating the needs and optimising the performance of the system in a proactive way. To achieve this, the service requests made by each user are continuously monitored and these data provide valuable information about the user's behaviour patterns and preferences. The collected data are used to feed reinforcement learning models, as explained in previous sections, to learn and make predictions about future requests and movements. In addition, these offloading mechanisms based on AI models permit adapting to network conditions in real time, which enables more efficient management of resources while optimising system performance. They also allow offloading and caching processes to be dynamically adjusted, facilitating system scalability and flexibility in resource management.

Caching mechanisms, on the other hand, involve storing data that are useful for a service or user on nodes close to the point of consumption of such data. This approach aims to reduce latency and improve system efficiency by ensuring that relevant information is quickly accessible when needed. In NANCY, a wide range of caching strategies are being explored, all focused on improving user experience. They consider factors such as user mobility, variations in network load and the individual specifications of each user profile.

To ensure the correct performance of the mechanisms mentioned above, SLAs are essential as they define the quality and reliability standards that must be met to guarantee user satisfaction and system performance. Therefore, SLA parameters express the minimum requirements for a service to be offered with an appropriate quality, taking into account: Availability, reliability, security, computing and network capacity. These parameters serve as a basis for the decision engines to trigger actions to maintain KPI levels in an adequate range. Considering the NANCY architecture, the workflow in Figure 16 is proposed to ensure the proper fulfilment of a given SLA when a request is received.

The procedure starts with the UE requesting a certain service to a service manager within the NANCY architecture (1). The service manager that receives this request, checks the UE subscription to make sure that it has access to such a service, and then builds the requirements for the service to be deployed and running without issues. These requirements are modelled in the form of an SLA (2) following the format proposed in Figure 16. Making use of the KPIs in the SLA, the service manager asks the AI blocks to determine where and how to deploy the requested service (3). The deployment decision is sent to the service orchestrator (4), which determines whether the service can be deployed in the local operator (ensuring the KPIs in the SLA) or not. In case the operator cannot fulfil the SLA, the service orchestrator forwards the service request to the marketplace to check if another operator can support providing the service (5). Once the final enforcement plan is ready (in the local operator or in an external one) (6), the corresponding service orchestrator sends the request to its attached resource orchestrator to configure each of the different domains participating in the service provisioning (7).

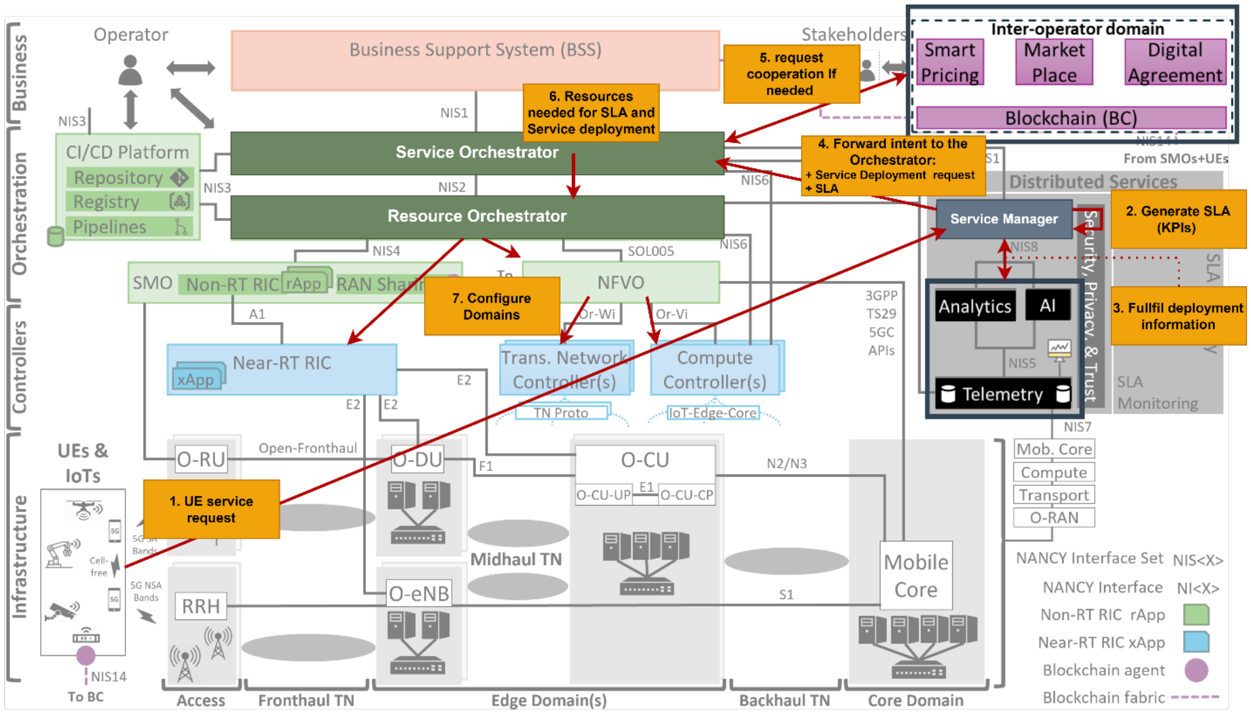


Figure 16. Offloading and caching workflow.

Specifically, in the case that the marketplace is used to handle the service deployment because the local operator is not able to fulfil the service's requirements indicated in the SLA, the marketplace module, which manages the publication and use of resources from other operators, handles the service provisioning request. Thus, the request is directed from the marketplace to the smart pricing module, which evaluates the available resources and prices from different network operators. Once the selected operator, with an associated price, is automatically determined by this block, it generates a new SLA and sends it to the service orchestrator of the operator that will provision the requested service. Additionally, the SLA is sent to the digital contract creator module, which incorporates extra business information into the SLA (see Figure 16), generating a digital agreement that is submitted and stored in the NANCY blockchain through the marketplace module. These inter-operator domain operations will be comprehensively explained in Deliverable 4.5: Smart Pricing Policies.

