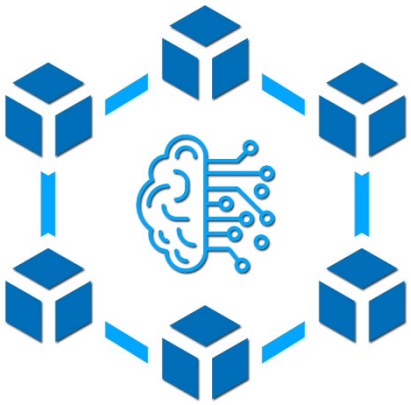

An Artificial Intelligent Aided Unified Network for Secure Beyond 5G Long Term Evolution [GA: 101096456]

Deliverable 3.2

NANCY Network Functionalities

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	NANCY Network Functionalities
Deliverable Number	D3.2
Work Package	WP3 NANCY Architecture & Orchestration
Associated Task	Task 3.2 Enabling common network functionalities
Dissemination Level	Public
Due Date	31 August 2024
Completion Date	29 August 2024
Submission Date	31 August 2024
Deliverable Lead Partner	SSS
Deliverable Author(s)	Alessandro Biondi (SSS), Daniel Casini(SSS), Luca Abeni (SSS), Mauro Marinoni (SSS), Carolina Fortuna (JSI), Blaz Bertalanic (JSI), Ljupcho Milosheski (JSI), Shih-Kai Chou (JSI), Andrej Cop (JSI), Gregor Cerar (JSI), Alvise Rigo (VOS), Giorgos Panayotidis (CERTH), Maria Belisioti (OTE), Ramón Sánchez Iborra (UMU), Rodrigo Asensio Garriga (UMU), Gonzalo Alarcón Hellín (UMU), Eleftherios Fountoukidis (SID), Konstantinos Kyranou (SID), Georgios Michoulis (SID)
Version	1.0

Document History

Version	Date	Change History	Author(s)	Organisation
0.1	10 April 2024	Initial version	Carolina Fortuna, Blaz Bertalanic, Daniel Casini, Alessandro Biondi	IJS, SSS
	24 April 2024	Adding to Section Spectrum Activity Monitoring	Ljupcho Milosheski	JSI
	25 April 2024	Adding section Localization Service, adding to Section Anomaly detection	Shih-Kai Chou, Blaz Bertalanic	JSI
	26 April 2024	Adding Section 4.3 Advanced Connectivity of Mobile Nodes	Gonzalo Alarcón Hellín, Ramón Sánchez Iborra, Rodrigo Asensio Garriga	UMU
0.2	29 April 2024	Adding to Section 3.4, Revision of Localization Service	Luca Abeni, Alessandro Biondi, Daniel Casini, Mauro Marinoni, Shih-Kai Chou	SSS, IJS
	30 April 2024	Revision of Section Spectrum Activity Monitoring and Anomaly detection	Ljupcho Milosheski, Blaz Bertalanic	JSI
0.3	20 May 2024	Merged contributions from CERTH	Giorgos Panayotidis, Daniel Casini, Mauro Marinoni	CERTH, SSS
0.4	28 May 2024	Merged contributions from OTE	Maria Belesioti , Daniel Casini, Mauro Marinoni	OTE, SSS

0.5	06 June 2024	Executive Summary	Daniel Casini	SSS
0.6	28 June 2024	Adding Integration with model repository	Blaz Bertalanic, Andrej Cop	IJS
0.7	30 July 2024	Extended Introduction, updated lists, and fixes.	Alessandro Biondi, Daniel Casini, Mauro Marinoni/ Carolina Fortuna, Gregor Cerar	SSS/ IJS
0.8	22 July 2024	Consistency check with general architecture	Blaz Bertalanic/ Eleftherios Fountoukidis, Konstantinos Kyranou, Georgios Michoulis	IJS/ SID
0.9	23 July 2024	Final version for internal review	Daniel Casini, Mauro Marinoni	SSS
0.95	31 July 2024	Integration of reviews	Mauro Marinoni	SSS
1.0	29 August 2024	Quality Revision	Anna Triantafyllou, Dimitrios Pliatsios	UOWM

Internal Review History

Name	Organisation	Date
Konstantinos Kaltakis	8BELLS	24 July 2024
Ramón Sánchez Iborra	UMU	25 July 2024
Gonzalo Alarcón Hellín	UMU	25 July 2024

Quality Manager Revision

Name	Organisation	Date
Anna Triantafyllou, Dimitrios Pliatsios	UOWM	29 August 2024

Table of Contents

Table of Contents 4
List of Figures 6
List of Tables 7
List of Acronyms 8
Executive summary 9
1. Introduction 10
1.1. Purpose of the document 10
1.2. Structure of the document 12
2. Services for O-RAN and MEC 13
2.1. Location Prediction Functionality 13
2.1.1. Monitoring framework and data collection 14
Lumos5G 14
IEEE CTW 2019 Competition Dataset 15
IEEE CTW 2020 Competition Dataset 15
LOG-a-TEC testbed 16
Spanish testbed at UMU premises 16
2.1.2. Model development methodology 17
2.1.3. Implementation 18
2.1.4. Results 20
2.2. Link Anomaly Detection Functionality 22
2.2.1. Monitoring framework and data collection 23
2.2.2. Model development methodology 24
2.2.3. Implementation 25
2.2.4. Results 26
2.3. Spectrum Activity Characterization 26
2.3.1. Monitoring framework and data collection 27
2.3.2. Model development methodology 28
2.3.3. Implementation 29
2.3.4. Results 30
2.4. Throughput Prediction 31
2.4.1. Monitoring framework and data collection 34
2.4.2. Model development methodology 35
a. LSTM 36
b. seq2seq 36

c. seq2seq 37
2.4.3. Implementation 37
2.4.4. Results 40
3. Technology Stack 43
3.1. Models in the Self Evolving Model Repo 43
3.2. RAN enabled protocol stack 44
3.3. MEC enabled protocol stack with ARM hardware architecture 46
3.3.1. Monitoring 47
3.4. Limiting the underutilization of virtualized resources 48
4. Support for the Usage Scenarios 52
4.1. Fronthaul 52
4.1.1. Direct connectivity 52
4.1.2. Coordinated multi-point connectivity 53
4.2. Advanced Coverage Expansion 53
4.3. Advanced Connectivity of Mobile Nodes 54
5. Conclusion 56
Bibliography 57

List of Figures

Figure 1 NANCY WP3 Technology Stack Overview..... 11
Figure 2. Localization in Different Services 14
Figure 3 :Distinct representation of the 4 anomaly types, that can occur in wireless signals..... 23
Figure 4: Proposed GNN model architecture, where GINE represents Graph Isomorphism Edge Network, GM represents Global Max Pooling, and BN represents Batch normalization. 24
Figure 5: Sample of 8 spectrogram segments from the data..... 28
Figure 6: Architecture of the self-supervised CNN system. 28
Figure 7: Distribution of samples from each cluster along the frequency band for the clusters obtained by the RN-based approach. 30
Figure 8: Original and filtered throughput of recording number 116 with MA(5)..... 35
Figure 9: Original and filtered throughput of recording number 116 with MA(10)..... 36
Figure 10: The predictions (orange) and the original data (blue) of throughput for 20 seconds (steps) 41
Figure 11: The performance (RMSE) at each time step for the three DL-models using the **B** and **S** feature groups..... 41
Figure 12 An example of a Docker image python template for Spectrum Activity Characterization MEC service..... 44
Figure 13 Template of a Helm chart to deploy the MEC service..... 45
Figure 14: VOSySmonitor overview..... 47
Figure 15: SCHED_DEADLINE scheduling architecture..... 49
Figure 16: CDF of the normalized response times when the threads execute in a container with one virtual CPU scheduled by the standard Linux scheduler or by SCHED_DEADLINE with (Q,P)=(7.5ms,11ms)..... 50
Figure 17: CDF of the normalized response times when the threads execute in VM with one virtual CPU scheduled by the standard Linux scheduler or by SCHED_DEADLINE with (Q,P)=(7.5ms,11ms).. 51
Figure 18: The major AR/VR use cases [47]..... 52

List of Tables

Table 1: Lumos5G Dataset Summary 14
Table 2: IEEE CTW 2019 Competition Dataset Summary 15
Table 3: IEEE CTW 2020 Competition Dataset Summary 15
Table 4: LOG-a-TEC Dataset Summary 16
Table 5: Feature Grouping 19
Table 6. RMSE Performance Summary 21
Table 7: Results of our proposed method compared to the state-of-the-art Hive-Cote2 and InceptionTime models 26
Table 8: Comparison of computational complexity 26
Table 9: Performance and models' size. 31
Table 10: The performance (RMSE) of the three DL models for four different experimental set-ups . 42

List of Acronyms

Acronym	Explanation
5GPPP	5G Public-Private Partnership
AI	Artificial Intelligence
ARM	Advanced RISC Machine
AR/VR	Augmented Reality/Virtual Reality
B-RAN	Brownfield Radio Access Network
BLE	Bluetooth Low Energy
BN	Batch Normalization
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CDF	Cumulative Distribution Function
CSI	Channel State Information
CU	Central Unit
C-ITS	Cooperative-Intelligent Transportation Systems
DL	Deep Learning
DSS	Decision Support Systems
DU	Distributed Unit
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GBDT	Gradient Boosting Decision Tree
GINE	Graph Isomorphism Network Edge
GM	Global Max Pooling
GNN	Graph Neural Network
GPS	Global Positioning System
IoT	Internet of Things
KNN	K-Nearest Neighbors
LSTM	Long Short-Term Memory
LTE	Long Term Evolution
MA	Moving Average
MEC	Multi-access Edge Computing
ML	Machine Learning
MRAT	Multi Radio Access Technology
NR	New Radio
OBU	On-Board Units
OFDM	Orthogonal Frequency Division Multiplexing
O-RAN	Open Radio Access Network
PHY	Physical Layer
QoS	Quality of Service
RAN	Radio Access Network
RAT	Radio Access Technologies
RF	Radio Frequency
RMSE	Root Mean Square Error
RSSI	Received Signal Strength Indicator
RSSNR	Received Signal Signal-to-Noise Ratio
SNR	Signal-to-Noise Ratio
SSL	Self-Supervised Learning
VGG	Visual Geometry Group
VM	Virtual Machine
VNF	Virtual Network Function

Executive summary

This document presents the design solutions for the common network functionalities of NANCY. These functionalities leverage the Multi-access Edge (MEC) protocol stack, which includes slice control and resource management.

First, the deliverable lists and describes in detail the considered MEC services, ranging from Location-based services (Section 2.1), Link Anomaly and Quality Detection (Section 2.2), Spectrum Activity Characterization (Section 2.3), and Throughput Prediction (Section 2.4). The services are described in detail and presented with implementation details (including code snippets) and experimental results. These functionalities will be provided to Work Package 6: NANCY System Integration, Validation, and Demonstration to be integrated into the NANCY platform.

Then, the deliverable presents the technology stacks needed to support the MEC services, including the self-evolving model repository (Section 3.1), the RAN-enabled and MEC-enabled protocol stacks (Section 3.2 and Section 3.3), and discusses how virtualized resources can be used efficiently by limiting their underutilization. The deliverable then presents how the self-evolving model repository integrates with the AI virtualizer (Section 4).

Finally, an overview of how these design solutions support the usage scenarios of NANCY is presented (Section 5).

1. Introduction

1.1. Purpose of the document

The NANCY architecture is based on three pillars: (a) the incorporation of Open-Radio Access Network (O-RAN) and prior 5GPPP project architectural design findings; (b) the development of NANCY enabling innovations and their integration into the architecture; and (c) Multi-access Edge Computing (MEC). WP3 as a whole pursues six objectives as follows:

(a) identify the architectural gaps in current research projects and state-of-art solutions;

(b) identify the key outcomes and the architecture commonalities from the O-RAN solution;

(c) specify the required architectural components to support B-RAN;

(d) define in detail the overall NANCY reference architecture, including the software framework, tools, schemes, and algorithms by taking into account the identified requirement in WP1, along with the current SOTA technology axes, models and O-RAN open architecture requirements;

(e) design novel Artificial Intelligence (AI)-based algorithms, functionalities and solutions following the experimental-driven modelling and optimization approach;

(f) identify and specify orchestration functions that will be used to manage the overall orchestration framework and support the dynamic nature of NANCY.

While objectives (a)-(d) have been addressed in D3.1 and objective (f) will be addressed in D3.3 and D3.4, the current D3.2 focuses mostly on objective (e). T3.2 titled Enabling Common network functionalities carries out work reported in this deliverable and focuses on developing AI-based functionalities that can be RAN on the MEC and/or O-RAN. Multi-access Edge Computing (MEC) and O-RAN are both significant concepts in the evolving landscape of 5G and beyond, serving different purposes but complementing each other. MEC aims to bring computation and storage resources closer to the end-users by deploying them at the network edge. This reduces latency, enhances performance, and supports real-time applications like Internet-of-Things (IoT), augmented/virtual reality (AR/VR), and autonomous driving. O-RAN aims to create a more open and interoperable RAN architecture using open standards and interfaces. This allows for more vendor diversity, flexibility, and innovation in the deployment and operation of RAN components. O-RAN also enables modular service creation, concatenation and orchestration as opposed to monolithic approaches that are frequent [1].

The network functionalities developed in this task are based on Machine Learning (ML) mechanisms, and are able to run on enterprise-level servers running both on the cloud and in the edge using Intel x86 hardware platforms as well as on ARM architectures that are particularly suitable for the edge and far edge as illustrated in Figure 1. These architectures can also be used with O-RAN, especially when running DU and CU functionalities. Their deployment and run time are managed by orchestrators leveraging layers of virtualization such as a light-weight container, compartments developed in T3.4 for ARM (ARMv8-A details in D3.4) and virtual machines and corresponding orchestration functionalities such as the Slice Manager as detailed later in this report as well as in D3.3 and D3.4 as depicted in Figure 1. Furthermore, time-critical functionalities can be prioritized and managed by the extensions of the existing Linux scheduler (SCHED_DEADLINE) running on the x86 and ARM hardware platforms. Furthermore, limiting the underutilization of virtualized resources by integrating advanced scheduling technologies for Linux that provide consistent benefits, in terms of resource isolation against standard scheduling solutions is also enabled.

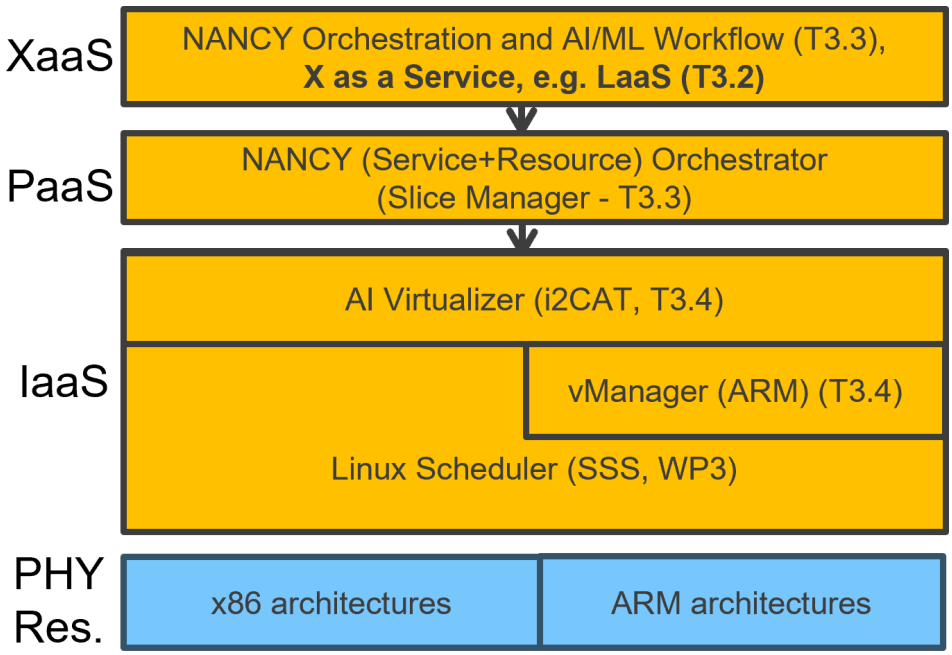

Figure 1 NANCY WP3 Technology Stack Overview.

All the network functionalities developed in this task can be instantiated as platform services on the ORAN RT and non-RT RIC to be used in service chains realizing complex network functionality. For instance, localization as a platform service may predict future locations to inform beamforming, proactive handover, caching, offloading, etc. At the same time, some of them can also be instantiated as user services in O-RAN or MEC to serve user space applications such as robot guidance in closed spaces without GPS coverage. Spectrum monitoring, link layer anomaly monitoring and throughput predictions also serve as platform services able to contribute to improving multiple performance criteria such as latency and quality of service and inform caching, offloading or other strategies. Furthermore, in addition to the four network functionalities developed and described in this report, namely location, link layer anomaly detection, spectrum characterization and throughput prediction that is also experimentally verified and instantiated through the Slice Manager, we also provide blueprints for experimentally evaluating models developed in other WPs.

As part of Task 3.2, NANCY realized a MEC-enabled system stack that encompasses MEC-based services alongside resource management at different scales, ranging from software workloads to network slices. Altogether, these contributions constitute the common network functionalities of the NANCY architecture, integrating results related to pillars (a), (b), and (c). These functionalities are expected to be used in all supported usage scenarios of NANCY.

The NANCY architecture was designed to integrate novel MEC services that improve multiple performance criteria such as latency and quality of service. Caching mechanisms based on the location of users and Artificial Intelligence (AI) / Machine Learning (ML) algorithms, when combined together, proved particularly effective in achieving beyond-state-of-the-art performance and enabling a new class of services, as well as enhancing connectivity and service delivery across many devices. Similar findings were concluded in monitoring the link status that, for Beyond 5G networks, requires facing unprecedented challenges to deal with the integrity and reliability of large-scale, high-throughput, and low-latency wireless communications.

As part of common network functionalities, NANCY established methods to also provide throughput prediction, given that the bandwidth metric has served as the dominant theoretical means to quantify the maximum achievable amount of information that can be transmitted and can hence be used as an enabler of information to build other services.

Finally, we also report on how the developed common network functionalities can be leveraged to support the usage scenarios of NANCY.

This deliverable reports on the results of Task 3.2 — Enabling common network functionalities and relates to project result R8 — A Novel AI Virtualizer for Underutilized Computational and Communication Resource Exploitation.