As part of a continuous SLA accomplishment strategy, analysis engines perform a constant evaluation of the system status with the data collected by monitoring the infrastructure components, verifying that the KPIs established in the SLA are being met. Besides this task, these modules are also responsible for measuring resource usage to maintain an optimal state of the necessary resources, ensuring correct system performance. It also detects anomalies in the behaviour of users and the system itself and establishes predictive models to anticipate situations that could affect service performance.

In order to achieve SLA compliance, caching and offloading mechanisms are key enablers. Both, when used simultaneously, enhance service delivery and the overall system status. Table 7 presents which parameters caching and offloading mechanisms have a direct impact.

Table 7 Benefits of offloading and caching mechanisms.

	Performance	**Availability**	**Capacity**	**Network**
Offloading	Reduces latency bringing services closer to the end user, and overall performance by distributing tasks or processing to more capable systems	By distributing workloads across multiple systems preventing a single point of failure	Reduces computational load on centralized servers and distributes it along the edge/fog	Reduces the overall network traffic, avoiding accessing centralized servers every time.
Caching	Reduces latency, especially for frequently accessed data or services	Cached content can still be accessed by services even though the original source is unavailable	Reduces the load on centralized servers	Reduces the overall network traffic, avoiding accessing centralized servers every time.

4. Offloading and Caching in NANCY Demonstrators and Testbeds

4.1 In-lab testbeds

4.1.1 Italian in-lab testbed

The Italian in-lab testbed will provide the environment to validate the "NANCY virtualization technology for deployments at the edge" (novel lightweight virtualization solutions for ARM edge servers, based on VOSyS monitor) and the technology for the analysis of different resources utilization (provided by SSS) (see Section 3.2.3). The scenario will focus on exploiting the ARM Trust-zone hardware-enforced isolation for hosting critical VNFs, like for example the "NANCY Telemetry framework" based upon the Linux kernel scheduling technology provided by SSS, based on the SCHED_DEADLINE scheduler, for the analysis of different resources utilization. The NANCY virtualization technology, relying on TrustZone, can provide hardware-isolated compartments ideal for running lightweight virtual machines where to offload network functions or mining workloads. This scenario will test and validate the technology proving the fact that a critical architectural function, like the "NANCY Telemetry framework", is running in a protected virtualization environment, isolated from other applications. The NANCY Telemetry framework is a key architectural function since, in this specific context, the collected metrics serve as an input data stream for further processing by the offloading and caching logic. Figure 17 shows an SK-AM69x Texas Instruments board integrated into the Italian in-lab testbed (ARMv8 edge server provided by Virtual Open Systems).

Besides, considering the technological advancements that have facilitated the development of immersive, interactive technologies, they are often related to video applications that are delay-intolerant to maintain users' quality of experience (QoE) and real-time response. These stringent latency requirements can be met by offloading the video streams to an edge server with adequate performance. In addition, the introduction of AI expands the potential of video analysis and makes autonomous monitoring and real-time threat detection possible. However, the effectiveness and efficiency of AI-powered video analysis applications rely heavily on the hardware infrastructure, specifically high-performance computing (HPC) architectures

To list just a few examples of applications that can benefit from the above-mentioned technological advancements, it is worth mentioning:

- Autonomous vehicles, that leverage multiple video cameras for environmental perception. Given the stringent latency requirements, these video streams need to be offloaded to the edge for the analysis of video streaming in real-time, e.g., for the automatic recognition of dangerous situations to alert both the drivers and pedestrians. This involves the real-time processing of video frames to extract valuable information.
- Augmented and virtual reality (AR/VR) applications, that demand stringent latency requirements and necessitate the processing of camera streams from headsets at the edge, with results relayed back to the headset for visual display.

If the above applications perform the video analysis providing AR/VR functionalities and run in the same edge where the video streams have been offloaded but in an environment with higher levels of security, this provides the system with a higher degree of reliability. This set the context in which the "NANCY virtualization technology for deployments at the edge" will be tested and validated in the Italian in-lab testbed, in relation to offloading and caching scenarios.

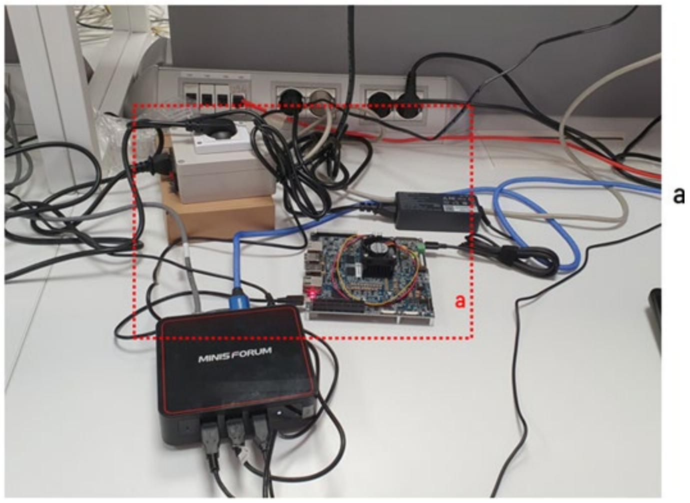


Figure 17. SK-AM69x Texas Instruments board in the Italian in-lab testbed.

4.1.2 Greek in-lab testbed

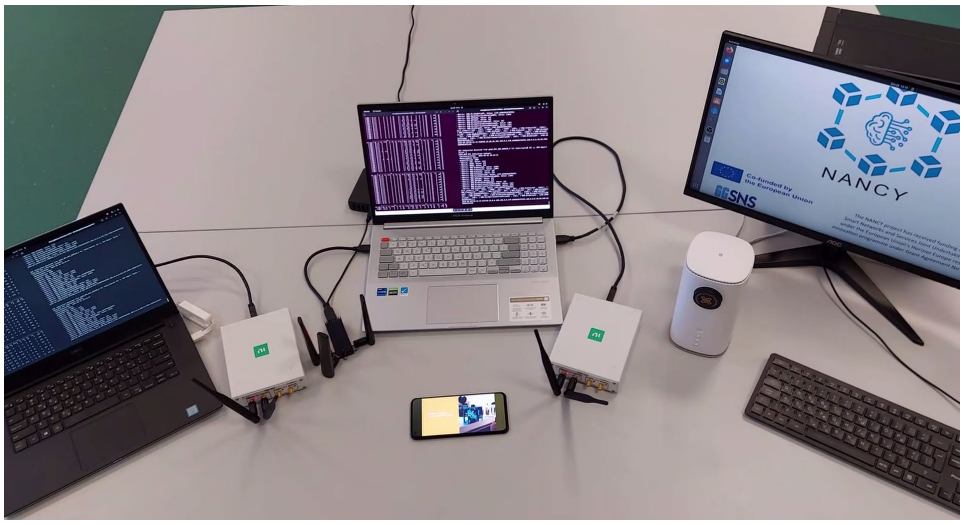


Figure 18: Main and Micro Operators in the Greek in-lab testbed

The Greek in-lab testbed (Figure 18) is focused on evaluating and showcasing several NANCY components in a coverage expansion scenario. The particular scenario involves three main entities, namely the main operator, the micro-operator, and the end devices. The micro-operator can provide connectivity to the end devices that have poor or no connection with the main operator. Furthermore, the micro-operator can also provide additional services to the end devices, such as offloading, security, and content caching, through the NANCY technologies.

The caching mechanism aims to prefetch and store content closer to the end devices, thereby minimizing the latency and reducing the network load. By profiling the devices, the suitable content can be pre-fetched from a content provider and cached either in the main operator's or the micro-operator's edge node.

This offloading scenario is particularly focused on the offloading of a video transcoding service. Video transcoding is a resource-intensive process that aims to convert a video from a particular format, scale, and quality/resolution to another, in order to be compatible with a particular UE or reduce the size of data that needs to be transported between the BS and the UE. In this scenario, the video transcoding application (e.g., VLC, FFmpeg, etc.) will be shipped as a containerised image that will receive as input the initial video and will output the transcoded video. The container will be managed by the Orchestrator, which will decide its appropriate deployment based on the available resources and the processing requirements.

4.2 Demonstrators

4.2.1 Italian demonstrator

The Italian Massive IoT testbed is designed to illustrate a fixed topology fronthaul network featuring direct connectivity. This configuration is intended to highlight particular facets and practical applications of IoT technology. Through this setup, the testbed aims to provide a comprehensive demonstration of how IoT systems can be efficiently integrated and managed within a network structure, emphasizing the potential for real-world deployment.

Within the NANCY testbed, a use case focused on machine-type communication (MTC) is demonstrated. The goal of the deployed applications is to exhibit a critical MTC scenario. In this setup, the indoor testbed connects IoT devices simulating traffic lights using 5G links. This configuration meets several stringent requirements: Providing reliable and resilient connectivity, enabling a dynamically changing number of IoT devices (simulated traffic lights) over time, guaranteeing low end-to-end latency and ensuring strict data privacy and security.

To meet low latency requirements, the IoT applications need to be offloaded and deployed within the MEC layer of the project's architecture. This strategic deployment ensures that the applications can operate with minimal delay, crucial for real-time IoT functionalities. The NANCY offloading capabilities will be demonstrated through a scenario where the edge layer is reconfigured. This reconfiguration aims to improve the high availability and resilience of the deployed IoT applications. The intent is to provide robust support for IoT operations, ensuring continuous and reliable performance even under dynamic or challenging conditions. This demonstration will showcase the effectiveness of offloading IoT applications to the edge layer by optimizing the deployment and maintaining QoS through adaptive management.

Concretely, the application to be offloaded to edge servers will be a real-time IoT solution designed for managing simulated traffic light devices, which have stringent low-latency requirements. This application is deployed using containerized virtualization technologies to ensure flexibility. The primary challenge is meeting the low-latency demands inherent to the real-time operation of the traffic light management system. Any delay in processing could result in inefficient traffic control, leading to congestion or even safety issues. Therefore, minimizing latency is crucial to maintaining the system's effectiveness and reliability. To address this challenge, the solution involves leveraging NANCY's advanced offloading capabilities. NANCY facilitates the intelligent deployment of the application at the network edge, significantly reducing latency by processing data closer to where it is generated. This edge computing approach ensures that the traffic management system can respond in real time, optimizing traffic flow and enhancing overall system performance.

4.2.2 Spanish demonstrator

The Spanish demonstrator constitutes a testbed composed of Unmanned Aerial Vehicles(UAVs) with Vehicular-to-everything (V2X) communication modules. These UAVs are capable of capturing video from the environment and streaming it. Moreover, one of these UAVs is additionally the entry point to the 5G network, as it has a 5G module in addition to the V2X module. Through this network of UAVs, and with each UAV being also packaged with limited computing resources in the form of a Raspberry Pi, the Spanish demonstrator will provide a resource pool capable of allocating offloaded tasks.

All in all, this demonstrator aims to demonstrate NANCY user-centric offloading mechanisms, with a video streaming use case in which the captured video goes through preprocessing steps prior to being stored. These preprocessing steps may be adapting the codec of the video for a streaming application or applying object recognition software to the video.

In addition to the aforementioned UAV resource pool, the Spanish demonstrator will employ MEC and cloud resources from the NANCY architecture as additional possible targets for deploying the VNFs that perform the different steps of video preprocessing. For this aim, the Spanish demonstrator will leverage the orchestration capabilities and decision engines from NANCY to determine the optimal allocation of these network functions according to the metrics indicated in Section 3.1 and to meet the KPIs of SLAs.