1.2. Structure of the document

The deliverable is structured as follows:

- **Section 1 - Introduction** provides the introduction of this deliverable.
- **Section 2 - Services for O-RAN and MEC** elaborates on the network functionalities based on AI techniques, namely location prediction, link layer anomaly monitoring, spectrum monitoring and throughput prediction developed to support.
- **Section 3 - Technology Stack** provides an overview of the technology stack.
- **Section 4 - Support for the Usage Scenarios** outlines how the services and technologies are leveraged to support the usage scenarios.
- **Section 5 - Conclusion** concludes this deliverable.

2. Services for O-RAN and MEC

Recently, training and deploying reliable and effective AI solutions in O-RAN has received increasing interest from academia and industry alike for applications ranging from controlling RAN resource and transmission policies, forecasting and classifying traffic and Key Performance Indicators (KPIs), placement of network components and functions, etc. [1]. In this section, we elaborate on four network functionalities based on AI techniques, namely location prediction, link layer anomaly monitoring, spectrum monitoring and throughput prediction developed to support. We develop these functionalities by employing beyond SotA ML algorithms on available real-world datasets developing suitable predictions and classification models. The models are then prepared as containerized services and ready for experimental in-lab and in-testbed validation and evaluation. In the following sections, we also elaborate on their instantiation through the Slice Manager, instantiation and execution on ARM architectures with decreased resource underutilization and provide usage scenario-related considerations.

2.1. Location Prediction Functionality

Location-based services (LBS) are software services that take into account the geographic location and even context of an entity in order to adjust the content, information or functionality delivered. Entities can be people, animals, plants, assets and any other object. Perhaps the most widely used LBS is the Global Positioning System (GPS), which integrates data from satellite navigation systems and cell towers and is used daily in navigation systems. Another popular application of LBS is locating tagged items and assets in indoor environments [2].

With 5G and beyond systems, accurate localization is no longer only important for providing more relevant information to the end user, but also for optimal operation and management of the network, e.g. for creating and steering the beams of antenna array-based radio heads [3]. Machine learning (ML) based localization and user tracking are envisioned to support future networks. Given the ubiquitous presence of wireless networks and the envisioned availability of radio-frequency (RF) measurements in emerging O-RAN architectures, ML methods promise the highest accuracy, albeit at a higher deployment cost. In particular, in the offline training phase, ML methods use available RF measurements and learn how to estimate locations [4].

In NANCY we aim to automate model development for localization on a wide range of data using a wide selection of ML techniques to achieve beyond SotA performance in a replicable way. The resulting models, wrapped as location functionality aim to support all three usage scenarios as depicted in Figure 2. They can inform decision strategies for several applications, such as caching and handover, offloading and coverage expansion.

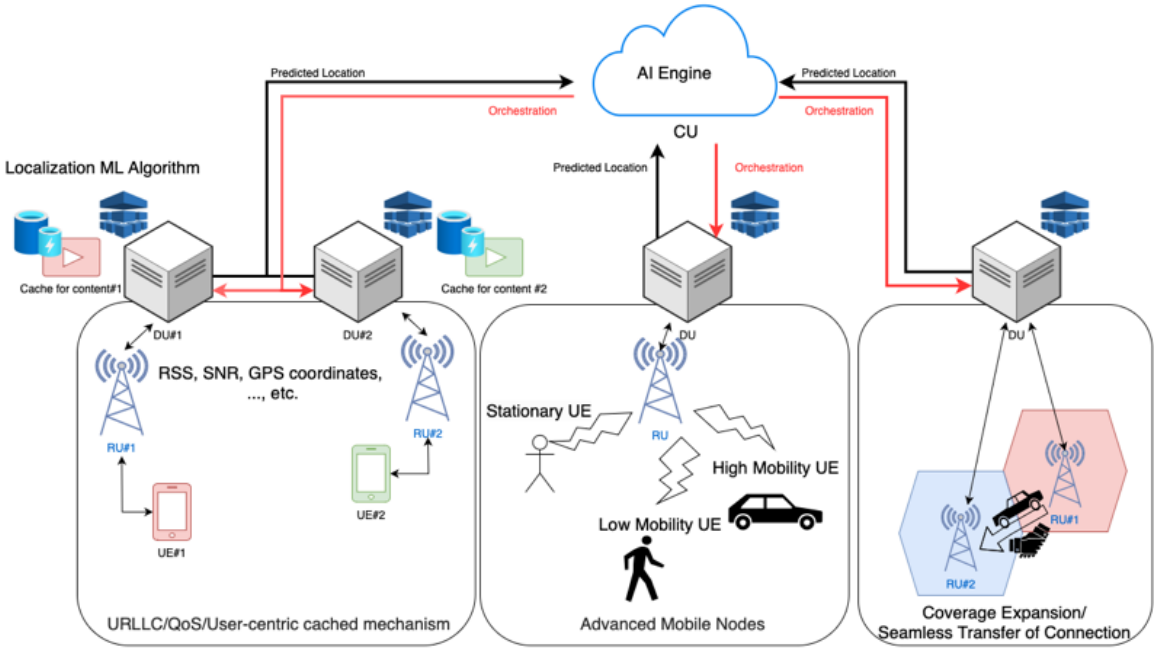

Figure 2. Localization in Different Services

2.1.1.Monitoring framework and data collection

To develop the localization functionality, we leverage 4 existing open source datasets and a NANCY dataset collected in the UMU Spanish test facility. These datasets enable the model and functionality development that can be later integrated into the overall NANCY platform in WP6. In an integrated set-up that embraces O-RAN principles, the collection of statistics is possible via a publish-subscribe model through open and standardized interfaces such as O1 and E2 [1].

A summary of each of the considered datasets for model development is provided below.

Lumos5G

Lumos5G dataset represents a 1300-meter-long "Loop" area. The measuring is done with the commercial cellular network, where the parameters are collected through a self-developed Android application and the experiment is taking place in downtown Minneapolis, which covers various environments, including roads, railroad crossings, and recreational outdoor parks [5]. The dataset presents several useful parameters for training purposes in 5G/4G LTE environments, with the setup and data summarized in Table 1.

Table 1: Lumos5G Dataset Summary

Property	Description
Number of samples	68,118 samples
Location Information	[Latitude, Longitude]/ sample
Mobility	Moving Speed and Mobility Mode
Direction	Compass Direction and Trajectory Direction
Physical Layer (PHY) Measurements	Radio Status, RSSI/RSRP/RSRQ/RSSNR
Throughput	[0:1920] Mbps
Tower ID	[1:24]

In Table 1, "Location Information" presents the GPS coordinates of the device. "Mobility" provides the speed of the measuring equipment and whether the measurement is done while walking or mounting to a car, and in "Direction," the azimuth angle is listed, whereas the physical layer (PHY) measurements consist of several physical layer characteristics, such as received signal strength indicator (RSSI) and received signal signal-to-noise ratio (RSSNR), the measurements of LTE and NR are captured at the same time and divided into two columns. The throughput value of the corresponding measurement and the connected base station ID are shown in "Throughput" and "Tower ID," respectively. Overall, 68k measurements are available.

IEEE CTW 2019 Competition Dataset

The IEEE CTW 2019 Competition dataset [6] is utilized for a positioning algorithm competition and is composed of CSI and signal-to-noise ratio (SNR). The associated location is in a three-dimensional form, and the measurement is acquired by an indoor massive setup with an 8 × 2 antenna array and a moving transmitter. The transmitter moves randomly in a 4 × 2m area while transmitting orthogonal frequency-division multiplexing (OFDM) pilots. The dataset is summarized in Table 2.

Table 2: IEEE CTW 2019 Competition Dataset Summary

Property	Description
Number of samples	17,486 samples
Number of subcarriers	924
Location Information	[x,y,z]
Data shape of CSI	$16 \times 924 \times 2$ (Real and imaginary parts)
Data shape of SNR	$16 \times$ Number of samples
Central Frequency	1.25 GHz
Bandwidth	20 MHz

IEEE CTW 2020 Competition Dataset

The IEEE CTW 2020 Competition dataset [7] is a suburban 1.25GHz localization dataset provided by the University of Stuttgart. The total measurement time was 8 hours, where the compact hand-wagon was manually pushed around in a residential area of size about 600 m × 800 m. The transmitter is based on an Ettus USRP B210 [8]. Its Field Programmable Gate Array (FPGA) was programmed to generate orthogonal frequency-division multiplexing (OFDM) symbols of B = 18MHz effective bandwidth. The receiver used a 64-element antenna array (with only 56 logging valid data) with dual-polarized patch antennas, while only the vertical polarization was used. The dataset is summarized in Table 3.

Table 3: IEEE CTW 2020 Competition Dataset Summary

Property	Description
Number of unlabelled samples	180.960 samples
Number of labelled samples	24.895 samples
Number of subcarriers	924
Location Information	[x,y,z]
Data shape of CSI	$56 \times 924 \times 2$ (Real and imaginary parts)
Data shape of SNR	$56 \times$ Number of samples
Central Frequency	1.25 GHz
Bandwidth	18 MHz

LOG-a-TEC testbed

Unlike previous datasets, the LOG-a-TEC dataset is used for the fingerprinting-based localization method [9]. The main idea of this methodology involves the compilation of signal features from every conceivable location within the designated area of interest, forming a comprehensive fingerprint database. Subsequently, the localization process entails the comparison of measured fingerprints at an unidentified location with those contained within the database, culminating in identifying the location corresponding to the most closely aligned fingerprint. This dataset is collected in the LOG-a-TEC testbed within the campus of Jozef Stefan Institute. More specifically, the measuring setup consists of a portable Bluetooth low energy (BLE) transceiver that emits BLE beacons with intervals of 100 milliseconds at a power level of -2 dBm, and a total of 25 BLE nodes capable of capturing the BLE signals from the portable transceiver. The portable transceiver stops at a designated point to let BLE nodes collect signals for one minute and then move to another location. Furthermore, the experimental area is 150 square meters rectangular and divided into a grid of 5 × 26 points, where each point is separated by 1.2 meters. A summary of the parameters used in the LOG-a-TEC dataset can be seen in Table 4.

Table 4: LOG-a-TEC Dataset Summary

Property	Description
Number of samples	723,147 samples
Location Information	[x,y]
Number of points	5 × 26 Points
Number of BLE nodes	25
RSS	[-128 : -54] dBm
Experimental Area	150 square meters
Beacon Broadcasting	BLE beacons at –2 dBm power. 100 ms intervals

Spanish testbed at UMU premises

A NANCY radio metric dataset has been collected at the UMU campus using a vehicle with multiple RATs with two network providers (Nokia and AW2S) and two types of generated traffic (TCP or UDP). As data collection campaigns tend to be time-consuming and intensive, a pre-collection campaign has been finalized logging 90 different metrics with slightly more than 2130 time stamps/samples for each network and each traffic type. For the NANCY functionalities in general as well as the location functionality in particular, the columns from the snippet below seem the most relevant, however, ongoing research is still determining the feature importance. A second, more extensive data collection campaign is also planned.

Column7-8: mcc mnc of the current network
Cloumn 14: rsrp from modem perspective
Column 15: snr from modem perspective
Column 39: plmn of home network (SIM configuration)
Column 42: global_cell_id
Column 43: physical_cell_id
Column 45: rsrp from cell perspective (this column seems to have one mistaken 0 at the end)
Column 46: rsrq from cell perspective
Column 47: snr from cell perspective
Column 48: Tracking area code
Column 57-64: Information related packet/bytes transmitted and received: ok, error. overflows and dropped
Column 72-73: Timestamps in different formats
Column 73-86: GPS info
Column87-88: Accumulative upload and download kbs
Column 89-90: Upload and Download kbps

2.1.2.Model development methodology

In order to be able to develop models across various datasets including various technologies and different measured parameters, we developed a reproducible and configurable pipeline that automates the model development process. The methodology automated by the pipeline begins with the data stage, where raw datasets are acquired and subsequently moved through a sequence of processing phases: preparation, feature engineering, training, evaluation, and results compilation. At the core of this workflow are two essential components: configurations and artifacts. The configurations contain crucial information such as pipeline instructions and ML model parameters. These configuration files are essential for orchestrating the pipeline by providing detailed instructions on how each stage should process the data, ensuring consistency and reproducibility throughout the workflow. The configurations interact with the preparation, feature engineering, training, and evaluation stages by providing the necessary settings and parameters as can be seen in the listing for the UMU dataset config dvc.yaml file below. This ensures that each stage follows a predetermined setup.

```
vars:
 - path:
    # Path are relative to location of dvc.yaml file
    scripts: ../../src/umu
    common: ../../src
    data: ../../artifacts/umu/data
    models: ../../artifacts/umu/models
    reports: ../../artifacts/umu/reports

 - splits: [Random, KFold, LeaveOneGroupOut]

 # Models are defined in params.yaml


stages:
 prepare:
  desc: Prepare UMU dataset for the MLOps pipeline
  cmd: |
   unzip ${path.data}/raw/UMU.zip -d ${path.data}/raw/
   python ${path.scripts}/prepare.py --method average --input ${path.data}/raw/tcp_nokia_20240325.xlsx --output
${path.data}/interim/umu.pkl
   rm ${path.data}/raw/*.xlsx
   rm ${path.data}/raw/__MACOSX/.*.xlsx
  deps:
   - ${path.scripts}/prepare.py
   - ${path.data}/raw/umu.zip
  outs:
   - ${path.data}/interim/umu.pkl


 featurize:
  desc: Enrich dataset with additional features
  cmd: >
   python ${path.scripts}/featurize.py
   --input ${path.data}/interim/umu.pkl
   --output ${path.data}/prepared/umu.pkl
  deps:
   - ${path.scripts}/featurize.py
   - ${path.data}/interim/umu.pkl
  outs:
   - ${path.data}/prepared/umu.pkl

 split:
  desc: Split dataset and store indices
  matrix:
   split: ${splits}
  cmd: >
   python ${path.scripts}/split.py
   --input ${path.data}/prepared/umu.pkl
```

```
    --split ${item.split}
    --output-indices ${path.data}/splits/${item.split}Split.pkl
  params:
    - split
  deps:
    - ${path.scripts}/split.py
    - ${path.data}/prepared/umu.pkl
  outs:
    - ${path.data}/splits/${item.split}Split.pkl


optimization:
  desc: Determine best hyper-parameters for algorithm on several CVs
  matrix:
    split: ${splits}
    model: ${models}
  cmd: >
    python ${path.common}/benchmark.py
    --use ${item.model}
    --data ${path.data}/prepared/umu.pkl
    --split-indices ${path.data}/splits/${item.split}Split.pkl
    --output-results ${path.models}/${item.model}-${item.split}Split-results.pkl
  params:
    - models.${item.model}
  deps:
    - ${path.common}/benchmark.py
    - ${path.data}/prepared/umu.pkl
    - ${path.data}/splits/${item.split}Split.pkl
  outs:
    - ${path.models}/${item.model}-${item.split}Split-results.pkl


evaluation:
  cmd: >
    python ${path.common}/report.py
    ${path.models}/*-results.pkl
    --output-report ${path.reports}/metrics.json
  deps:
    - ${path.common}/report.py
    - ${path.models}
  metrics:
    - ${path.reports}/metrics.json:
        cache: false
```

As the models are trained, various outputs and intermediate data are generated and stored in the artifacts directory. This directory is structured to hold datasets at different stages of preparation, including raw, interim, splits, and prepared data, as well as the results of feature engineering, training, and evaluation processes. The artifacts ensure that each step in the pipeline is documented and reproducible, enabling easy access for subsequent runs or validations.

2.1.3.Implementation

To dive in into some of the implementation details of various steps of the methodology elaborated in the previous sub-section, we focus on feature engineering and model development aspects by zooming into the Lumos 5G dataset. Recalling the summary of the dataset in Table 1, we group different features to identify which group is superior for predicting the location of the user. For this particular dataset, we consider three groups of features as listed in Table 5. P represents the measurement from the physical layer and throughput. M represents the mobility.

Table 5: Feature Grouping

Dataset	Feature Group	List of Features
Lumos5G	P	PHY measurement
	P+M	PHY measurements + Mobility
	P+M+Tower ID	PHY measurements + Mobility + Tower ID

In the following listing, we first present code snippets for feature engineering followed by training with two classical ML algorithms and finishing with the evaluation approach.

First, we import the data and divide it into different feature groups. Since some tower IDs are missing in the original Lumos5G dataset, the importing of data in feature group P+M+Tower ID is done in the following way,

```
# Iterate through each tower_id
for tower_id in range(1, 25):
    # Filter data for the current tower_id
    tower_data = df[df['tower_id'] == tower_id]
    # Select features and target variables
    features = ['lte_rssi','lte_rsrp',
'lte_rsrq','nr_ssRsrp','nr_ssRsrq','nr_ssSinr','Throughput','longitude','latitude','movingSpeed','compassDirection']
    X = tower_data[features]
    y_latitude = tower_data['latitude']
    y_longitude = tower_data['longitude']
    # Check if the dataset size is sufficient for splitting
    if len(tower_data) < 2:
      print(f"Skipping Tower ID {tower_id} due to insufficient data.")
      continue
```

Across all the experiments, we divide the dataset into training and testing data in a 70/30 ratio and utilize MinMaxScaler. The training and prediction are done with the below parameters:

GBDT:

```
# Initialize and train the model for latitude
    model_latitude = GradientBoostingRegressor(n_estimators=2000, max_depth=9, min_samples_split=5, learning_rate=0.01,
loss='huber')
    model_latitude.fit(X_train_scaled, y_latitude_train)

    # Make predictions on the test set for latitude
    predictions_latitude = model_latitude.predict(X_test_scaled)

    # Initialize and train the model for longitude
    model_longitude = GradientBoostingRegressor(n_estimators=2000, max_depth=9, min_samples_split=5, learning_rate=0.01,
loss='huber')
    model_longitude.fit(X_train_scaled, y_longitude_train)

    # Make predictions on the test set for longitude
    predictions_longitude = model_longitude.predict(X_test_scaled)

```

k-NN:

```
model_longitude = KNeighborsRegressor(n_neighbors=1)
model_longitude.fit(X_train_scaled, y_lon_train)

# Predictions for longitude
predictions_longitude = model_longitude.predict(X_test_scaled)

# Initialize and train the model for latitude
model_latitude = KNeighborsRegressor(n_neighbors=1)
```

```
model_latitude.fit(X_train_scaled, y_lat_train)

# Predictions for latitude
predictions_latitude = model_latitude.predict(X_test_scaled)

```

RMSE results are calculated as follows:

```
# Function to calculate geodesic distance between two points (in meters)
def haversine(lon1, lat1, lon2, lat2):
    # Convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # Difference in longitudinal and latitudinal coordinates
    dlon = lon2 - lon1
    dlat = lat2 - lat1

    # Haversine formula to calculate geodesic distance
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = 6371e3 * c  # Earth radius * central angle between points
    return distance
# Calculate RMSE using geodesic distance
rmse_distance = np.sqrt(np.mean([(haversine(lon_true, lat_true, lon_pred, lat_pred))**2
                        for (lon_true, lat_true), (lon_pred, lat_pred) in zip(zip(y_lon_test, y_lat_test), zip(predictions_longitude,
predictions_latitude))]))
print('Root Mean Squared Error (RMSE) distance in meters:', rmse_distance)
```

2.1.4.Results

As localization is a well-researched topic where several models have been proposed and evaluated on dedicated datasets, we focused the efforts so far on automating the model development methodology to be able to develop a model that works well on several datasets. For that, we also integrated classical ML techniques such as linear regression, random forest regressor, boosted trees regressor, and k-nearest neighbour regressor, as well as automated model search using AutoML. Finally, several SotA deep learning architectures such as [2], [6], [10], and [11]. Additionally, beyond SotA symbolic regression approaches are also considered. Intermediate results are provided as follows.