Spanish demonstrator extension in UMU testbed

The offloading and caching mechanisms are one of the key components to be shown in the extension of the Spanish demonstrator. Armed with a real-world B5G network, offloading and caching will be shown as part of the emerging trend of Digital Twins (DTs) and Trusted Execution Environments (TEE). Focusing on the first paradigm, offloading mechanisms will be shown in an enriched vehicular scenario. On-board units (OBU) will be used to establish a data session towards a DT, so-called virtual OBU (vOBU). This strategy permits to surrogate several services traditionally handled by the physical OBU to the vOBU: Data collection, session establishment with vehicular services, and portability through migration algorithms, among others. With this aim, vOBU units represent the digital counterpart of a moving vehicle, whose location changes frequently over time. vOBU units are the result of an offload request from part of the vehicle, which connection and data management towards a centralized V2X service wants to delegate to. As a result, the physical OBU releases the resource consumption (CPU, energy, radio...) attached to this process towards a resource-abundant part of the infrastructure as the MEC domain is.

The offload policy will be carefully handled, as latency in vehicular scenarios is a critical parameter to delimit. Once the offloading request is sent by the vehicle, a decision engine, following the KPIs of the service SLA, drives the orchestrator to deploy and connect the vOBU in the best worker node from a Kubernetes environment. AI-based location engines start monitoring the movement of the vehicle after its attachment to the network. Real-time radio measurements feed the engine to infer the vehicle's GPS position. When an upcoming coverage outage is anticipated by the decision engines, a migration mechanism moves the offloaded task to the best following edge node by following the workflow defined in Section 3.3. Therefore, the vehicle trajectory is inferred thanks to an ML-trained engine, capable of performing a correlation of the radio metrics measurement and a GPS coordinate. This movement pattern is employed to estimate the QoS variability and a potential SLA violation. More specifically, this is the triggering point for the migration process towards more adequate processing and connectivity-provider nodes, i.e., edge node and gNB. This offloaded function migration is done seamlessly, crossing the borders of different operators' infrastructure if needed, guaranteeing continuous SLA enforcement. Besides, this complex orchestration process to enable the intra- or inter-domain offloading implies anticipated data caching in the new nodes hosting the vOBU as the state of such a function should be transferred as well before launching the new instance of the vOBU in its new location.

TEE is the other big paradigm to be shown within the UMU in-lab testbed, and it is used to prove the benefits and potential of ARM-based OPTEE Trust Zones, critical cached content will be stored in such a protected environment, granting meticulous access permission to its content. The cached information will be used to streamline the service delivery by accelerating the authorization process, hence, the creation and enforcement of SLA. Privacy-preserving useful user information will be stored in the cache of close nodes. Which access, will be protected through the security capabilities provided by the OPTEE. Cached content is envisioned to include parameters such as UE's Pseudonym and access attributes, former SLAs could be also stored to avoid the complexity of the SLA creation. More details about these processes will be given in D4.3.

4.2.3 Greek demonstrator

AR/VR applications have revolutionized the user experience by delivering seamless, high-fidelity content over 5G networks. These applications are very challenging since they require very high bandwidth, intensive computational capability and extremely low latency. With the evolution of 5G technology and the commercial deployment of 5G networks, these challenges could be addressed with caching playing a significant role in optimizing AR/VR video transmission by enhancing network performance and efficiency. With 5G, cached data can be stored at the edge of the network, being closer to end-users facilitating reduced latency since delivery can be achieved faster and more efficiently.

Caching at RAN can significantly increase the performance of such AR/VR systems when compared to traditional techniques [106]. Regarding the Greek outdoor demonstrator, the AR/VR applications that will be used are managed by efficient caching algorithms both in terms of storage and content retrieval. Moreover, the use of Blockchain will provide enhanced security and transparency to these processes while preventing unauthorized access to the 5G network.

Besides caching, task offloading can also be implemented in OTE 5G networks for succeeding performance enhancement, latency reduction and optimizing resource utilization. The Greek demonstrator aims to achieve the required latency constraints and the quality of service (QoS) requirements for AR/VR applications. The possibilities of slicing and edge processing will be utilized to address these issues. More specifically, with slicing, it is possible to ensure the necessary resources for a specific application, avoiding delays due to reduced throughput (especially during video transmission) or low packet prioritization. On the other hand, edge computing brings application hosting from centralized data centres down to the network edge, closer to the users and the data generated by applications, leading to a subsequent reduction of latency.

Intensive computational tasks can be offloaded from user devices to edge servers where they will be processed, using AI mechanisms and algorithms. If needed, a collaboration between edge and cloud infrastructure, located in OTE premises, can be established in order to execute the offloaded tasks efficiently.

5. Conclusion and Outlook

This deliverable presented the main activities carried out in NANCY's T4.1, which is focused on the design and development of effective procedures and operational blocks to handle task offloading and user-centric data caching mechanisms. A well-defined end-to-end workflow has been designed to identify the involved components and their interactions within the NANCY architecture. Each of these components presents a specific role in this process that has been comprehensively described. In the intra-operator domain, it is worth noting the development of specific decision engines and resource management mechanisms that are able to cooperate thanks to the NANCY service and resource orchestrators with the aim of fulfilling the requirements established by the service' SLA. To this end, an SLA-based infrastructure management scheme has been defined and presented in this document. Considering that the fulfilment of the SLA is highly desirable (if not mandatory) in communication infrastructures, sometimes a serving operator cannot cope with such requirements; therefore, a solution should be found in the inter-operator domain. To this end, the marketplace, DAC, and smart pricing modules, supported by an underlying blockchain, can cooperate with each other to find such a desired solution besides the boundaries of the current service provider. This implies a series of both control and technical procedures to allow the migration of a service from one domain or operator to a different one. Although T4.1 finishes its activities at the time of submitting this report, its participants will continue working on the developments accomplished in this task to integrate and demonstrate them in the NANCY's demonstrators and in-lab testbeds.

Bibliography

[1] A. Akbari-Moghanjoughi, J. R. de Almeida Amazonas, G. Santos-Boada, and J. Solé-Pareta, "Service Level Agreements for Communication Networks: A Survey," 2023, arXiv:2309.07272.