Table 6 shows the corresponding RMSE results for feature groups considered in three different datasets. These results are presented in meters. We can observe that these two regression-based models work well when the designated area is relatively small. For example, the IEEE CTW 2019 dataset is collected in an 8 square meter area and the error of the models is in the range of 0.5-1 meters. While the experimental area of LOG-a-TEC is somewhat larger than CTW 2019, it covers 150 square meters and the error range is also below 1 meter. On the other hand, the Lumos5G dataset covers a loop of 1300-meter and the error ranges between 47 – 154 meters. Furthermore, from the results of the CTW 2019 dataset, we can find that CSI information serves as better input than SNR. Both inputs can help us to interpret the relationship between the transmitting and receiving nodes, where SNR describes the ratio of the power of the signal to the power of the noise. On the other hand, CSI includes factors that can characterize the wireless channels and offers more detailed system information. This showcases the importance of utilizing inputs that have richer representations of the communication environment for predicting the location of the user.

Table 6. RMSE Performance Summary

Approach	Lumos5G			IEEE CTW 2019			LOG-a-TEC		
	P	P+M	P+M+ Tower ID	S	C	S +C	R	R+ Ravg	R +Ravg+R std
GBDT	153.41	112.29	47.29	0.444	0.381	0.31	0.663	0.089	0.089
k-NN	126.6	107.4	66.02	1.06	0.86	0.745	0.794	0.056	0.042

The evaluation pipeline outputs results for several techniques, with several evaluation strategies and several metrics as can be seen in the listing below. For the initial Spanish testbed model development we selected the classical regressors with KFold, leave one group out and random evaluation strategies. The considered metrics are mean average error, mean Euclidean distance error, r squared and root mean square error. The final model and results will be reported in a scientific publication while the code (https://github.com/cfortuna/nancy-saas-localization) will be opened upon acceptance. In the meantime, it will be made available to the project partners as needed.


```
{
 "KNeighborsRegressor": {
  "KFold": {
   "mae": 4.748935349514614,
   "mede": 7.272458411943651,
   "r_squared": 0.9971716717576624,
   "rmse": 13.524191986747852
  },
  "LeaveOneGroupOut": {
   "mae": 3.298312062622098,
   "mede": 5.103712133921238,
   "r_squared": 0.9984551430508362,
   "rmse": 9.881186676047221
  },
  "Random": {
   "mae": 4.470374008340888,
   "mede": 6.848781135077714,
   "r_squared": 0.9973400923675911,
   "rmse": 13.098180840517188
  }
 },
 "LinearRegression": {
  "KFold": {
   "mae": 132.64124699355642,
   "mede": 207.34768294046182,
   "r_squared": 0.5313749135395425,
   "rmse": 181.74586510922097
  },
  "LeaveOneGroupOut": {
   "mae": 132.65344391303492,
   "mede": 207.40959931750248,
   "r_squared": 0.5325657771556465,
   "rmse": 181.6471832836293
  },
  "Random": {
   "mae": 133.4218774311487,
   "mede": 209.23245496934942,
   "r_squared": 0.5160468071246354,
   "rmse": 184.85861858881418
  }
 },
 "RandomForestRegressor": {
  "KFold": {
   "mae": 14.646271244109066,
```

```
    "mede": 22.46593249403624,
    "r_squared": 0.9780719193435119,
    "rmse": 38.28421061880832
   },
   "LeaveOneGroupOut": {
    "mae": 11.843505667012026,
    "mede": 18.147295470840607,
    "r_squared": 0.9816001835261788,
    "rmse": 35.099845046390556
   },
   "Random": {
    "mae": 15.130418016942631,
    "mede": 23.183869984954764,
    "r_squared": 0.9737094623847792,
    "rmse": 42.52032153685479
   }
  },
  "XGBRegressor": {
   "KFold": {
    "mae": 9.335501533184793,
    "mede": 15.348982945120564,
    "r_squared": 0.991094500859049,
    "rmse": 22.141841262621693
   },
   "LeaveOneGroupOut": {
    "mae": 7.736403025052239,
    "mede": 12.732924047699322,
    "r_squared": 0.9941623104083032,
    "rmse": 17.478201162027798
   },
   "Random": {
    "mae": 9.48664139725085,
    "mede": 15.620050247412705,
    "r_squared": 0.9900610355444059,
    "rmse": 23.169136513923085
   }
  }
}
```


2.2. Link Anomaly Detection Functionality

The development of next-generation Radio Access Networks (RANs) promises to revolutionize the telecommunication landscape, enhancing connectivity and service delivery across many devices. At the center of this transformation lies the important role of wireless signals, the robustness of which is important for the seamless operation of both legacy and emerging communication technologies.

With the complexity and density of devices within RANs intensifying, particularly in the light of Beyond 5G networks, the integrity and reliability of wireless communication face unprecedented challenges. Anomalies in wireless signals, ranging from unexpected disruptions to security breaches, pose significant risks to the stability and efficiency of these networks. Consequently, the detection and timely resolution of such anomalies are not merely beneficial but essential for maintaining service continuity and user trust. By continuously monitoring key network performance indicators, such as Channel Quality Indicator, Signal-to-Interference-plus-Noise Ratio, and Received Signal Strength Indicator (RSSI), network anomaly detection systems contribute to the prediction and prevention of issues that could otherwise lead to significant network disruptions.

In some of the previous scientific works a set of wireless anomalies in RSSI signal, that occur over a longer period of time when operating a network, were observed, namely Sudden Degradation

(SuddenD), Sudden Degradation with Recovery (SuddenR), Instantaneous Degradation (InstaD), and Slow Degradation (SlowD):

- **SuddenD** anomaly appears as a drop in RSSI causing that never recovers. From the user perspective, SuddenD anomaly may result in services becoming unavailable or offline. Network-wise, it could be due to the transmitter ceasing electromagnetic field generation or the receiver failing to capture data.
- **SuddenR** displays itself as a drop in RSSI maintaining this state for a period, then returning to its original state, causing temporary communication disruption. From a user's perspective, services may temporarily slow down or become unavailable before resuming normal function.
- **InstaD** is seen as spikes in the received RSSI, where a short temporary disruption are occuring. From the user's perspective, a real-time service might suddenly become laggy, whereas non-real-time services might continue to function without any issues.
- **SlowD** appears as a slowly degrading signal through time. From the user perspective, the anomaly could go unnoticed because of no immediate effect on the application but would start noticing degraded experience after a certain period.

An example of the presented anomalies is shown in .

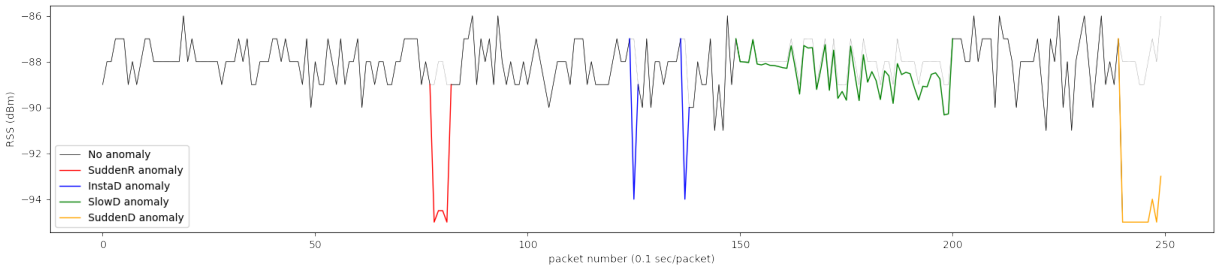

Figure 3 :Distinct representation of the 4 anomaly types, that can occur in wireless signals

The following solution includes also an original scientific contribution, that was published in the IEEE Open Journal of Communication Society [12].

2.2.1.Monitoring framework and data collection

To identify the presented anomalies, we selected the Rutgersers WiFi dataset, comprising 29 devices and 2,123 unique real-world time series RSSI measurements, each spanning 300 time steps. Although the selected dataset is based on WiFi signal, the described anomaly shapes are common across different technologies, thus the concept is valid also for B5G networks. This dataset enables the model and functionality development that can be later integrated into the overall NANCY platform in WP6. In an integrated set-up that embraces O-RAN principles, the collection of metrics and statistics is possible via a publish-subscribe model through open and standardized interfaces such as O1 and E2 [1].

We augmented this dataset by injecting anomalies as defined by the parameters described in [13]. Specifically, 33% of the time series traces were randomly selected to receive one of the four types of anomalies, while the remaining traces were left unaltered. This procedure was executed separately for each anomaly type, resulting in four distinct raw time series datasets, one for each anomaly. These datasets collectively form our comprehensive final dataset, where each anomaly type is represented by 700 samples, culminating in a total of 8,492 samples.

2.2.2.Model development methodology

To develop an AI-based model to correctly identify and classify the presented anomalies, the computational complexity of the models must be considered. It is imperative for the algorithm to detect anomalies in a near real-time fashion, to enable quick prevention or mitigation of the detected problems, but at the same time must also achieve high recognition performance.

Recently, Graph Neural Networks (GNN) showed a very good ratio between computational complexity and performance. Based on that, we decided to transform RSSI time series data into graphs and classify the anomalies with GNNs. GNNs have the ability to adeptly capture the intricate relationships present in the time series data, leading to enhanced performance in class prediction.

To transform time series data into graphs, we utilized the Visibility Graph (VG) method. A VG is a mathematical representation that transforms time series data into a graph structure. In this graph, each point in the time series is considered a node, and edges between nodes are established based on a "visibility" criterion between points. Specifically, two points in the time series are connected by an edge if a straight-line segment between them does not intersect any intermediate data points, ensuring that each is "visible" to the other in the context of the dataset. Additionally, we can add weights to the edges that are simply computed with Euclidian distance between the “visible” points.

By treating time points as nodes in a graph, GNNs offer a detailed perspective on their interrelationships. In contrast to conventional deep learning models that often struggle with long-term dependencies due to vanishing or exploding gradients, GNNs manage these challenges more effectively. The GNN model architecture is based on the Graph Isomorphism Edge Networks, which were designed to modeled after the Weisfeiler-Lehman isomorphism test, which assesses whether two graphs are isomorphic. In the context of our study, graphs representing similar anomalies often display comparable nodes and connections. Therefore, graphs depicting the same type of anomaly are typically isomorphic. The architecture of the utilized model is presented in Figure 4.

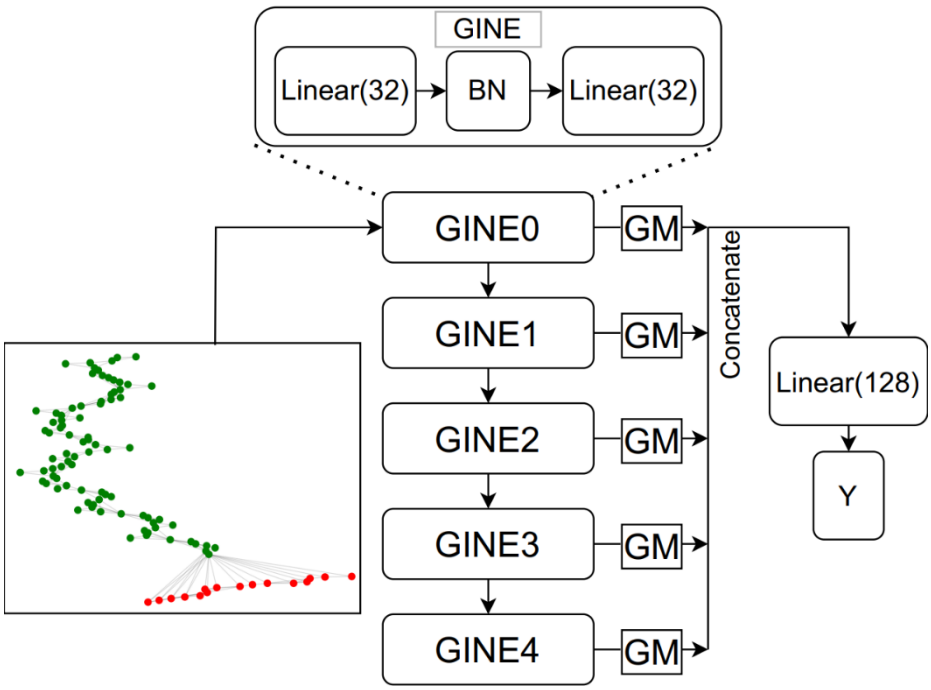

Figure 4: Proposed GNN model architecture, where GINE represents Graph Isomorphism Edge Network, GM represents Global Max Pooling, and BN represents Batch normalization.

2.2.3.Implementation

Model development and training were done with Python programming language. Pytorch was utilized as the primary deep learning library and pytorch-geometric as a library for GNNs. The defined model architecture can be seen in the code snippet below:

```
1. class GINE(torch.nn.Module):
2.     """GINE"""
3.   def __init__(self, dim_h):
4.       super(GINE, self).__init__()
5.       edge_dim = 1
6.       train_eps = True
7.       #dim_h = 32
8.       self.conv1 = GINEConv(
9.           Sequential(Linear(dim_h, dim_h),
                    BatchNorm1d(dim_h), ReLU(),
                    Linear(dim_h, dim_h), ReLU()), edge_dim=edge_dim, train_eps=train_eps)
10.
11.       self.conv2 = GINEConv(
            Sequential(Linear(dim_h, dim_h), BatchNorm1d(dim_h), ReLU(),
                    Linear(dim_h, dim_h), ReLU()), edge_dim=edge_dim, train_eps=train_eps)
12.       self.conv3 = GINEConv(
            Sequential(Linear(dim_h, dim_h), BatchNorm1d(dim_h), ReLU(),
                    Linear(dim_h, dim_h), ReLU()), edge_dim=edge_dim, train_eps=train_eps)
13.
14.       self.conv4 = GINEConv(
            Sequential(Linear(dim_h, dim_h), BatchNorm1d(dim_h), ReLU(),
                    Linear(dim_h, dim_h), ReLU()), edge_dim=edge_dim, train_eps=train_eps)
15.       self.conv5 = GINEConv(
            Sequential(Linear(dim_h, dim_h), BatchNorm1d(dim_h), ReLU(),
                    Linear(dim_h, dim_h), ReLU()), edge_dim=edge_dim, train_eps=train_eps)
16.
17.       self.lin1 = Linear(dim_h*5, dim_h*5)
18.       self.lin2 = Linear(dim_h*5, 5)
19.
20.   def forward(self, data):
21.       x, edge_index, edge_weight, batch = data.x, data.edge_index, data.edge_attr, data.batch
22.
23.       # Node embeddings
24.       h1 = self.conv1(x, edge_index, edge_attr=edge_weight)
25.       h2 = self.conv2(h1, edge_index, edge_attr=edge_weight)
26.       h3 = self.conv3(h2, edge_index, edge_attr=edge_weight)
27.       h4 = self.conv4(h3, edge_index, edge_attr=edge_weight)
28.       h5 = self.conv5(h4, edge_index, edge_attr=edge_weight)
29.
30.       # Graph-level readout
31.       h1 = global_max_pool(h1, batch)
32.       h2 = global_max_pool(h2, batch)
33.       h3 = global_max_pool(h3, batch)
34.       h4 = global_max_pool(h4, batch)
35.       h5 = global_max_pool(h5, batch)
36.
37.       # Concatenate graph embeddings
38.       h = torch.cat((h1, h2, h3, h4, h5), dim=1)
39.
40.       # Classifier
41.       h = self.lin1(h)
42.       h = h.relu()
43.       h = F.dropout(h, p=0.5, training=self.training)
44.       h = self.lin2(h)
45.
46.       return h
```

To transform the time series data into a graph we utilised the ts2vg python library, more specifically function NaturalVG(), to transform data into natural visibility graphs.

The models were trained using pytorch-lightning library with EarlyStopping algorithm, for 1000 epochs with an adaptable learning rate.

To evaluate the computational complexity, PyPapi library was utilised.

Full code is available at:
https://github.com/sensorlab/GNN-TS-anomaly-detection/tree/main

2.2.4.Results

The evaluation results of the model are shown in Table 7: Results of our proposed method compared to the state-of-the-art Hive-Cote2 and InceptionTime . We have compared our model to the state-of-the-art time series classification models Hive-Cote2 and InceptionTime. As can be seen from the figure, our method achieves near perfect F1-score for each of the five classes and significantly outperforms both Hive-Cote2 and InceptionTime.

Table 7: Results of our proposed method compared to the state-of-the-art Hive-Cote2 and InceptionTime models

Class	Our proposed method			Hive-Cote2			InceptionTime		
	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
SuddenD	1.00	1.00	**1.00**	1.00	0.99	0.99	1.00	0.99	0.99
SuddenR	1.00	1.00	**1.00**	0.99	0.97	0.98	1.00	0.99	0.99
InstaD	0.97	0.99	**0.98**	0.97	0.86	0.91	0.98	0.84	0.90
SlowD	1.00	1.00	**1.00**	0.99	0.98	0.99	0.91	0.43	0.48
Normal	0.99	0.99	**0.99**	0.99	1.00	**0.99**	0.93	0.99	0.96

Table 8 compares the methods in terms of computational complexity, where the results show that our method is significantly more computationally efficient, compared to the other two, even if we consider an overhead introduced by the transformation of time series data into graphs.

Table 8: Comparison of computational complexity

Model	Model CC	Transform. CC
Our model	≈10	≈0.051
Hive-Cote2	≈59	/
InceptionTime	≈1135†	/

2.3. Spectrum Activity Characterization

Accurate spectrum monitoring is important for maximizing the performance and efficiency of Open RAN (O-RAN) networks since it provides data on the spectrum occupancy, which could be used by xApps and rApps in the Near-RT RIC and Non-RT RIC accordingly, for optimizing O-RAN parameters. In general platform services for cellular networks that could benefit from such data include Automated Frequency Coordination, Interference management, Energy Efficiency, Quality of Service Optimization and Network Slicing.

By continuously assessing the spectrum occupancy by different Radio Access Technologies (RAT), O-RAN can utilize intelligent applications to dynamically fine-tune network settings and allocate spectrum resources in real-time. Thus, enhancing the overall efficiency and performance of the network and ensuring accommodation of growing data traffic demands.

Employing deep learning (DL) architectures that utilize unsupervised learning approaches, such as self-supervised deep clustering, can significantly enhance the processing and analysis of raw Radio Frequency (RF) time series data, because of the ability to extract knowledge from large unlabeled data. Spectrograms are preferred type of radio data due to their simplicity and effectiveness in low signal-to-noise ratio (SNR) conditions for RAT classification. Spectrograms, structured as 2D matrices, enable the application of advanced machine vision architectures such as Convolutional Neural Networks (CNN). The cross-utilization of techniques from the machine vision domain could further drive innovations in spectrum sensing technologies, which are crucial for advancing next-generation radio networks. The SSL DC approach, used in this work, highlights the potential of unsupervised deep learning to adapt and evolve within the domain of spectrum analysis, advancing the development of robust and efficient spectrum sensing methods.

Utilizing unsupervised deep learning enables network operators to employ advanced analytical techniques to process and analyze complex datasets without extensive manual effort. This approach not only reduces costs and saves time but also facilitates more dynamic and responsive network optimization strategies, considering it provides knowledge about the spectrum bands occupancy and transmission patterns of different RATs, which could potentially lead to more advanced and autonomous wireless network operations.

2.3.1. Monitoring framework and data collection

The dataset[1] used for the analysis consists of fifteen days of spectrum measurements acquired at a sampling rate of *5* power spectrum density (PSD) measurements per second using *1024* FFT bins in the *868MHz* license-free (shared spectrum) SRD band with a *192kHz* bandwidth. The data is acquired in the LOG-a-TEC testbed. Details of the acquisition process and a subset of the data can be found in [14]. The acquired data has a matrix form of *1024xN*, where *N* is the number of measurements over time. By windowing the data with a window size *W*, the resulting raw images for training would have *1024xW* dimension which can be computationally intractable for less capable computing platforms. Therefore, in addition to windowing in time, we are also windowing the dataset in frequency (i.e., FFT bins) direction. This dataset enables the model and functionality development however, in an integrated set-up that embraces O-RAN principles, the collection of metrics and statistics is possible via a publish-subscribe model through open and standardized interfaces such as O1 and E2 [1].

The segmentation of the complete data matrix into non-overlapping square images along time and frequency (FFT bins) is realized for a window size *W=128*. An example of such segmentation containing 8 square images is shown in Figure 5, corresponding to an image resolution of *25.6* seconds (*128* measurements taken at *5* measurements per second) by *24 kHz*. The window size is chosen to be large enough to contain any single type of activity and small enough to avoid having too many activities in a single image while also keeping in mind the computational cost. Dividing the entire dataset of *15* days using *W=128* and zero overlapping, produces *423,904* images of *128x128* pixels, where the pixel values are scaled to the range between *0* and *1*.

From previous work [15], we obtained labels of part of the transmissions in the employed dataset provided by experts. The labels are bounding boxes around transmissions in the same spectrogram

[1] https://github.com/sensorlab/self-supervised-spectrum-sensing

data. We adapt these labels to our use case, considering the data is segmented into square-shaped regions. Each label as a rectangular region in the data matrix covers at least one square segment. All square segments that overlap with the labeled regions are marked as active, i.e., containing transmission(s). Thus, we obtained labeled subset of square-shaped spectrograms that contain transmissions, as marked by experts, which have been used for the subsequent evaluation of the compared models.

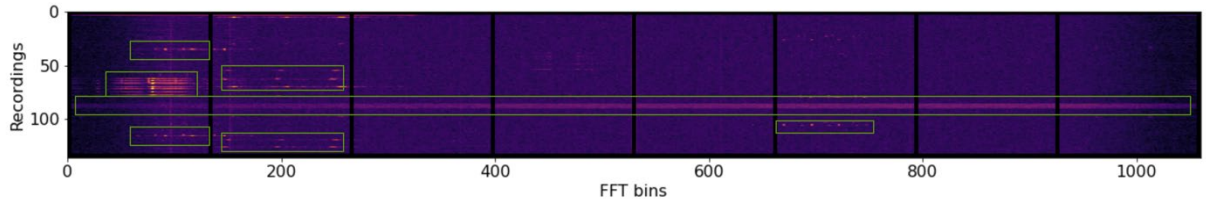

Figure 5: Sample of 8 spectrogram segments from the data.

2.3.2.Model development methodology

As a reference SSL system, we adopted the existing self-supervised *DeepCluster* architecture proposed in [16] for RGB image feature learning based on a VGG (Visual Geometry Group), as a standard deep Convolutional Neural Network (CNN) architecture with batch normalization that is regularly used in computer vision applications. This system alternates between clustering the image features produced by CNN and updating its weights by predicting the cluster assignments.

Inspired by the approach of this reference system, we developed a SSL system depicted in Figure 6, for spectrum activity identification and clustering from spectrograms, characterized by much less content compared to RGB images, thus allowing for significant complexity reduction and performance optimization. As it can be seen from the figure, the system design contains two branches, (1) a so-called unsupervised branch depicted with red arrows and (2) a supervised branch depicted with blue arrows. Both branches contain a feature extractor. In the unsupervised branch, it is followed by a dimensionality reduction (PCA) and normalization (L2) preprocessing block and concluded with the K-means clustering block. In the supervised branch, the feature extractor is followed by fully connected layers as a classifier. Note that even if we refer to the second branch as supervised, it does not rely on actual labelled data. It relies on pseudo-labels that are repeatedly generated by K-means in the unsupervised branch and refined through feedback, i.e., backpropagation. By working in tandem, the two branches realize self-supervision.

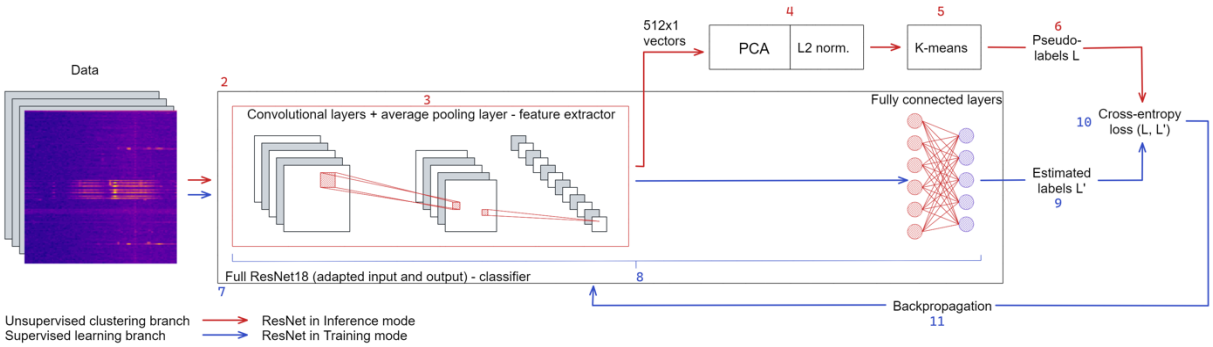

Figure 6: Architecture of the self-supervised CNN system.

Motivated by the findings in [17] we also select ResNet architecture as a possible feature extraction architecture that can replace VGG in the DeepCluster architecture depicted in Figure 6, as it was shown

to perform better in a supervised task of modulation classification. As we show later, with ResNet we achieve very similar performance as with VGG at a significant complexity reduction of roughly 10 times less trainable parameters. From the ResNet family, we use ResNet18 (18 showing the number of convolution layers in CNN) in its original form, but with input and output layers customized according to the shape of the images and the number of classes, i.e., a single input channel in the case of spectrogram images compared to a *3*-channel input required for RGB images that ResNet18 was originally designed for. We also added PCA feature space reduction before clustering to adapt to the significantly lower amount of content in spectrograms compared to RGB images.

The training of the SSL pipeline is as follows: In the first phase, the CNN is initialized randomly and the input data, which consists of image-like spectrum segments, is unlabeled. The clustering algorithm (unsupervised branch, marked with red arrows in Figure 6, is used to cluster the features and provide pseudo-labels (denoted by L in the figure) at the beginning of each training epoch. The initialization of the cluster centers is random. The features are extracted using the *feature extractor* module and have a size of *512x1* for the CNN and *1024x1*. Thus, a descriptor in the form of *512*x*1* vector is obtained for each spectrum segment image. These descriptors are then *PCA*-reduced, *L2*-normalized and finally clustered.

In the second step, the feature extraction and classification modules are used in training mode, as a supervised classification model. The flow of this pipeline activity is indicated by the blue arrows in Figure 6. Using the provided cluster assignments from the previous step as *pseudo-labels (L)* for the input images, the model is trained for one epoch. This completes one iteration of the entire pipeline work cycle. The procedure stops when the predefined number of iterations (training epochs) is reached. In our experiments, we used *200* training epochs. This number was determined empirically by observing the convergence of the loss function.

2.3.3.Implementation

The model was developed using the Pytorch [18] deep learning library. The implementation is available at https://github.com/sensorlab/self-supervised-spectrum-sensing.
The following pseudo-code outlines the deep clustering architecture. It operates over a defined number of epochs, iteratively improving feature representations through unsupervised learning while simultaneously refining classification through supervised learning.

```
1. Number of Epochs = N
2. Iteration = 0
3.
4. while Iteration < Number of Epochs do
5.    Unsupervised branch:
6.       Set Model to EvaluationMode
7.       Extract Features using CNN or ViT
8.       Process Features using PCA + L2 norm
9.       Cluster Features using K-Means
10.      Pseudo-labels (L) - Cluster Assignments
11.
12.   Supervised branch:
13.      Set Model to TrainingMode
14.      Estimate Labels
15.      Estimated Labels (L') - Estimated Labels
16.      Calculate Cross-entropy loss between L and L'
17.      Backpropagation to update classifier weights
18.
19.   Increment Iteration
20. end while
```

2.3.4.Results

The number of clusters is a parameter that should be specified and influences the learning process since it also defines the number of output classes of the supervised branch, which is being trained. Experimentally we identified that the *22*-cluster automatic CNN-based model using ResNet18 DL architecture provides the best results. For this model, the average spectrograms and histograms in Figure 7 show their effectiveness in learning general features related both to the transmission-specific content and the "background" of the spectrograms. Regarding the "background"-related activities, such as the varying noise on the left-most and right-most sub-bands, which result from the variable sensitivity of the equipment. Cluster *2* in Figure 7, occupies the left-most sub-band, while *Cluster 4* corresponds to the samples occupying the right-most sub-band. Additionally, the model learns features that are specific to the different patterns generated by the transmissions. The clusters 0, 1, 5, 6, 9, 10, 13...15, 19...22, in Figure 7, show horizontal line activities, which according to Figure 5 appear across the entire bandwidth. Their histograms for each of these clusters also confirm this observation, showing that samples assigned to these clusters are from all *8* sub-bands, and their distribution along the entire channel is roughly uniform. The clusters *7, 16* and *17* show the capability of the automatic model to distinguish the transmission-free spectrograms. So, samples from these three clusters show the idle spectrum regions. Finally, Clusters *6* and *18* show groups of dot-like transmission bursts.

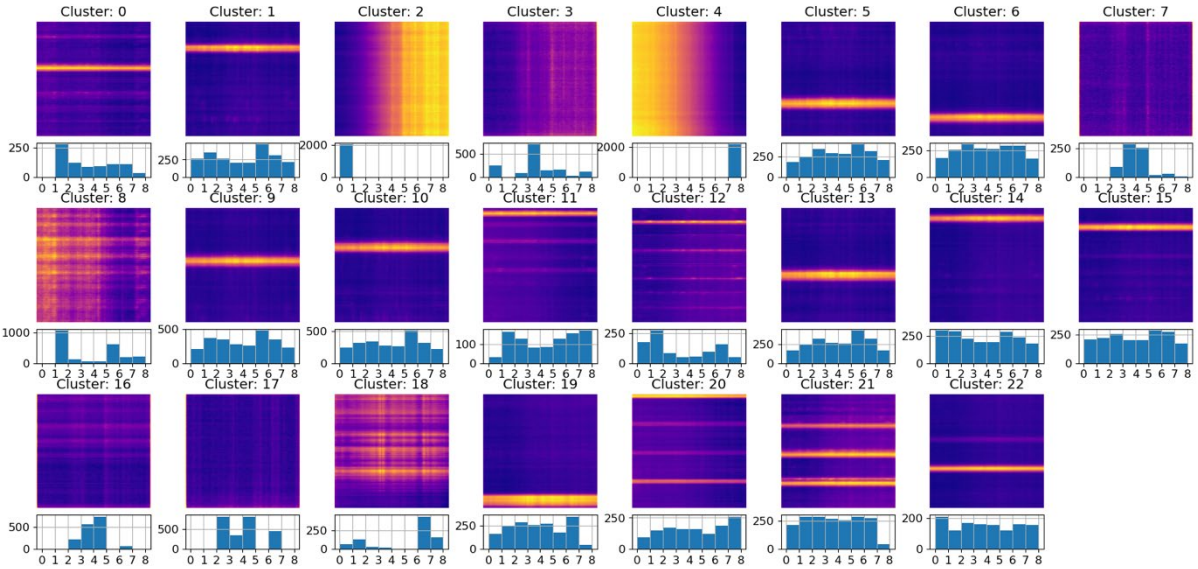

Figure 7: Distribution of samples from each cluster along the frequency band for the clusters obtained by the RN-based approach.

For the evaluation of the models, we consider the task of transmission detection. For this goal, we select the clusters without transmission as *Idle* clusters and the transmission-rich clusters as *occupied.* Information on the spectrum occupancy per sub-band during time could be used for learning the patterns of behavior of the transmissions and employing such knowledge for adjusting the network for opportunistic usage of the spectrum. For such a task, we turn the problem to a binary classification, with the clusters containing *occupied* samples (*0...6, 8...15, 18...22*) representing one class *Active*, and clusters *7, 16* and *17* representing the second *Idle* class. Performance measurement is done with the standard metrics Precision, Recall and F1 score, and the results are summarized in Table 9. The CNN-based models show comparable performance across the variation of clusters. This justifies the usage of the proposed lower complexity CNN, the ResNet18 instead of the VGG11, preserving the performance and significantly reducing the complexity in terms of the number of required DL model parameters by roughly *11* times, according to Table 9.

Table 9: Performance and models' size.

Algorithm	RN	VGG
Precision %	76.0	76.6
Recall %	93.6	94.1
F1 %	83.9	84.5
Num parameters *10^6	11	133

2.4. Throughput Prediction

Currently deployed 5G networks [19] have been called to address a wide range of challenges compared to their predecessor 4G LTE systems. The main challenge was the need for processing, storage and communication of large volumes of data with diverse types and attributes. Furthermore, the support of emerging services and applications that result from novel communication capabilities is another necessity. To that end, performance optimization not only requires revisiting the definition and exploitation of domains such as time, frequency, space and code but also calls upon methods that overcome limitations stemming from computational complexity and dynamic resource demands. The last two are essentially an outcome of massive data communication and massive device connectivity.

Transitioning from the benefits reaped by 5G communications, i.e. lower latency, higher data rates, improved energy and spectral efficiencies along with extended coverage and reliability, 6G aims to go significant steps beyond these [20]. Thus, ultra-low latency, higher data rates, greater energy and spectral efficiencies and massive connectivity in the broad context of "connecting everything" and processing and exchange of huge heterogeneous data, is what 6G networks promise to provide.

Throughout wireless network evolution, the bandwidth metric has served as the dominant theoretical means to quantify the maximum achievable amount of information that can be transmitted, i.e., maximum network capacity in predefined time intervals. Although this is a direct measure of the efficiency of communication that a certain deployed infrastructure possesses, it fails to encompass the practical constraints that reduce this theoretical value. These constraints must be considered in the optimization procedure when mathematically formulating the performance-related objective function to be optimized. As opposed to bandwidth, throughput [21] is what practically defines the amount of information bits transferable in a given scenario and is thus affected by the interplay of several parameters resulting in its derivation. The necessity of calculating throughput as a practical measure of quantifying the amount of information actually transmitted is even more pronounced in the 6G regime which is characterized by massive connectivity in a decentralized scenario. In the 6G ecosystem, large channel variation characteristics, large amount of data communicated and processed, as well as security and privacy are key goals that require revisiting the manner of fulfilling diverse services and applications in a secure, virtualized and reliable context.

Throughput is a most critical performance indicator as it not only reflects practical mechanisms of degrading the theoretic notion of bandwidth, as stated above, but is also tightly coupled with transmission techniques and other performance metrics. Furthermore, it also aids in accurately tracing the limitations of a certain scenario and assists in efficient channel modelling along with resource scheduling and management, which are even more pronounced in decentralized scenarios with dynamic channel and network topologies. Moreover, traffic congestion is related to throughput in a cellular network and is also indicative of the level of interference that must be managed [22], especially in densified 6G cellular networks. Rate outage minimization is also a key objective. Hence, accurately

estimating the dynamics of throughput can directly compensate for outages and enhance coverage in 6G cellular networks.

At densely deployed 6G cellular heterogeneous networks, interference experienced by cell edge users or between macro and femtocells calls for joint optimization together with efficient resource management. However, delays caused by such approaches may hinder performance and degrade user throughput and cell energy efficiency. Towards an accurate statement, throughput is relevant to any performance bottlenecks that may arise in a given 6G decentralized network and thus must be precisely estimated and maximized. This is necessary to achieve optimal performance as to the specific metric that must be optimized, as well as to achieve the required goals, while still tackling the trade-offs involved in system deployment. To that end, O-RAN [23] architectures pose this formidable challenge when combined with the Blockchain component. The initial problem of optimizing throughput is more complex as additional issues, such as decentralized secure data and control information exchange along with the offloading schemes are present. In terms of optimizing resources, the need to efficiently exploit but not exhaust the computation power of the distributed nodes is of paramount importance. This means that the achieved and requested throughput must match.

An important aspect encountered in optimization approaches of radio resource management is the notion of non-polynomial (NP) hardness [24] dictating that suboptimal solutions must be sought. Another reflection of radio resource management is employing game theory where users are perceived as players [25]. Furthermore, traditional optimization includes approaches combining power minimization via measured SINR along with spectrum sharing [26] [27]. Sub-optimality for traditional optimization approaches is also a result of imperfect or practically partial knowledge of 6G network parameters. These mainly include the Channel State Information stemming from the dynamic variations that characterize the channel along with the increased signal attenuation that results from using a higher frequency spectrum in such networks. High mobility patterns are also demanding to effectively model and produce high processing overhead. Another very challenging aspect is the imperfect Quality of Service (QoS) requirement parameter due to the users' diverse requirements in a massive connectivity scenario.