[2] J. Goo, R. Kishore, H. R. Rao, and K. Nam, "The role of service level agreements in relational management of information technology outsourcing: An empirical study," MIS Quarterly, vol.33, no. 1, pp. 119-145, 2009.

[3] Benlarbi, S. (2006). Estimating SLAs Availability/Reliability in Multi-services IP Networks. In: Penkler, D., Reitenspiess, M., Tam, F. (eds) Service Availability. ISAS 2006. Lecture Notes in Computer Science, vol 4328. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11955498_3

[4] J. Gozdecki, A. Jajszczyk and R. Stankiewicz, "Quality of service terminology in IP networks," in IEEE Communications Magazine, vol. 41, no. 3, pp. 153-159, March 2003, doi: 10.1109/MCOM.2003.1186560.

[5] P. Leitner, A. Michlmayr, F. Rosenberg and S. Dustdar, "Monitoring, Prediction and Prevention of SLA Violations in Composite Services," 2010 IEEE International Conference on Web Services, Miami, FL, USA, 2010, pp. 369-376, doi: 10.1109/ICWS.2010.21.

[6] Yudhistira Nugraha, Andrew Martin. Understanding Trustworthy Service Level Agreements: Open Problems and Existing Solutions. International Workshop on Open Problems in Network Security (iNetSec), May 2017, Rome, Italy. pp.54-70.

[7] M. Barletta, M. Cinque, and C. Di Martino, "SLA-Driven Software Orchestration in Industry 4.0," IEEE Internet Things Mag., vol. 5, no. 4, pp. 136–141, Dec. 2022, doi: 10.1109/IOTM.001.2200216.

[8] Z. Tang, X. Zhou, F. Zhang, W. Jia and W. Zhao, "Migration Modeling and Learning Algorithms for Containers in Fog Computing," IEEE Transactions on Services Computing, vol. 12, no. 5, pp. 712-725, Sep.-Oct. 2019, doi: 10.1109/TSC.2018.2827070.

[9] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou and X. Shen, "Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach," IEEE Transactions on Mobile Computing, vol. 20, no. 3, pp. 939-951, 1 March 2021, doi: 10.1109/TMC.2019.2957804.

[10] T. Bahreini and D. Grosu, "Efficient Algorithms for Multi-Component Application Placement in Mobile Edge Computing," IEEE Transactions on Cloud Computing, vol. 10, no. 4, pp. 2550-2563, Oct.-Dec. 2022, doi: 10.1109/TCC.2020.3038626.

[11] N. -E. -H. Yellas, B. Addis, R. Riggio and S. Secci, "Function Placement and Acceleration for In-Network Federated Learning Services," 18th International Conference on Network and Service Management (CNSM), 2022, pp. 212-218, doi: 10.23919/CNSM55787.2022.9964625.

[12] Solozabal, R., Ceberio, J., Sanchoyerto, A., Zabala, L., Blanco, B., & Liberal, F. (2020). Virtual network function placement optimization with deep reinforcement learning. IEEE Journal on Selected Areas in Communications, 38(2), 292–303.

[13] Eyckerman, R., Reiter, P., Latre, S., Marquez-Barja, J., & Hellinckx, P. (2022). Application Placement in Fog Environments using Multi-Objective Reinforcement Learning with Maximum Reward Formulation. In Proc. of IEEE NOMS, pp. 1–6.

[14] Goudarzi, M., Palaniswami, M. S., & Buyya, R. (2021). A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments. arXiv:2110.12415.

[15] ETSI. (2022). Multi-access Edge Computing (MEC); Framework and Reference Architecture. European Telecommunications Standards Institute, Group Specification (GS) MEC 003, version 3.1.1.

[16] Brik, B., Frangoudis, P. A., & Ksentini, A. (2020). Service-Oriented MEC Applications Placement in a Federated Edge Cloud Architecture. In Proc. of IEEE ICC, pp. 1–6.

[17] Torres-Pérez, C., Coronado, E., Cervelló-Pastor, C., Camargo, J. S., & Siddiqui, M. S. (2023). Distributed Learning for Application Placement at the Edge Minimizing Active Nodes. In 2023 2nd International Conference on 6G Networking (6GNet), pp. 1–4.

[18] J. Zheng, P. Caballero, G. de Veciana, S. J. Baek, and A. Banchs, "Statistical multiplexing and traffic shaping games for network slicing," in 2017 15th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2017, pp. 1–8.

[19] F. Pervez, A. Sultana, C. Yang and L. Zhao, "Energy and Latency Efficient Joint Communication and Computation Optimization in a Multi-UAV-Assisted MEC Network," in IEEE Transactions on Wireless Communications, vol. 23, no. 3, pp. 1728-1741, March 2024, doi: 10.1109/TWC.2023.3291692.

[20] M. Chen and Y. Hao, "Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network," in IEEE Journal on Selected Areas in Communications, vol. 36, no. 3, pp. 587-597, March 2018, doi: 10.1109/JSAC.2018.2815360.

[21] X. Chen, "Decentralized Computation Offloading Game for Mobile Cloud Computing," in IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 4, pp. 974-983, 1 April 2015, doi: 10.1109/TPDS.2014.2316834

[22] F. Wang, J. Xu, X. Wang and S. Cui, "Joint Offloading and Computing Optimization in Wireless Powered Mobile-Edge Computing Systems," in IEEE Transactions on Wireless Communications, vol. 17, no. 3, pp. 1784-1797, March 2018, doi: 10.1109/TWC.2017.2785305.

[23] S. Bi and Y. J. Zhang, "Computation Rate Maximization for Wireless Powered Mobile-Edge Computing With Binary Computation Offloading," in IEEE Transactions on Wireless Communications, vol. 17, no. 6, pp. 4177-4190, June 2018, doi: 10.1109/TWC.2018.2821664.