Artificial Intelligence (AI) algorithms [28] bear the attractive potential of reducing computation complexity and avoiding sub-optimal solutions in a manner that traditional optimization approaches failed to accomplish. The key property of AI optimization algorithms mainly relies on capturing dynamic patterns of data processing of high dimensionality. In cases where dataset dependencies involve a multitude of features, where mapping between input and output leads to high dimensional representations or cases with lack of sufficient data, have proven to be effectively addressed, in particular by Machine Learning (ML) optimization algorithms [29]. ML algorithms also confront the non-universality of traditional non-ML optimization algorithms. However, they may also induce complexity issues in cases where training sets must be relatively large so as to accurately train the model. It must also be stressed that, along with feature dependencies, the data serving as input to the ML model are subject to noise and imperfections, which result from the devices collecting the measurement data.

ML has provably emerged as a viable methodology to predict future states of a variable adequately addressing complex optimization problems in the area of B5G/6G [30] communications. Novel technologies integrated into 6G networks that require increased signalling overhead, hardware complexity as well as flexible and dynamic Radio Resource Management (RRM), are needed in a decentralized network security and data privacy framework [31] at the edge. ML is thus employed in a cross-layer optimization perspective to integrate service-aware technologies and achieve enhanced functionalities. The ability of ML algorithms to predict future network Key Performance Indicators

(KPIs) values to better manage resources and meet performance requirements such as QoS has proven to be highly beneficial. These benefits are traced in the context of applying decentralized and edge computing in a Blockchain-assisted implementation [32]. In such a scenario, the optimal use of past measurement data must be leveraged, in order to conduct accurate predictions that will render the 6G network intelligent and efficient.

The prediction of throughput via ML [33] lies within the scope of exploiting the aforementioned benefits of applying ML algorithms for performance optimization. Hence, detecting patterns in dynamically varying datasets and carrying out decision-making with significantly reduced complexity and overhead are the attractive advantages that ML offers in 6G network optimization. Regarding the differentiation of Supervised and Unsupervised Learning [34], the throughput prediction belongs to the Supervised Learning category. In Supervised ML, the prediction is conducted in an attempt to boost network performance. By foreseeing outage events, as a result of User Equipment (UE) throughput prediction, resources can be managed in the context of allocating them to meet QoS requirements and avoid overprovisioning. This is an actual benefit that ML prediction brings. Relevant to this is the prediction of Signal-to-Interference-Noise-Ratio (SINR) values which aids in assessing signal quality while considering interference levels critical to 6G densified cellular infrastructure. On the other hand, unsupervised learning usually involves clustering conducted based on data features. As a brief comment, these methods address the user selection optimization needs, as well as relay node placement in a given cell, so as to enhance cell edge coverage, and also enable interference management. Moreover, throughput prediction can also contribute to Service Level Agreement requirements and ascertain that services are provided to users as agreed upon. As aforementioned, in order to perform throughput prediction, different features of the involved dataset were taken into account so as to accurately assess future throughput values based on the respective training set. Prediction follows a time-based approach, in that the data are provided to the model in a time-series form, along with the features considered.

In the context of the current deliverable, throughput prediction can offer practical benefits to three main services i.e. Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communications (URLLC) and massive Machine Type Communications (mMTC). In the ultra-large-scale deployments, envisaged by overdensified 6G cellular heterogeneous networks, highly dynamic in nature communication environment, user requests and traffic patterns, high mobility patterns and decentralized architecture network complexity in the 6G ecosystem must be accounted for. This also necessitates the use of low-overhead computation resources optimization. This will boost performance, as the security technologies adopted introduce novel issues and trade-offs that must be taken into consideration. Hence, to deploy open, flexible and self-organizing networks, resources must be optimally allocated, as dynamic patterns are captured by ML techniques.

eMBB [35] is a type of service class that mainly requires higher data rates, whereas in highly mobile, decentralized scenarios with edge computing and offloading, information exchange must be secure with enhanced data privacy. Thus, it is straightforward that throughput prediction will enable such a rate-demanding service to achieve stringent latency and extend network functionalities and flexibility; all highly important from the standpoint of seamless connectivity.

URLLC [36] refers to a service class that ultimately dictates low latency communication, a performance indicator that lies at the core of the transition effort from B5G to 6G communications. Thus, low latency together with reliability in a massive connectivity network paradigm can directly benefit from throughput prediction as outage events will be proactively handled, traffic load will be better balanced, resources will be more efficiently delivered, and energy efficiency will be drastically improved. In particular, energy will be better coordinated at a distributed node scale.

Proceeding to mMTC [37], the concept of massively connected devices is tightly coupled with the eMBB and URLLC, as the KPIs to be optimized in the last two are also present in these massive connectivity scenarios together with QoS constraints. As mMTC can be perceived to be related to 6G Zero Touch architecture, in the sense of minimal human intervention, security and privacy in massive connectivity become more of a necessity. By predicting KPIs, such as throughput, the already mentioned requirements through ML are met, thus extending to seamless connectivity, edge intelligence, scalability and security.

Towards an inclusive summarization, throughput prediction is an essential optimization step as when integrated with O-RAN architecture and mobile edge computing will contribute to ultra-secure massive connectivity and efficient RRM. This is due to the representative value of throughput, which reflects coverage loss, resource underutilization and deterioration of network security. ML-based throughput prediction will optimize spectrum sharing. In the 6G regime, using higher spectrum bands will improve rates in the presence of complex mobility patterns. As AI-enabled Blockchain is to introduce a novel secure orchestration and enable proactive self-recovery and self-healing mechanisms, throughput prediction promotes network functionality that will aid AI Blockchain to compensate for attack events via pattern detections in the collected throughput measurements. This will lead to enabling the aforementioned mechanisms. The aforementioned advantage can also be exploited by considering other KPIs, such as latency and outage probability estimation. By improving network functionalities, Blockchain can build upon such a deployment to integrate O-RAN transparency along with trustworthiness. Addressing latency in delay-sensitive use cases can also leverage throughput prediction to speed up network convergence and also aid new services provided to edge users along with the associated computational resource demands. Ultimately, B-RAN deployment will be aided by identifying the trade-offs between decentralization and KPIs ML-based prediction for improved transparency, trustworthiness, availability and real-time decision making.

2.4.1.Monitoring framework and data collection

In the NANCY project, we used the Lumos 5G dataset, which is partially publicly available [19]. In particular, as mentioned in [38]:

- **Location**: The data was collected in Minneapolis, specifically at the Minneapolis-Saint Paul International Airport and around the U.S. Bank Stadium.
- **User Equipment (UE)**: The throughput data was perceived by applications running on user equipment (Android-based devices), which is essential for understanding real-world performance.
- **Features**: The dataset includes various features related to throughput, such as a) coordinates, moving speed and compass direction of UE, b) signal strength (PHY features: Radio Type, LTE Signal Strength, 5G Signal Strength) and c) the position of UE according to the panel (e.g., distance, angle etc.), however the latter in the publicly available dataset is not provided.

Only the part of the dataset named "1300m Loop" is provided in the publicly available dataset, where the recordings took place on different days in the same region. In total, there are 118 different recordings with the UE. This dataset enables the model and functionality development however, in an integrated set-up that embraces O-RAN principles, the collection of metrics and statistics is possible via a publish-subscribe model through open and standardized interfaces such as O1 and E2 [1].

2.4.2.Model development methodology

First, it is important to outline the pre-processing stage of the Lumos 5G dataset (1300m Loop), before describing in detail the proposed method for predicting upcoming throughput values (in Mbps). As mentioned in [38], the dataset includes many missing and noisy data; therefore, in order to overcome these limitations, **linear interpolation** [39] and the **moving average (MA) filter** [40] were utilized in all the time series (recordings). In particular, the linear interpolation method was used to "estimate" the missing values of the features related to the signal strength. However, there were 5 recordings where most of the features associated with the signal strength were missing; hence, these recordings were eliminated. In addition, in the case where the values of signal strength features were out of their reasonable boundary limits, these values were considered missing. Finally, the throughput signals (for all recordings) include noise, mainly due to the imperfection of UE collecting the throughput measurements. Hence, the MA filter with time window *5* was used for every recording to overcome this limitation. In Figure 9, an indicative example of the proposed denoising process is illustrated, where the original normalized throughput is depicted by the blue curve and filtered by the orange curve. It is clear that the MA(5) filter applied denoising to the original throughput measurements without introducing a delay (lag). In case where a higher time window is used, e.g., MA(10), the filtered signal introduced lags and thus was excluded from the implementation. Next, only the filtered throughputs were used.

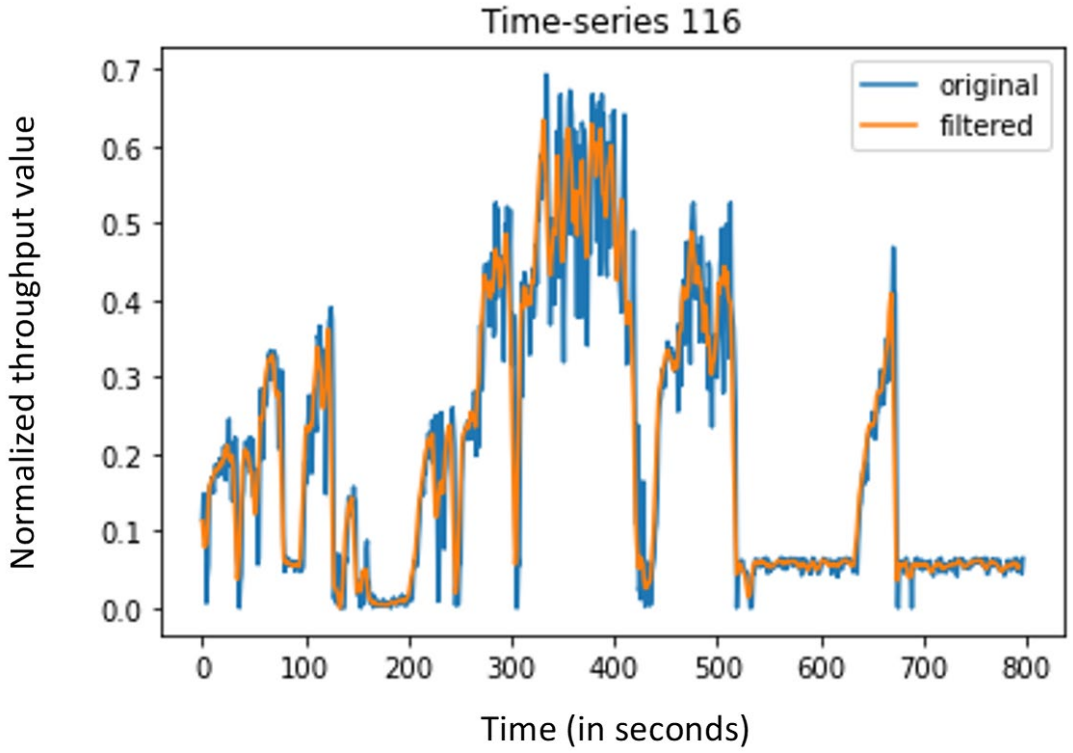

Figure 8: Original and filtered throughput of recording number 116 with MA(5)

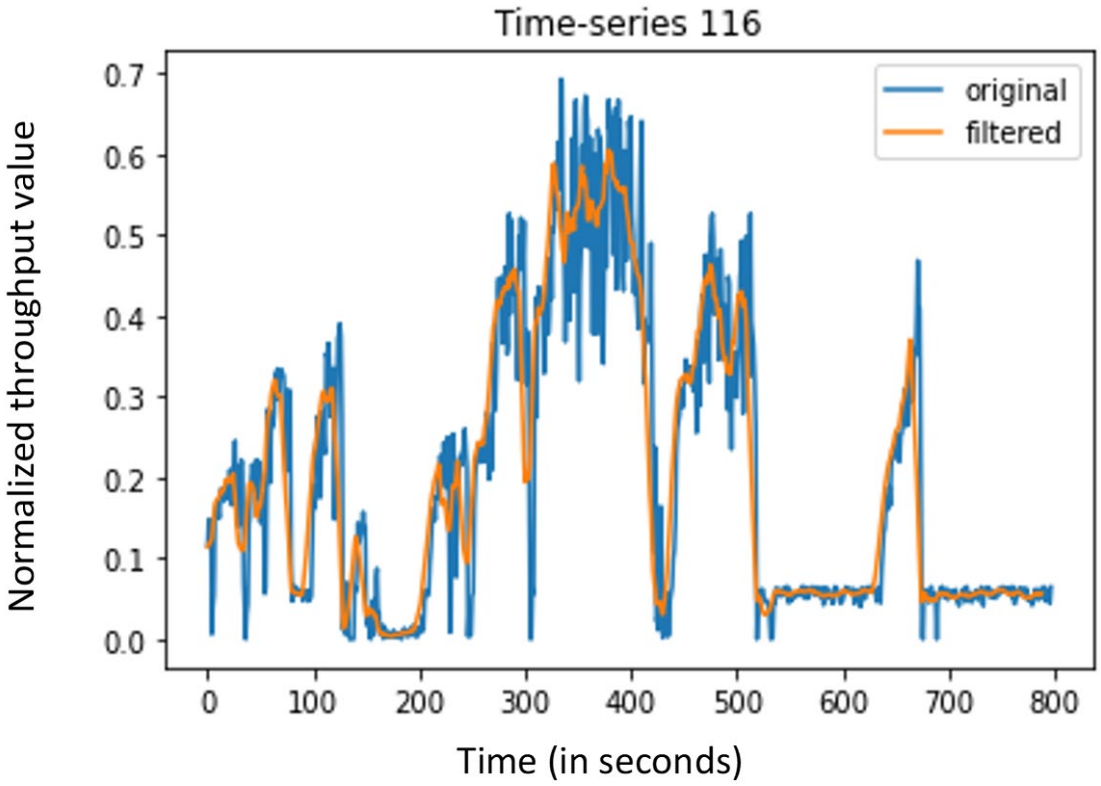

Figure 9: Original and filtered throughput of recording number 116 with MA(10).

The scope of this task is to provide future predictions -in other words, forecasting- of throughput for the next 20 seconds (as in [38]) using the past values of features described in subsection 2.4.2. Next, we present 3 DL-based models that can handle sequence input and provide sequence output (predictions for the next 20 steps-seconds).

a. LSTM

The proposed LSTM model consists of 2 LSTM layers (the firstone with return_sequences=True, while the second with return_sequences=False), and the output of each LSTM-layer is followed by a dropout layer for regularization. Finally, two Dense layers (without any activation function) are used to produce the final output sequence.

b. seq2seq

The seq2seq model [41] consists of 3 layers: the encoder, decoder and output layer.

- The encoder layer includes an LSTM layer that processes the input sequence and returns its final hidden state (ouput_state, hidden_state, cell_state).
- The decoder layer starts with a RepeatVector layer that repeats the encoder's last hidden_state across the time steps of interest (in our case, 20 steps ahead). This is followed by a dropout layer for regularization. Then, another LSTM layer generates the output sequence from the repeated vector. The initial states of this LSTM layer are set equal to the last states provided by the encoder layer. A final dropout layer is added before the output layer.
- Finally, in the output layer, a Dense layer (without any activation function) produces the final output sequence.

c. seq2seq with attention Luong

The seq2seq model with Luong attention mechanism [42] consists of 4 layers: the encoder, decoder, attention, and output layer.

- The encoder and the decoder layer are the same as in the seq2seq model.
- The Luong attention mechanism takes into account the output_states of the encoder and decoder layer to calculate the context vector as a weighted sum of the encoder's output states, using the attention weights. Next, the context vector is concatenated with the decoder's output state.
- Finally, the Dense layer provides the final output sequence.

Next, details about the training stage are provided. In particular, in all models, the following apply:

- Hidden units for LSTM layers are set equal to 100.
- Dropout equal to 0.2.
- Adam optimizer has been used with mean square error as a loss function.
- The first 80% of the recording (time-series) is set as the train set, while the rest 20% is the validation and test set (10% for each one).
- An early stopping technique has been used with epoch patience equal to 10.
- Batch size equal to 256.

It is worth mentioning that the split (train, validation, and test set) described earlier has been performed in all the time series (118 recordings in total), therefore, it is ensured that the model will be evaluated in all recordings and scenarios.