[24] D. H. Abdulazeez and S. K. Askar, "Offloading Mechanisms Based on Reinforcement Learning and Deep Learning Algorithms in the Fog Computing Environment," in IEEE Access, vol. 11, pp. 12555-12586, 2023, doi: 10.1109/ACCESS.2023.3241881.

[25] Z. Ning, P. Dong, X. Wang, J. J. P. C. Rodrigues, and F. Xia, "Deep Reinforcement Learning for Vehicular Edge Computing: An Intelligent Offloading System," ACM Trans. Intell. Syst. Technol., vol. 10, no. 6, Art. no. 60, Nov. 2019, pp. 1-24, doi: 10.1145/3317572.

[26] S. Gong, Y. Xie, J. Xu, D. Niyato and Y. -C. Liang, "Deep Reinforcement Learning for Backscatter-Aided Data Offloading in Mobile Edge Computing," in IEEE Network, vol. 34, no. 5, pp. 106-113, September/October 2020, doi: 10.1109/MNET.001.1900561.

[27] "PREDICT-6G – Towards a deterministic 6G network: reliable,time sensitive and predictable". [Online.] Available: https://predict-6g.eu

[28] "6G BRAINS". [Online.] Available: https://6g-brains.eu

[29] "Home - 5G-CLARITY Project". [Online.] Available: https://5gclarity.com/

[30] "AI@EDGE – A secure and reusable Artificial Intelligence platform for Edge computing in beyond 5G Networks". [Online.] Available: https://aiatedge.eu/

[31] S. Sheng, P. Chen, Z. Chen, L. Wu and H. Jiang, "Edge Caching for IoT Transient Data Using Deep Reinforcement Learning," IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society, Singapore, 2020, pp. 4477-4482, doi: 10.1109/IECON43393.2020.9255111.

[32] S. Wang, Q. Zhu, H. Huang, Y. Lei, W. Zhan and H. Duan, "Deep Reinforcement Learning Based Real-Time Proactive Edge Caching in Intelligent Transportation System," 2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), Chengdu, China, 2023, pp. 162-166, doi: 10.1109/ICCCBDA56900.2023.10154769.

[33] W. Yang and Z. Liu, "Efficient Vehicular Edge Computing: A Novel Approach With Asynchronous Federated and Deep Reinforcement Learning for Content Caching in VEC," in IEEE Access, vol. 12, pp. 13196-13212, 2024, doi: 10.1109/ACCESS.2024.3355462.

[34] H. Wu, A. Nasehzadeh and P. Wang, "A Deep Reinforcement Learning-Based Caching Strategy for IoT Networks With Transient Data," in IEEE Transactions on Vehicular Technology, vol. 71, no. 12, pp. 13310-13319, Dec. 2022, doi: 10.1109/TVT.2022.3199677.

[35] S. Sharma and S. K. Peddoju, "IoT-Cache: Caching Transient Data at the IoT Edge," 2022 IEEE 47th Conference on Local Computer Networks (LCN), Edmonton, AB, Canada, 2022, pp. 307-310, doi: 10.1109/LCN53696.2022.9843211.

[36] X. He, K. Wang, H. Lu, W. Xu and S. Guo, "Edge QoE: Intelligent Big Data Caching via Deep Reinforcement Learning," in IEEE Network, vol. 34, no. 4, pp. 8-13, July/August 2020, doi: 10.1109/MNET.011.1900393

[37] Y. Yu, S. Wu, S. Shao, J. Chen and X. Chen, "A Multi-Agent Deep Reinforcement Learning based Cooperative Edge Data Caching Approach," 2023 24st Asia-Pacific Network Operations and Management Symposium (APNOMS), Sejong, Korea, Republic of, 2023, pp. 294-297.

[38] M. Zhang, Y. Jiang, F. -C. Zheng, M. Bennis and X. You, "Cooperative Edge Caching via Federated Deep Reinforcement Learning in Fog-RANs," 2021 IEEE International Conference on Communications Workshops (ICC Workshops), Montreal, QC, Canada, 2021, pp. 1-6, doi: 10.1109/ICCWorkshops50388.2021.9473609.

[39] L. Zhao, Y. Ran, H. Wang, J. Wang and J. Luo, "Towards Cooperative Caching for Vehicular Networks with Multi-level Federated Reinforcement Learning," ICC 2021 - IEEE International Conference on Communications, Montreal, QC, Canada, 2021, pp. 1-6, doi: 10.1109/ICC42927.2021.9500714.

[40] Royo, C. B., & Vidal, F. G. (2008, December). E-marketplaces: A Telecom Operator Perspective. In *2008 International Conference on Computational Intelligence for Modelling Control & Automation* (pp. 506-511). IEEE.

[41] Salvaggio, J. L., & Nelson, R. A. (2019). Marketplace vs. Public Utility Models for Developing Telecommunications and Information Industries. In *Mediation, Information, and Communication* (pp. 253-266). Routledge.

[42] Bhat, S., Udechukwu, R., Dutta, R., & Rouskas, G. N. (2017). Network service orchestration in heterogeneous 5G networks using an open marketplace. *IET Networks*, *6*(6), 149-156.

[43] Paris, S., Martisnon, F., Filippini, I., Chen, L., and Martignon, F. (2013, April). A bandwidth trading marketplace for mobile data offloading. In *2013 Proceedings IEEE INFOCOM* (pp. 430-434). IEEE.

[44] Tkachuk, R. V., Ilie, D., Tutschku, K., & Robert, R. (2021). A survey on blockchain-based telecommunication services marketplaces. *IEEE Transactions on Network and Service Management*, *19*(1), 228-255.

[45] S. Hosny, F. Alotaibi, H. El Gamal and A. Eryilmaz, “Towards a mobile content marketplace,” in *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, IEEE, 2015, pp. 675-679.

[46] Y. Jiao, P. Wang, D. Niyato and Z. Xiong, “Social welfare maximization auction in edge computing resource allocation for mobile blockchain,” in *2018 IEEE international conference on communications (ICC)*, IEEE, 2018, pp. 1-6.