2.4.3.Implementation

The function *clean_df* gets as input all the features (df in dataframe format) and the interpolation method (intrp). If there are missing values (or nan in df) the output is the interpolated df, otherwise it is the original df.

```
1. def clean_df(df, intrp):
2.
3.    # check features with nan values
4.    col_nan = [df[col].isnull().any() for col in df.columns]
5.
6.    # find the features with nan values
7.    indx_trg = [i for i, x in enumerate(col_nan) if x]
8.    col_trg = list(df.columns[indx_trg])
9.
10.    #interpolation
11.    df_new = df.copy()
12.    df_new[col_trg] = df[col_trg].interpolate(method=intrp)
13.
14.    # check cases with nan and discard them
15.    ls = [df_new[col].isnull() for col in col_trg]
16.    max_index_nan = [np.max(l[l].index) for l in ls]
17.
18.    if col_trg == []:
19.        return df_new
20.    else:
```

```
21.       if np.nanmax(max_index_nan) > -1:
22.          t = int(np.nanmax(max_index_nan))
23.       else:
24.          t = -1
25.       return df_new[t+1:]
```

The moving average filter is applied using the command *df_avg = df.rolling(window=5).mean().*

Next, the three DL models are provided. The proposed LSTM model is provided where the *_build_model* includes the main part of the LSTM model. The *input_shape* and *output_shape* include the length size of the input and output, respectively, which in our case are both equal to 20, while n_*hidden* stands for the hidden units.

```
1. class lstm:
2.    def __init__(self, n_hidden, input_shape, output_shape):
3.       self.n_hidden = n_hidden
4.       self.input_shape = input_shape
5.       self.output_shape = output_shape
6.       self.model = self.build_model()
7.
8.    def build_model(self):
9.       inp = Input(shape=self.input_shape)
10.
11.       lstm_1 = LSTM(self.n_hidden, return_sequences=True)(inp)
12.       lstm_1 = Dropout(0.2)(lstm_1)
13.
14.       lstm_2 = LSTM(self.n_hidden, return_sequences=False)(lstm_1)
15.       lstm_2 = Dropout(0.2)(lstm_2)
16.
17.       dns = Dense(units=round(self.n_hidden/2))(lstm_2)
18.       out = Dense(units=self.output_shape[0])(dns)
19.
20.       model = Model(inputs=inp, outputs=out)
21.       return model
22.
23.    def compile_model(self, loss='mean_squared_error'):
24.       self.model.compile(loss=loss, optimizer=Adam(learning_rate=0.001))
25.
26.    def fit(self, x_train, y_train, epochs, batch_size, validation_split=0.1):
27.       es = Earlystopping(monitor="val_loss", patience=10, restore_best_weights=True)
28.       self.model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=validation_split,
callbacks=[es], verbose=1)
29.
30.    def summary(self):
31.       self.model.summary()
```

In the same way, the seq2seq and seq2seq Luong models are provided.

```
1. class seq2seq:
2.    def __init__(self, n_hidden, input_shape, output_shape):
3.       self.n_hidden = n_hidden
4.       self.input_shape = input_shape
5.       self.output_shape = output_shape
6.       self.model = self._build_model()
6.
7.    def _build_model(self):
8.       encoder_inp = Input(shape=self.input_shape)
9.       encoder_last_h1, encoder_last_h2, encoder_last_c = LSTM(self.hidden, activation='tanh', return_sequences=False,
return_state=True)(encoder_inp)
10.       decoder_inp = RepeatVector(self.output_shape[0])(encoder_last_h2)
```

```
11.       decoder_inp = Dropout(0.2)(decoder_inp)
12.       decoder = LSTM(self.hidden, activation='tanh', return_state=False, return_sequences=False)(decoder_inp,
initial_state=[encoder_last_h2, encoder_last_c)
13.       decoder = Dropout(0.2)(decoder)
14.
15.       out = Dense(self.output_shape[0])(decoder)  # for return sequences
16.       model = Model(inputs=encoder_inp, outputs=out)
17.       return model
18.
19.    def compile_model(self.optimizer='adam', loss='mean_squared_error')
20.       self.model.compile(loss=loss, optimizer=optimizer)
21.
22.    def fit_model(self, x_train, y_train, epochs, batch_size, validation_split=0.1):
23.       es = Earlystopping(monitor='val_loss', patience=10, restore_best_weights=True)
24.       self.model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=validation_split, verbose=1,
callback =[es])
25.
26.    def summary(self):
27.       self.model.summary()
```

```
1. class seq2seq_luong:
2.    def __init__(self, n_hidden, input_shape, output_shape):
3.       self.n_hidden = n_hidden
4.       self.input_shape = input_shape
5.       self.output_shape = output_shape
6.       self.model = self._build_model()
6.
7.    def _build_model(self):
8.       encoder_inp = Input(shape=self.input_shape)
9.       encoder_stack_h, encoder_last_h, encoder_last_c = LSTM(self.n_hidden, activation='tanh', return_sequences=True,
return_state=True)(encoder_inp)
10.       decoder_inp = RepeatVector(self.output_shape[0])(encoder_last_h)
11.       decoder_inp = Dropout(0.2)(decoder_inp)
12.
13.       decoder_stack_h = LSTM(self.n_hidden, activation='tanh', return_sequences=True,
return_state=True)(decoder_inp, initial_state=[encoder_last_h, encoder_last_c])
14.       decoder_stack_h = Dropout(0.2)(decoder_stack_h)
15.
16.       attention = dot({decoder_stack_h, encoder_stack_h], axes=[2, 2])
17.       attention = Activation('softmax')(attention)
18.       context = dot([attention, encoder_stack_h], axes=[2,1])
19.       decoder_combined_context = concatenate([context, decoder_stack_h])
20.
21.       out = tf.squeeze(TimeDistributed(Dense(1))(decoder_combined_context), axis=-1)
22.       model = Model(inputs=encoder_inp, outputs=out)
23.       return model
24.
25.    def compile_model(self.optimizer='adam', loss='mean_squared_error')
26.       self.model.compile(loss=loss, optimizer=optimizer)
27.
28.    def fit_model(self, x_train, y_train, epochs, batch_size, validation_split=0.1):
29.       es = Earlystopping(monitor='val_loss', patience=10, restore_best_weights=True)
30.       self.model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=validation_split, verbose=1,
callback =[es])
31.
32.    def summary(self):
33.       self.model.summary()
```

2.4.4.Results

The performance measure of the three DL methods considered for predicting the next 20 seconds of throughput is the root mean score error (RMSE). RMSE is calculated for all the time series and for all 20 predictions. In order to define which features (or feature groups) provide better performance when predicting throughput, we have conducted 4 experimental set-ups. In each experimental setup, different groups of features are utilized. More specifically, we define according to the Lumos 5G dataset [38]:

- **Basic (B) group,** which includes *Throughput*, *nrStatus, latitude, longitude* and *abstractSignalStr*
- **Positional (P) group,** which includes *moving speed* and *compass direction*
- **Signal Strength (S) group,** which includes *lte_rssi, lte_rsrp, lte_rsrq, nr_ssRsrp, nr_ssRsrq*, and nr_*ssSinr* features where:
    - abstractSignalStr: Indicates the abstract signal strength as reported by Android API
    - nrStatus: Indicates if the UE was connected to 5G network or not
    - lte_rssi: Get Received Signal Strength Indication (RSSI) in dBm of the primary serving LTE cell
    - lte_rsrp: Get reference signal received power (RSRP) in dBm of the primary serving LTE cell
    - lte_rsrq: Get reference signal received quality (RSRQ) in dB of the primary serving LTE cell
    - nr_ssRsrp: Obtained by parsing the raw string representation of `SignalStrength` object with range: -140 dBm to -44 dBm
    - nr_ssRsrq: Obtained by parsing the raw string representation of `SignalStrength` object with range: -20 dB to -3 dB
    - nr_ssSinr: Obtained by parsing the raw string representation of `SignalStrength` object with range: -23 dB to 40 dB.

Next, the MinMax normalization method was used on all features to deal with the different ranges of their values. Finally, it should be noted that taking the first differences of the time series (throughput values) is a common method [23], to remove trends and make the time series stationary. At the end of the proposed framework, the predictions were provided on the original scale.

Table 10 illustrates the overall performance (for all the time series and time-steps) of the DL models, each one of which uses a different features' group as input. It can be seen that the LSTM model using the Basic and Signal Strength features group provides the best performance with RMSE equal to 239 Mbps (for all time series and time-steps). Furthermore, it can be concluded that, including the Positional feature group, the results worsen for all DL methods (RMSE increases), while the Signal Strength group features improve the performance.

In Figure 10, the predictions of the LSTM model (using B and S feature groups) for 20 seconds ahead are illustrated. It is clear that in the first seconds (4-7 steps) the predictions (orange) are very close to the original values of throughput (blue), while, as time progresses, there is a greater discrepancy. Next, in Figure 11, the performances of the three DL models (using B and S feature groups) are illustrated for each time step. It can be seen that all methods for the first time-steps, i.e., 4-7 seconds, provide almost the same performance. However, as time progresses, the LSTM model provides the best performance at each time step. Thus, within the NANCY project, the proposed prediction model is the LSTM model using the B and S feature groups.

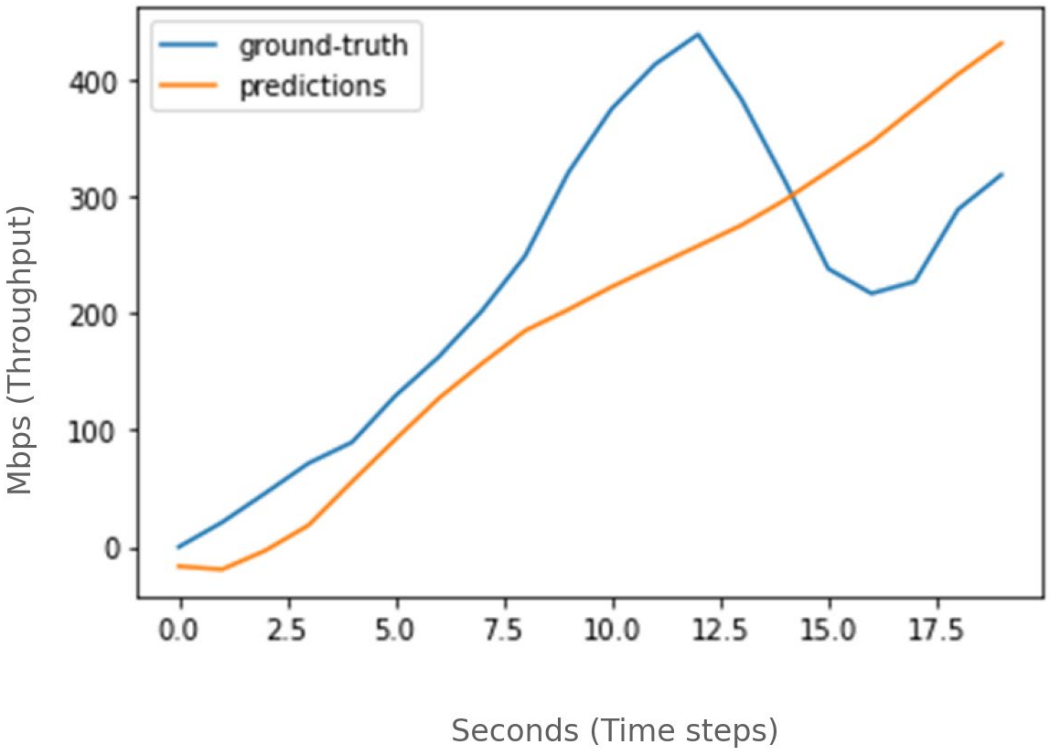

Figure 10: The predictions (orange) and the original data (blue) of throughput for 20 seconds (steps)

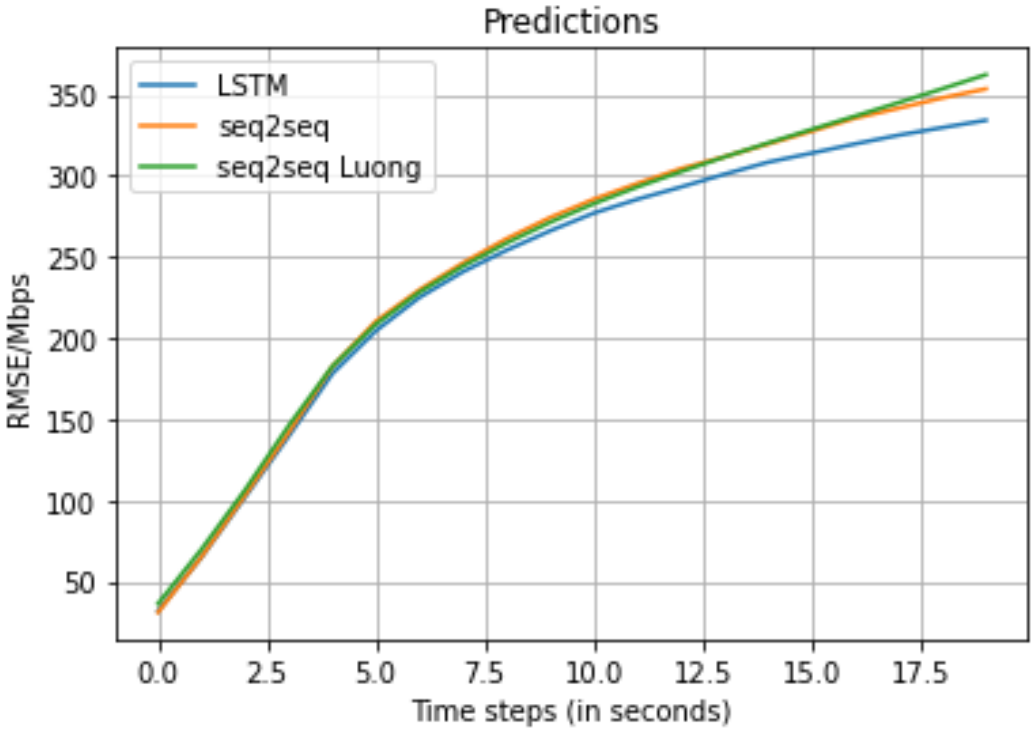

Figure 11: The performance (RMSE) at each time step for the three DL-models using the **B** and **S** feature groups

Table 10: The performance (RMSE) of the three DL models for four different experimental set-ups

	LSTM	seq2seq	seq2seq Luong
B + S + P	267 Mbps	258 Mbps	290 Mbps
B + P	276 Mbps	264 Mbps	271 Mbps
B + S	239 Mbps	248 Mbps	249 Mbps
B	247 Mbps	260 Mbps	259 Mbps

3. Technology Stack

3.1. Models in the Self Evolving Model Repo

Adapting existing machine learning AI/ML models and procedures into a production AI/ML workflow is necessary for effective retraining, storage, and deployment of ML models. This section outlines the key processes involved in managing ML models for the Radio Access Network (RAN) environments enabled by the Self-Evolving Model Repository (SEM). SEM offers the automation of AI/ML workflows for AI-native networks as described in D3.3 'NANCY AI-based B-RAN Orchestration'.

The lifecycle of ML models of a 6G platform and user services running on MEC or O-RAN involves several components, starting with data storage and transformation. The storage of collected and transformed data is vital for model training.

To maintain performance, models must be trained with the latest data. SEM handles this data storage, providing the latest datasets whenever models need retraining. For example, in RAN environments, SEM manages and transforms metrics such as Channel Quality Indicator (CQI), Signal-to-Interference-plus-Noise Ratio (SINR) and Received Signal Strength Indicator (RSSI) and stores this transformed data such as visibility graphs as features for model training and retraining.

SEM automates the retraining process, ensuring models are continuously updated and improved according to KPIs. This keeps models relevant and accurate for dynamic network environments as the retraining is performed on the newest data. Once trained, models require a model repository. This repository includes versioning, allowing actors to track and fetch different versions of models if needed. SEM provides the capability for requesting models by name and version. SEM also enables the deployment of trained ML models as a service. Models stored in the repository can be deployed according to user preferences and specific use cases, such as detecting wireless signal anomalies to ensure network stability or spectrum activity analysis to maximize the performance of RAN networks. Effective model deployment together with automated retraining ensures that the most appropriate model is utilized, enhancing the application's overall performance.

Integrating MEC services and ML procedures into SEM is straightforward and can be done with minimal adjustments. The process can be separated into components:

Data Preparation: Data collection and preparation code can be adapted to work within SEM's architecture, ensuring all necessary data is cleaned, transformed, organized, and ready for model training and retraining stages. For instance, transforming RSSI time series data into graphs using the visibility graph method for anomaly detection models. In this example, a data transformation procedure must be integrated as a task in SEM, collecting data from the network and storing the transformed data in SEM data storage.

Adapting Training Procedures: The procedure for model training can be configured to align with SEM's template for model training and retraining. The code must be adapted to SEM's training tools and configured to utilize the required computational resources. Once trained, models are autonomously registered and versioned in the model repository for deployment and future retraining. For example, training a GNN model to classify anomalies in network performance metrics involves integrating the model training scripts with SEM's model training task by specifying the model architecture and training function. The anomaly detection models are stored in the model repository by name and version and can be retrieved for retraining on updated data by running the training function on the stored model and new data.

Deployment as a Service: After models are stored in the repository, they are deployed as a service on SEM. Inference (model predictions) is handled directly within SEM. This centralized approach simplifies the deployment process and ensures reliable performance across the network. Deployment involves configuring endpoints for the models, setting up metrics to track inference performance, and configuring automatic scaling based on network demand. An example use case includes deploying an anomaly detection model that classifies RSSI anomalies to preemptively address network issues and deploying a link anomaly detection model as a service, preventing issues, that could lead to network disruptions. The ML model is loaded from the model registry, configured into prediction mode, and deployed on the network. As the demand for model predictions increases, SEM autonomously handles horizontal autoscaling of the model replicas.

Self-evolving MEC services and model retraining: SEM allows for the collection of KPIs from deployed models and the network. To support retraining decisions, the collection and analysis of KPIs for specific ML procedures need to be developed. Based on model performance, a decision can be made to retrain a model with new data and redeploy it. This process ensures that models continuously evolve and improve, maintaining their effectiveness in real-world applications. For example, retraining a GNN anomaly detection model or a CNN spectrum activity monitoring model with new data that reflects recent changes in network conditions involves collecting KPIs, setting thresholds for retraining triggers, and automating the data pipeline to collect and transform the latest metrics seamlessly.

3.2. RAN enabled protocol stack

To deploy developed services, we utilise the Slice Manager (SM). The SM is responsible for managing the lifecycle of network slices, which includes creating, maintaining, and tearing down these slices. It ensures that network slices are allocated resources, remain isolated from each other, and maintain dedicated connectivity. The services need to be containarized and deployed with the SM's Open Source MANO (*OSM*), through the Helm charts as discussed in the following.

```
# Imports
# ...

# SEMR's model store endpoint
os.environ['MLFLOW_TRACKING_URI'] = 'http://<SEMR_IP>:31007'

# Model version from the env variable
model_version = os.getenv('MODEL_VERSION', "latest")

# Modify the model_uri to point to the correct model name
model_uri = f"models:/CNN_spectrum/{model_version}"

# Modify the mlflow loading function
model = mlflow.pytorch.load_model(model_uri)
model.eval()
print("Model loaded!")

@app.post("/")
async def echo(data: List[List[List[float]]]):
    # Data transformations
    tensor_data = torch.tensor(data)
    tensor_data = tensor_data.unsqueeze(0)

    # Inference
    cnn_labels_array = cnn_predict(tensor_data, model)
    counts = Counter(cnn_labels_array)

    return {"prediction": str(counts)}
```

Figure 12 An example of a Docker image python template for Spectrum Activity Characterization MEC service

To containerize a developed service for model inference, the Docker image has to be created. As seen in Figure 12, the developed models can deployed with minimal Python code. First, we have to define the SEMR's model stores endpoint, which connects to the MLFlow API. Second, we define which version of the model from the repository we would like to deploy and URI to the model we would like to deploy. Through the MLFlow wrapper, we load the model from the SEMR. Finally, we have to define the API point for the service and define the input and output parameters of the model. The model inference code is then integrated into the existing Dockerfile, ensuring that all dependencies are managed appropriately. The requirements (such as libraries) for model inference should be appended to the requirements.txt file, which includes all necessary libraries and frameworks. Once the requirements.txt file is updated, these dependencies need to be imported into the Docker image during the build process. Subsequently, the Docker image should be built using the Docker CLI or an automated CI/CD pipeline to ensure consistency and reproducibility. Finally, the built Docker image must be pushed to a Docker registry, such as Docker Hub or a private repository, to make it accessible for deployment with the SM. This systematic approach ensures that the model inference service is containerized efficiently, facilitating scalable and consistent deployment.

The containarized service then has to be defined as a Helm chart as shown in Figure 13. In the Helm chart, we first defined the number of replicas of the service, then we provide a detailed description of the built MEC service container and define the version of the model we would like to load from the SEMR (value »latest« ensures that the most recent model is deployed). Next, we can also define the CPU architecture on which the container should run and the service endpoint for the access through an API call. The Helm charts are then onboarded to the SM's OSM and can be instantiated and configured using the SM's API calls.

```yaml
deployment:
  replicas: 3

container:
  image: copandrej/cnn-spectrum-endpoint-arm # Container image
  port: 8000
  env:
    - name: MODEL_VERSION
      value: "latest"

nodeSelector:
  arch: arm64 # Deploy to edge ARM nodes

service:
  targetPort: 8000 # equal to container port
  nodePort: 30070 # service endpoint
```

Figure 13 Template of a Helm chart to deploy the MEC service

Using SM's API, the developed services can be instantiated and configured using Helm values overrides. These overrides specify crucial parameters such as the model version, Docker image, service port, number of replicas, and other necessary configurations required by the service as illustrated in Figure 14. When a new model version is uploaded to the SEMR (indicating that the models have been retrained), the inference service can be re-instantiated using SM's API with updated configurations through values overrides. This re-instantiation process ensures that the latest model version is deployed without the need for additional modifications to the Docker images or Helm charts. This capability streamlines the update process, allowing for seamless integration of retrained models into the target environment, thereby maintaining operational efficiency and minimizing downtime.

3.3. MEC enabled protocol stack with ARM hardware architecture

In the context of B5G networks, the edge domain comes to the foreground as it plays a pivotal role in the transformation of the architecture. New paradigms like the MEC enable the deployment of services closer to end-users, leveraging the computational capabilities of powerful edge devices and reducing latency. What is more, the computational tasks will be deployed to the edge devices in the form of Virtual Network Functions, as the NFV paradigm mandates. As a matter of fact, Virtualization is a key element of the new radio networks, because it brings flexibility and scalability in the network management thanks to the decoupling of the various functions from the proprietary hardware.

For the edge devices to be exploited to the greatest extent, NANCY has to consider the current architectural nuances within the embedded domain. Specifically, NANCY focuses on the peculiarities of the ARMv8 architecture as it is very suitable for edge computing due to its inherent capabilities to provide an energy-efficient way, as well as fast and parallel processing of data-intensive tasks, such as image processing, machine learning and cryptography. In this regard, Virtualization in ARMv8 is nowadays much focused towards isolating critical requirements like security and real-time processing from non-critical ones, and NANCY is taking this into account.

In the core of the ARM architecture, there exist the Trustzone hardware security extensions. These extensions provide support for complete hardware-supported isolation of critical tasks from non-critical. Upon these extensions, the VOSySmonitor software product is built, with the aim of offering transparent co-execution of mixed-critical tasks. In other words, VOSySmonitor makes it possible to execute at the same time, in the same physical hardware and operating systems with different levels of criticality. VOSySmonitor is actually a firmware, that runs first in an ARM-based system and partitions it in a way it consolidates both a critical and a non-critical OS. Also, it should be noted that although the ARMv8 architecture supports two levels of criticality, it is feasible to deploy more than two OSes on the same hardware platform.

To provide more technical details, VOSySmonitor runs at the highest execution level of the ARM architecture and is a critical task. This gives it the privilege to control and manage the resources of the system, as well as the allocation of resources to the various OSes. When the system boots, VOSySmonitor holds the responsibility to instantiate many of the system peripherals, given the use case and the configuration it needs to apply for the critical and the non-critical worlds. The safety-critical, or Secure world always has priority over the Non-Secure world, and VOSySmonitor guarantees that with its internal policies. In the context of NANCY, we introduce the term “Compartment”, to refer to the bare-metal OSes with the different critical levels that are deployed on top of the physical hardware.

To bind this architectural overview to NANCY requirements, NANCY exploits the ability to have separate, isolated Compartments that execute simultaneously, with some of them critical and some not, by installing a VNF per compartment. Network services like localization, anomaly detection and others, will be run in dedicated Compartments and will be completely isolated. With this setup, NANCY achieves improved transparency, because the VNF runs isolated in the Compartments without affecting the rest of the system. Also, the solution achieves bare-metal execution times, because each Compartment is deployed directly in the physical hardware. At the same time, VOSySmonitor has to ensure that it is able to accommodate the level of abstraction required by the VNFs, which will have to interface with a generic view of the hardware.

Zooming a bit out (see Figure 14), VOSySmonitor eventually has to accept the VNF deployment requests by the 5G infrastructure. The connection point between a high-level orchestrator, like

OpenStack, with VOSySmonitor is implemented through a new component called the vManager. The vManager talks with a local instance of an enforcing agent, such as libvirt, to accept the necessary configurations and finally apply them by appropriately instructing VOSySmonitor. NANCY implements the vManager, designed to specifically interact with libvirt, and also integrates the necessary modifications to VOSySmonitor, in a way that it becomes capable of deploying the VNFs dynamically in the Compartments upon request. Summing up, VOSySmonitor and the vManager serve as enabling technologies to offer NANCY a low-latency Virtualization solution, striving to comply with the natural characteristics of the embedded domain at the network edge.

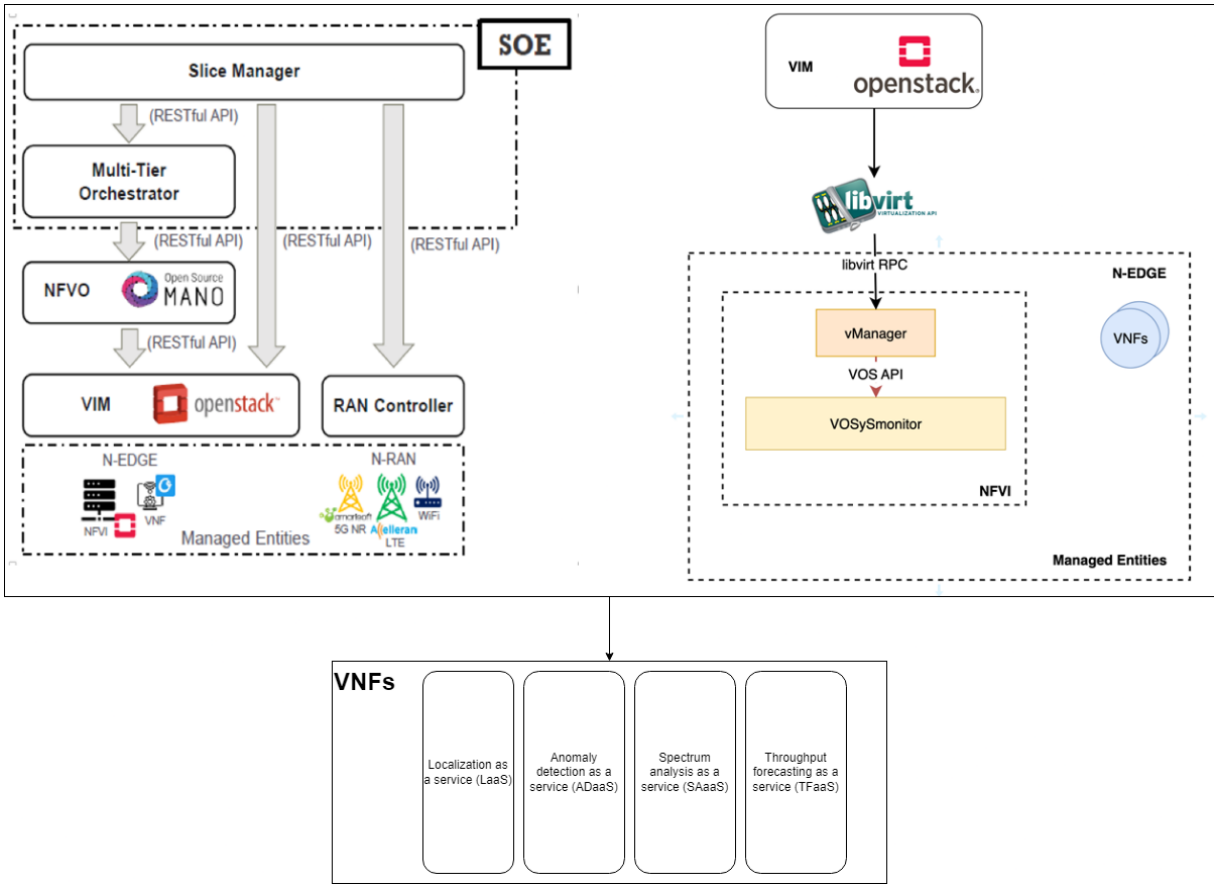

Figure 14: VOSySmonitor overview

3.3.1.Monitoring

The monitoring of resources at the edge of the network is essential for NANCY, in order for the Orchestrator entities to proceed into making informed decisions about the best exploitation of these resources.

For VNFs that are deployed in the bare-metal, isolated ARMv8 Compartments, the monitoring of resources at the VNF level can be achieved with all the usual Linux tools. For this, it should be ensured that the deployed VNFs in the Compartments are Linux-based images, which is mandated by the Orchestrators which is usually the case for NFV deployments.

More specifically in NANCY, where the high-level Orchestrator usually integrates with a Telemetry system, the open-source Prometheus tool will be exploited.

In detail, Prometheus comes with a Linux tool called *node_exporter*, which when properly configured can scrape metrics such as CPU, Memory, Disk utilization as well as Network information from a Linux system.

In its internals, the *node_exporter* interfaces with multiple utilities of the Linux system, which are called Collectors. To provide an example, some of the by-default enabled Collectors in the *node_exporter* configuration are the *cpu, meminfo, diskstats, netdev, netstats* utilities and others.

Being a Telemetry system, Prometheus can handle the data to higher-level entities of NANCY via an Orchestrator like OpenStack. Also, Prometheus integrates well with tools like Grafana, which can be proven useful in order to visualize the utilization of resources.

3.4. Limiting the underutilization of virtualized resources

The services introduced in Section 2 originate diverse computing workloads with heterogeneous computational requirements that, if not properly handled, may lead to underutilization of computing resources. Most relevant to NANCY is the case in which those services are hosted within containers or virtual machines. While various interpretations of the terms container and virtual machines exist, they typically refer to an enclosed operational environment that houses one or more processes/threads. Furthermore, virtual machine monitors that run on a host system consist of processes/threads themselves.

Traditional resource management and CPU scheduling algorithms, as used by containerization frameworks, may indeed be ineffective in properly assigning computing resources to the processes/threads spawned by the NANCY services while preserving latency requirements.

For example, a popular approach consists of assigning an application to a portion of the available computational resources (e.g., cores) for being used exclusively. Nevertheless, such coarse-grained allocation has the disadvantage of always assigning an integer fraction of the physical cores to execute, often causing an underutilization of the computing platform, especially in the presence of lightweight workloads. Alternatively, applications can share processing cores at the cost of being exposed to interferences caused by other untrusted applications, which may compromise real-time guarantees.

The SCHED_DEADLINE [43] scheduling policy implemented in the Linux kernel, jointly used with principled allocation and configuration algorithms, can help tackle this problem thanks to its resource reservation capabilities that allow reserving a given amount of CPU time within periodic time windows. This allows providing real-time guarantees to applications sharing processing resources. Each of these capabilities assigned to processes/threads is also referred to as CPU *reservation* (RSV) in short. The scheduling policy allows both controlling the CPU quota reserved for processes/threads as well as their maximum service delay given by the time span in which the protected process/thread is not assigned CPU resources.

The scheduling architecture when SCHED_DEADLINE is used to manage the workloads of a set of containers is illustrated in Figure 15 (from [44]).

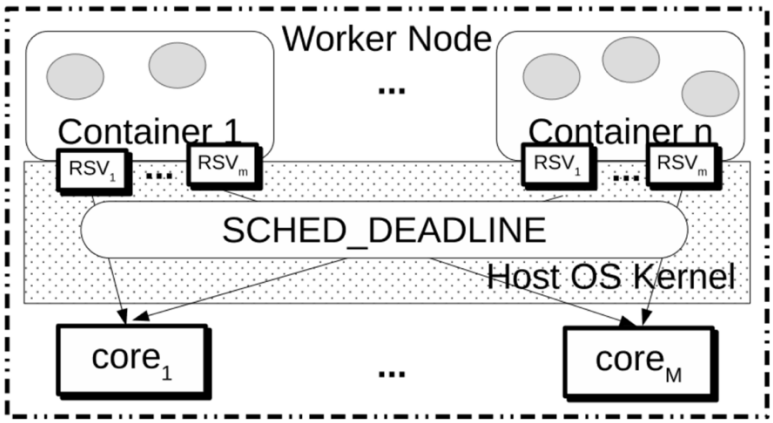

Figure 15: SCHED_DEADLINE scheduling architecture

Resource reservation and, more specifically, SCHED_DEADLINE serve as a proactive method to prevent resource underutilization caused by CPU resource contention. Indeed, by controlling with fine granularity the CPU quota and service delay reserved for containerized services, it is possible to both

- (i) guarantee that some service does not excessively use CPU resources beyond the amount required to provide the desired quality of service; hence avoiding underutilizing CPU resources to perform computations that are not required at some time instants; and
- (ii) ensure that the services are assigned CPU resources with bounded service delay so that they can actively perform computations within constrained deadlines so that they can properly utilize at best CPU resources to ensure fruitful execution progress.

In NANCY, resource reservation is applied to groups of containerized processes/threads thanks to a kernel patch. In this way, a group of processes/threads that belong to the same service can be inserted in a dedicated control group (cgroup) and assigned the same reservation capabilities (that are associated with the cgroup). This also allows implementing a hierarchical scheduling framework where further scheduling logic (e.g., based on priorities) is implemented to regulate access to CPU resources when a reservation is actively served.

The mentioned patch allows associating each control group with a dedicated runtime (time budget) “Q”, a period “P”, and a number of CPU cores “m”. This results in creating “m” fixed-priority queue for the threads/processes contained in the cgroup, with each queue being associated with a scheduling entity that is reserved “Q” units of execution time every period “P”. The SCHED_DEADLINE scheduling policy selects one of such scheduling entities, and the standard fixed priority scheduler is used to schedule the highest-priority thread/process from the queue associated with the scheduling entity selected by SCHED_DEADLINE. The resulting scheduling hierarchy is compliant with the Compositional Scheduling Framework (CSF) [45] and hence allows providing performance guarantees to the real-time tasks running in the cgroup/container.

If the application’s threads/processes run in a hypervisor-based VM instead of a container, SCHED_DEADLINE can be used to schedule the VM’s virtual CPU cores without requiring any patch to the Linux kernel. In this case, the virtual CPU cores are implemented by the VM as threads that can be directly scheduled by the Linux kernel’s CPU scheduler. Hence, the SCHED_DEADLINE policy can be directly used for every virtual CPU thread to implement the CSF.

As an example, an application composed of 3 periodic real-time threads has been scheduled in a Kubernetes container and in a KVM-based VM, measuring the threads’ response times. The experiments have been run on an Intel(R) Xeon(R) CPU E5-2650 v2 with 16 cores at 2.60GHz. The first

thread executes for 8ms every 60ms; the second thread executes for 13ms every 80ms and the third thread executes for 22ms every 90ms; every thread is periodically activated and should finish its execution before the next periodic activation. We collected at least 1000 samples, each corresponding to a periodic activation, of each task. The response time (defined as the difference between the instant when the periodic execution finishes and the activation time) is measured (and should be smaller than the thread's period to avoid compromising real-time guarantees). If the three threads execute on a virtual CPU scheduled with runtime Q and period P, the CSF theory allows dimensioning Q and P so that all the activations of all the threads are served before the end of the period. In particular, for the three threads mentioned above, a virtual CPU runtime Q=7.5ms and a period P=11ms allow for respecting all the temporal constraints.

Figure 16 displays the experimental Cumulative Distribution Function (CDF) of the normalized response times (response times divided by thread period) when the threads execute in a container with one virtual CPU scheduled by the standard Linux scheduler or by SCHED_DEADLINE (using the mentioned kernel patch) with (Q,P)=(7.5ms,11ms). From the figure, it is possible to notice that with the standard Linux scheduler, there is a non-zero probability of seeing a normalized response time larger than 1 (indicating that the execution of an activation finishes after the next activation), while SCHED_DEADLINE allows ensuring that all the normalized response times are smaller than 1.

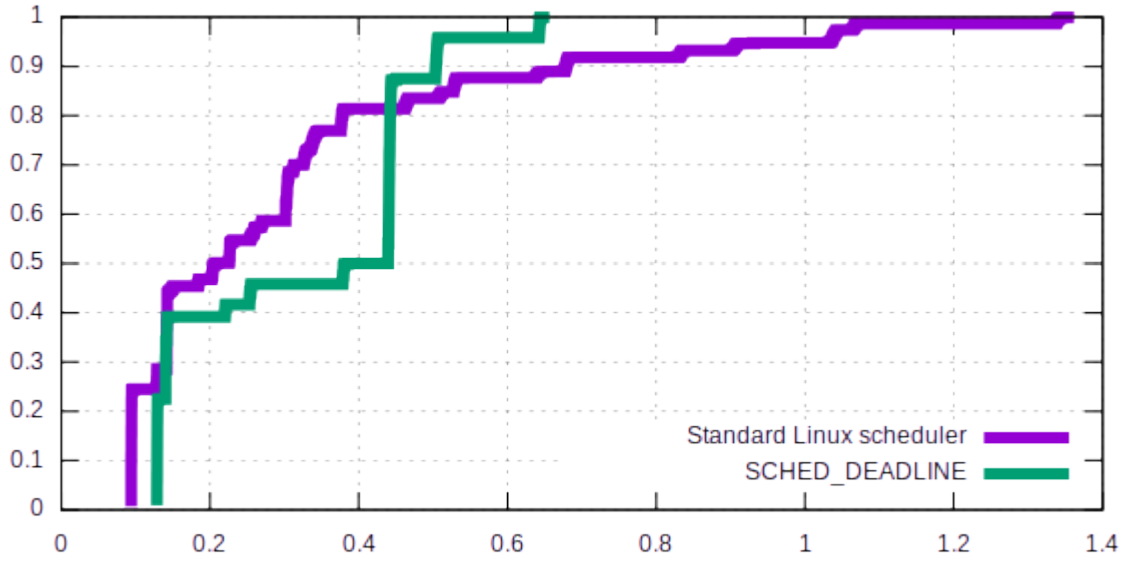

Figure 16: CDF of the normalized response times when the threads execute in a container with one virtual CPU scheduled by the standard Linux scheduler or by SCHED_DEADLINE with (Q,P)=(7.5ms,11ms).

Figure 17 plots the results of the same experiment repeated using a KVM-based VM instead of a Kubernetes container (and hence using the standard SCHED_DEADLINE policy instead of the patched kernel).