[47] K. Liu, X. Qiu, W. Chen, X. Chen and Z. Zheng, “Optimal pricing mechanism for data market in blockchain-enhanced internet of things,” *IEEE Internet of Things Journal,* vol. 6, no. 6, pp. 9748-9761, 2019.

[48] M. Flamini and M. Naldi, “Optimal Pricing in a Rented 5G Infrastructure Scenario with Sticky Customers,” Future Internet, vol. 15, no. 2. MDPI AG, p. 82, Feb. 19, 2023. doi: 10.3390/fi15020082

[49] Kim, D.H.; Ndikumana, A.; Kazmi, S.A.; Kim, K.; Munir, M.S.; Saad, W.; Hong, C.S. Pricing Mechanism for Virtualized 489 Heterogeneous Resources in Wireless Network Virtualization. In Proceedings of the 2020 International Conference on Information 490 Networking (ICOIN), Barcelona, Spain, 2020; pp. 366–371.

[50] S. K. A. Kumar, D. Crawford and R. Stewart, “Pricing Models for 5G Multi-Tenancy using Game Theory Framework,” IEEE Communications Magazine, vol. 62, no. 5, May 2024.

[51] C. Ene, “Smart contracts - the new form of the legal agreements,” Proc. Int. Conf. Bus. Excell., vol. 14, no. 1, pp. 1206–1210, Jul. 2020, doi: 10.2478/picbe-2020-0113.

[52] S. Li, L. Da Xu, and S. Zhao, “5G Internet of Things: A survey,” J. Ind. Inf. Integr., vol. 10, pp. 1–9, Jun. 2018, doi: 10.1016/j.jii.2018.01.005.

[53] “Smart Contracts and Chaincode — Hyperledger Fabric Docs main documentation”. [Online.] Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/smartcontract/smartcontract.html#smart-contract

[54] https://www.etsi.org/deliver/etsi_gr/PDL/001_099/019/01.01.01_60/gr_PDL019v010101p.pdf

[55] H. Hawilo, A. Shami, M. Mirahmadi and R. Asal, "NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC)," in *IEEE Network*, vol. 28, no. 6, pp. 18-26, Nov.-Dec. 2014, doi: 10.1109/MNET.2014.6963800.

[56] W. Attaoui, E. Sabir, H. Elbiaze and M. Guizani, "VNF and CNF Placement in 5G: Recent Advances and Future Trends," IEEE Transactions on Network and Service Management, vol. 20, no. 4, pp. 4698-4733, Dec. 2023, doi: 10.1109/TNSM.2023.3264005.

[57] Eder, Michael. "Hypervisor-vs. container-based virtualization." *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)* 1 (2016).

[58] R. Rajkumar, C. Lee, J. Lehoczky y D. Siewiorek, «A resource allocation model for QoS management,» de Proceedings Real-Time Systems Symposium, 1997.

[59] R. Rajkumar, C. Lee, J. Lehoczky y D. Siewiorek, «Practical Solutions for QoS-based Resource Allocation Problems,» de Proceedings 19th IEEE Real-Time Systems Symposium, 1998.

[60] C. Lee, J. Lehoczky , R. Rajkumar, and D. Siewiorek, «On quality of service optimization with discrete QoS options,» Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium, 1999.

[61] G. Saurav, R. Rajkumar, J. Hansen y J. Lehoczky, «Scalable resource allocation for multi-processor QoS optimization,» de *23rd International Conference on Distributed Computing Systems*, 2003.

[62] A. Mok, X. Feng y D. Chen, «Resource partition for real-time systems,» de Proceedings Seventh IEEE Real-Time Technology and Applications Symposium, 2001.

[63] A. Easwaran, I. Shin y I. Lee, «Optimal virtual cluster-based multiprocessor scheduling,» Real-Time Systems, vol. 43, nº 1, pp. 25--59, 2009.

[64] E. Bini, M. Bertogna y S. Baruah, «Virtual Multiprocessor Platforms: Specification and Use,» de 2009 30th IEEE Real-Time Systems Symposium, 2009.

[65] I. Shin y I. Lee, «Periodic resource model for compositional real-time guarantees,» de 24th IEEE Real-Time Systems Symposium, 2003.

[66] Y. Li, L. Xuan Phan y B. T. Loo, «Network functions virtualization with soft real-time guarantees,» de IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016.

[67] V. Struhar, S. Craciunas, M. Ashjaei, M. Behnam y A. Papadopoulos, «Hierarchical Resource Orchestration Framework for Real-Time Containers,» ACM Trans. Embed. Comput. Syst., 2024.

[68] D. Casini, A. Biondi y G. Buttazzo, «ask splitting and load balancing of dynamic real-time workloads for semi-partitioned EDF,» IEEE Transactions on Computers, 2021.

[69] P. Lucas, K. Chappuis, M. Paolino, N. Dagieu, and D. Raho, ”Vosysmonitor, a low latency monitor layer for mixed-criticality systems on ARMv8-A.” 29th Euromicro Conference on Real-Time System (ECRTS 2017), 2017, p. 1-18.

[70] NANCY Consortium, “NANCY Architecture Design”. [Online.] Available: https://nancy-project.eu/wp-content/uploads/2024/02/NANCY_D3.1_NANCY_Architecture_Design_v1.0.pdf

[71] O-RAN Alliance. [Online]. Available: https://www.o-ran.org/

[72] ETSI, ”ETSI GS NFV SOL005 v3.5.1, Release 3; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point”, 2021. [Online.] Available: https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/03.05.01_60/gs_nfv-sol005v030501p.pdf

[73] Open Source MANO, «OSM SOL005 Interface», 2022. Available: https://osm.etsi.org/gitweb/?p=osm/SOL005.git

[74] Linux Foundation, ”Aether” . [Online.] Available: https://aetherproject.org/

[75] ETSI, ”Open-Source MANO (OSM)” . [Online.] Available: https://osm.etsi.org/

[76] ETSI, ”ETSI”, 2024. [Online.] Available: https://www.etsi.org/.

[77] 3GPP, «3GPP», 2024. [Online.] Available: https://www.3gpp.org/.

[78] TMForum, ”Open APIs” . [Online.] Available: https://www.tmforum.org/oda/open-apis/

[79] NANCY D6.1 “B-RAN and 5G End-to-end Facilities Setup”, August 2024.

[80] ETSI OSM NB API featuring ETSI NFV SOL005, Available: https://forge.etsi.org/swagger/ui/?url=https%3A%2F%2Fosm.etsi.org%2Fgitweb%2F%3Fp%3Dosm%2FSOL005.git%3Ba%3Dblob_plain%3Bf%3Dosm-openapi.yaml%3Bhb%3DHEAD

[81] TMForum, “TMF633 Service Catalog Management API v4.0.0”, 2021, Available: https://www.tmforum.org/resources/standard/tmf633-service-catalog-api-user-guide-v4-0-0/

[82] TMForum, “TMF641 Service Ordering Management API v4.1.1”, 2021, Available: https://www.tmforum.org/resources/specifications/tmf641-service-ordering-management-api-user-guide-v4-1-1/

[83] TMForum, “TMF638 Service Inventory Management API v4.0.1”, 2020, Available: https://www.tmforum.org/resources/specification/tmf638-service-inventory-api-user-guide-v4-0-0/

[84] TMForum, "TMF657 Service Quality Management API v4.0.1," 2020, Available: https://www.tmforum.org/resources/specification/tmf657-service-quality-management-api-user-guide-v4-0/

[85] TMForum, "TMF623 SLA Management API v1.0.1," 2015, Available: https://www.tmforum.org/resources/interface/tmf623-sla-management-api-rest-specification-r14-5-0/

[86] TMForum, "TMF632 Party Management API v5.0.0", 2023, Available: https://www.tmforum.org/resources/specifications/tmf632-party-management-api-rest-specification-v5-0-0/

[87] TMForum, "TMF669 Party Role Management API v5.0.0", 2023, Available: https://www.tmforum.org/resources/specifications/tmf669-party-role-management-api-user-guide-v5-0-0/

[88] TMForum, "TMF634 Resource Catalog Management API v4.1.0", 2021, Available: https://www.tmforum.org/resources/specification/tmf634-resource-catalog-management-api-user-guide-v4-1-0/

[89] TMForum, "TMF652 Resource Ordering Management API v4.0.0", 2020, Available: https://www.tmforum.org/resources/specification/tmf652-resource-order-management-api-user-guide-v4-0-0/

[90] TMForum, "TMF639 Resource Inventory Management API v4.0.0", 2020, Available: https://www.tmforum.org/resources/specification/tmf639-resource-inventory-api-user-guide-v4-0/

[91] Cloud Native Computing Foundation, Kubernetes, "Production-Grade Container Orchestration". [Online.] Available: https://kubernetes.io/

[92] OpenInfra Foundation, "OpenStack". [Online.] Available: https://www.openstack.org/

[93] jFrog Container Registry. [Online.] Available: https://jfrog.com/container-registry/

[94] GitLab Container Registry. [Online.] Available: https://docs.gitlab.com/ee/user/packages/container_registry/

[95] GitHub Container Registry. [Online.] Available: https://github.com/features/packages

[96] Harbor. [Online.] Available: https://goharbor.io/

[97] Materna-Workload-Traces. [Online.] Available: https://www.kaggle.com/datasets/kpiyush04/maternaworkloadtraces

[98] "Smart Contracts and Chaincode — Hyperledger Fabric Docs main documentation". [Online.] Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/smartcontract/smartcontract.html#smart-contract

[99] ETSI, "PDL Services for Decentralized Identity and Trust Management," May 2023. [Online.] Available: https://www.etsi.org/deliver/etsi_gr/PDL/001_099/019/01.01.01_60/gr_PDL019v010101p.pdf

[100] J. A. Hurtado Sánchez, K. Casilimas, and O. M. Caicedo Rendon, "Deep Reinforcement Learning for Resource Management on Network Slicing: A Survey," Sensors, vol. 22, no. 8. MDPI AG, p. 3031, Apr. 15, 2022. doi: 10.3390/s22083031.

[101] E. J. Salazar, M. Jurado, and M. E. Samper, “Reinforcement Learning-Based Pricing and Incentive Strategy for Demand Response in Smart Grids,” Energies, vol. 16, no. 3. MDPI AG, p. 1466, Feb. 02, 2023. doi: 10.3390/en16031466.

[102] F. Z. Dekhandji and A. Recioui, “An Investigation into Pricing Policies in Smart Grids,” The 1st International Conference on Computational Engineering and Intelligent Systems, vol. 4. MDPI, p. 15, Feb. 10, 2022. doi: 10.3390/engproc2022014015.

[103] N. C. Luong, P. Wang, D. Niyato, Y.-C. Liang, Z. Han, and F. Hou, “Applications of Economic and Pricing Models for Resource Management in 5G Wireless Networks: A Survey,” IEEE Communications Surveys &amp; Tutorials, vol. 21, no. 4. Institute of Electrical and Electronics Engineers (IEEE), pp. 3298–3339, 2019. doi: 10.1109/comst.2018.2870996.

[104] J. Lelli, S. Claudio, L. Abeni y D. Faggioli, “Deadline scheduling in the Linux kernel,” Software: Practice and Experience, vol. 46, Jun. 2016.

[105] G. Buttazzo, “Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications,” Third Edition, Springer.

[106] H. Ahlehagh and S. Dey, “Video-Aware Scheduling and Caching in the Radio Access Network,” IEEE/ACM Transactions on Networking, vol. 22, no. 5. Institute of Electrical and Electronics Engineers (IEEE), pp. 1444–1462, Oct. 2014. doi: 10.1109/tnet.2013.2294111.