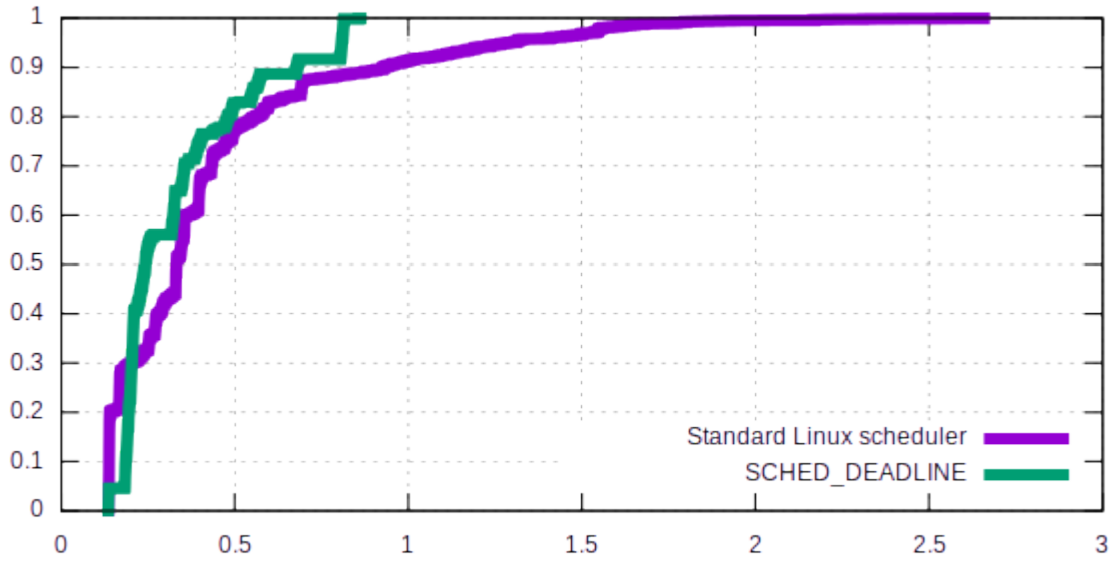

Figure 17: CDF of the normalized response times when the threads execute in VM with one virtual CPU scheduled by the standard Linux scheduler or by SCHED_DEADLINE with (Q,P)=(7.5ms,11ms)

Again, the results of the experiment show that SCHED_DEADLINE allows respecting the thread's temporal constraints; the standard Linux scheduler does not. These two figures also show that for simple workloads and one single virtual CPU core, the container-based approach and hypervisor-based VMs provide similar results. When the workload is more complex and multiple virtual CPU cores are used, the container-based approach enables better exploitation of the CPU time, as it can better support global schedulers (while hypervisor-based VMs can provide predictable results only using partitioned schedulers) [46].

4. Support for the Usage Scenarios

Efficient resource allocation and operator selection can be improved by exploiting the prediction of throughput (Section 2.4), the assessment of link quality (Section 2.2), and spectrum usage (Section 2.3) to strategically enhance connectivity and service quality. Furthermore, integrating location-based insights (Section 2.1) improves operator selection, ensuring users connect via the most suitable operators based on their geographic context. Such a comprehensive approach maximizes network efficiency and significantly elevates user experience.

4.1. Fronthaul

4.1.1. Direct connectivity

5G and B5G networks are anticipated to resolve the future market demands and business models, by connecting society and meeting the requirements of various applications. Direct 5G communication is a promising technology expected to enable new applications and services by improving spectrum efficiency and the overall system throughput, energy efficiency, and by reducing network latency. In addition, it improves connectivity providing higher data rates and higher bandwidth with better QoS. AR/ VR applications can be employed in several segments, such as mobile streaming and video delivery, healthcare, tourism, industry 4.0 verticals, entertainment and PPDR use cases (see Figure 18 []).

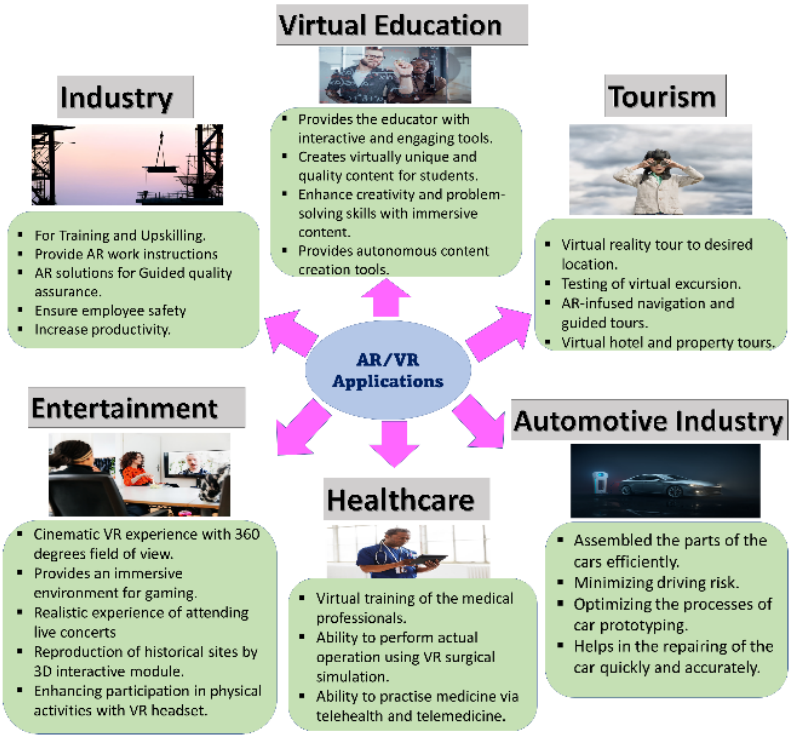

Figure 18: The major AR/VR use cases [47]

The combination of immersive and B5G technology is poised to significantly impact user experience in AR/VR applications, transforming various industries and creating a plethora of possibilities. NANCY platform is designed to provide extremely low latency ensuring a seamless AR/VR experience.

In the Fronthaul network of fixed topology usage scenario, and more specifically in the direct connectivity usage scenario, each UE performs a computation-intensive and delay-sensitive task, such as video streaming and AR/VR application. Considering that all base stations and access points belong

to the same network provider and are equipped with MEC capabilities; they have high availability of computation resources and can execute AI functions. Moreover, task offloading can take place (for example, a mobile device can offload its tasks inside the same dot).

In this scenario, high reliability, data rate achieved by the end-UEs, and latency will be measured as key network parameters and NANCY KPIs. Perceived user experience and privacy will also be considered. Therefore, the use of the AI-based orchestrator is required to jointly optimize the resource and NF allocation. The ability to offer high-quality content seamlessly, even during peak congestion with faster speed and almost zero latency enhances overall user experience and creates new innovative business models for content providers and telecom operators.

4.1.2. Coordinated multi-point connectivity

In the CoMP connectivity scenario, a UE is within the range of at least two BS/APs and the interactions between UE and BSs/APs are not trust-based (D2.1 'NANCY Requirements Analysis'). Two different CoMP usage scenarios are considered:

1. Usage scenario 1.1, where both BSs/APs belong to the same operator.
2. Usage scenario 1.2, where BS1/AP1 belongs to operator 1 (OP1) and BS2/AP2 belongs to OP2.

Usage scenario 1.2 is of special interest to NANCY, requiring establishing a deal between the two operators that can be organized into B-RAN to form a sizable and ubiquitous wireless network, which can significantly improve the utilization of spectral resources and infrastructures. The "*Spectrum Activity Characterization*" service described in Section 2.3 could play a crucial role in this scenario. The variant network traffic loads could cause a strong imbalance in spectrum demands; by leveraging the "*Spectrum Activity Characterization*" MEC service it is possible to continuously monitor and analyze spectrum usage, identify patterns and meet the dynamic spectrum requests coming from different users, considering spatial and temporal variance. This service could also help in selecting the operator that best fits the resource demand coming from the user, and it could aid in delivering high-quality service and obtaining higher spectrum efficiency, optimizing the efficient utilization of the available radio spectrum, achieving fine-grained spectrum resource management and, as a secondary effect, energy efficiency.

Moreover, in conjunction with the *'Throughput Prediction'* service, described in Section 2.4, it is possible to further enhance the quality of experience perceived by the user while, at the same time, ensuring service continuity and utilizing the resource infrastructure more efficiently.

4.2. Advanced Coverage Expansion

In the coverage expansion scenario, the intermediate node can act both as a consumer and operator, by deploying a private 5G network. This deployment can significantly improve the coverage and quality of communications of a network.

The physical position of a device has a direct impact on the signal quality. As the distance between the BS and the device is increased, the signal quality is decreased due to the propagation loss of the signal. The signal degradation is also affected by fading due to multipath propagation. The *'Location-based'* MEC service can retrieve the location of the intermediate node and the end-user devices and predict the respective paths. As a result, the RAN can be optimally orchestrated in order to ensure high signal quality and reduce handover latency.

By utilizing the *'Link Anomaly/Quality'* service, described in Section 2.2 the intermediate operator can directly manage and mitigate common link anomalies, such as signal attenuation, interference, and

bandwidth saturation that often degrade network performance in traditional setups. This is particularly beneficial in scenarios where extending coverage is critical, such as in remote or densely built environments where traditional networks struggle. The operator's dual role not only extends the physical reach of the network but also ensures superior link stability and quality, which is critical for supporting high-demand applications and services.

The characterization of the spectrum activity is critical for optimizing the efficient utilization of the radio spectrum, especially in densely populated and interference-limited environments. By leveraging the *'Spectrum Activity Characterization'* service, the intermediate node can continuously monitor and analyze spectrum usage, identifying patterns of activity and potential points of interference. This data enables dynamic allocation and reallocation of spectrum resources, improving overall network efficiency and performance. This approach enables the operator to ensure that expanded coverage not only reaches further but also operates more reliably and efficiently, adapting in real-time to changing demands and conditions in the network environment. This proactive management of spectrum activity is key to maintaining high-quality communications links and supporting a wide range of applications and services without degradation or interruption.

Finally, the *'Throughput Prediction'* service can enhance the quality of service by utilizing the infrastructure more efficiently. Infrastructure telemetry data can be leveraged by AI-based decision-making algorithms that can flag network flows, indicate service placement strategies, or dictate routes. Using the telemetry data of RAN nodes, network fabric, and system resource consumption, the total incoming throughput to the RAN nodes (i.e., the gNBs) of both providers can be estimated. Each provider can adjust the number of nodes necessary to offer optimal coverage with the least energy consumption. Moreover, the main provider can perform intelligent traffic steering operations taking into account the traffic volume that will originate from the intermediate operator. This information can prevent congestion and offer better total infrastructure utilization on the main provider's side.

4.3. Advanced Connectivity of Mobile Nodes

The Internet of Things in Motion (IoTM) has enabled a highly interconnected ecosystem with multiple involved dynamic elements, allowing connectivity through mobile devices, e-health gadgets or connected vehicles, among many others [47]. This scenario has increasingly promoted the use of self-driving and public vehicles that are interconnected, due to their advantages in terms of city mobility, avoiding traffic jams and becoming an eco-friendly alternative [48]. These vehicles are being integrated into smart cities and Cooperative-Intelligent Transportation Systems (C-ITS) thanks to the connectivity capabilities of On-Board Units (OBUs). OBUs make use of Multi-Radio Access Communication (MRAT) technologies to access cloud-based services such as traffic information, vehicle tracking, route planning, etc., using radio technologies such as vehicular Wi-Fi, cellular networks or LPWANs [49].

Due to the different profiles in terms of energy efficiency and limited power consumption that vehicles present (cars, motorbikes, scooters, bicycles, etc.), as well as the specific quality of service (QoS) requirements, e.g., latency, throughput or reliability, of the wide range of vehicular services, Decision Support Systems (DSSs) in OBUs are needed to select the most optimal communication technology for each given message [50]. DSS uses machine learning (ML) techniques such as Neural Networks and Random Forests to make optimal decisions [51]. In addition, the TinyML paradigm adapts ML algorithms to limited platforms such as microcontroller units (MCUs), enabling innovation on limited-resources devices [52], [53]. The combination of Multi-Radio Access Technology (MRAT) with Decision Support Systems (DSS) offers a clear advantage in terms of QoS and energy efficiency, optimizing power consumption by selecting the most adequate technology to transmit data according to the system's requirements. Low power consumption allows MRATs to be attached to personal mobility

vehicles, such as electric scooters or segways, avoiding the need for heavy and voluminous power supplies.

Vehicles equipped with MRAT OBUs need to connect to different base stations/access points of different operators without experiencing service interruptions. This requires collaboration between operators to enable seamless handover of connections and ensure continuity of service as vehicles move across different coverage areas. In addition, optimization of resources is essential to avoid network congestion and ensure QoS. With multiple operators involved, it is important to ensure the security and privacy of transmitted data, so the integration of blockchain systems in vehicular networks can guarantee authentication and transactions between operators, as well as preserve data integrity and confidentiality.

The novel solution presented in NANCY is the Multi-Radio Access Technology - Nomadic Connectivity Providers (MRAT-NCPs), a novel concept which serves as a connectivity amplifier to improve network coverage and capacity. Under this paradigm, a MRAT-NCP element provides connectivity towards a given 5G infrastructure to remote end-devices that are out-of-range of such a 5G network. To do so, different radio technologies to connect the MRAT-NCP element and the end-devices can be employed, e.g., WiFi, 5G's side-channel (PC5), etc. Besides, the MRAT-NCP may connect to different network operators even simultaneously if it hosts a sufficient number of radio interfaces. In this line, since UEs can transmit data through multiple MRAT-NCPs that may be connected to different 5G networks, network resource utilization is optimized, allowing more efficient redistribution of traffic to avoid congestion, and improving the overall performance of the network.

Besides these benefits, the use of MRAT-NCPs may also contribute to significant cost reduction, as it enables more efficient resource use and precise network management, as well as facilitating the migration of offloading functions. This results in energy, infrastructure and maintenance savings. In addition, improvements in handover prediction and management, as well as the ability to dynamically allocate resources, ensure stricter compliance with Service Level Agreement (SLA) requirements, ensuring that services are maintained within the parameters agreed with customers. This, combined with inter-operator cooperation and the mobility capability of MRAT-NCPs, ensures uninterrupted connectivity for users and improves the provided QoS.

As mentioned above, one of the key innovations to enable the NANCY's MRAT-NCP paradigm is the ML-based selection of the most adequate radio interface to establish reliable and dynamic radio links. While traditional RAT selection is reactive, based on observations of decaying link parameters such as signal strength, bit error rates, and throughput, AI-enabled selection can be proactive by relying on predicted values for link parameters. In this scenario, predicted anomalies of values for the quality of the link based on the service discussed in Section 2.2, estimated noise due to increased spectrum activity as discussed in Section 2.3, as well as throughput estimated from the service discussed in Section 2.4. Furthermore, for scenarios where the equipment supports beamforming, the localization service described in Section 2.1 is able to inform the controller of the beamforming on the location of the user. Alternatively, the predicted location could also be used for proactive handover or offloading.

5. Conclusion

This deliverable presents the development of network functionalities within the NANCY architecture, exploiting MEC and O-RAN technologies. The presented integration of MEC and O-RAN technologies shows the enhanced performance on various network functionalities. These technologies support low-latency, high-throughput, and reliable wireless communications crucial for advanced functions such as autonomous driving, IoT, and AR/VR.

The development of several AI-based network functionalities has been presented. Among them are location-based services that predict user position to optimize network operations such as beamforming and handover, monitoring link quality and detecting anomalies to enhance network reliability, spectrum activity characterization to manage resources efficiently, and throughput prediction to optimize network performance and resource allocation. A self-evolving model repository has been developed and integrated with the AI virtualizer to optimize network operations dynamically. This allows network performance to be improved through continuous learning and adaptation of AI models.

Strategies for efficiently utilizing virtualized resources, including advanced scheduling techniques and lightweight containers, have been outlined. These strategies permit minimizing resource underutilization, assuring optimal performance and cost efficiency. The resulting functionalities support several usage scenarios, including advanced coverage expansion, coordinated multi-point connectivity, and enhanced connectivity for mobile nodes. They have been validated, exhibiting their feasibility and efficacy in real-world scenarios. Implementation details, including code snippets and experimental results, are provided to facilitate further development and integration.

Bibliography

[1] S. D'Oro, L. Bonati, M. Polese and T. Melodia, "OrchestRAN: Orchestrating network intelligence in the open RAN," IEEE Transactions on Mobile Computing, vol. 23, no. 7, pp. 7952-7968, 2023.

[2] A. Pirnat, B. Bertalanič, G. Cerar, M. Mohorčič, M. Meža and C. Fortuna, "Towards Sustainable Deep Learning for Wireless Fingerprinting Localization," in ICC 2022 - IEEE International Conference on Communications, 2022.

[3] J. Lorca, M. Hunukumbure and Y. Wang, "On Overcoming the Impact of Doppler Spectrum in Millimeter-Wave V2I Communications," in IEEE Globecom Workshops (GC Wkshps), 2017, pp. 1-6.

[4] A. Varshavsky, E. de Lara, J. Hightower, A. LaMarca and V. Otsason, "GSM indoor localization," Pervasive and Mobile Computing, vol. 3, no. 6, pp. 698-720, 2007.

[5] A. Narayanan, E. Ramadan, R. Mehta, X. Hu, Q. Liu, R. A. K. Fezeu, U. K. Dayalan, S. Verma, P. Ji, T. Li, F. Qian and Z.-L. Zhang, "Lumos5G: Mapping and Predicting Commercial mmWave 5G Throughput," ACM Internet Measurement Conference, 2020.

[6] M. Arnold, J. Hoydis and S. t. Brink, "Novel Massive MIMO Channel Sounding Data Applied to Deep Learning-based Indoor Positioning," 12th International ITG Conference on Systems, Communications and Coding (SCC), 2019.

[7] M. Gauger, A. Maximilian and S. ten Brink, "Massive MIMO Channel Measurements and Achievable Rates in a Residential Area," 24th International ITG Workshop on Smart Antennas (WSA), 2020.

[8] Ettus Research, "USRP User Manual," [Online]. Available: http://www.ettus.com/.

[9] B. Bertalanič, G. Morano and G. Cerar, "LOG-a-TEC Testbed outdoor localization using BLE beacons," International Balkan Conference on Communications and Networking (BalkanCom), 2022.

[10] G. Cerar, A. Švigelj, M. Mohorčič, C. Fortuna and T. Javornik, "Improving CSI-based Massive MIMO Indoor Positioning using Convolutional Neural Network," Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), 2021.

[11] M. Arnold, S. Dorner, S. Cammerer and S. Ten Brink, "On Deep Learning-Based Massive MIMO Indoor User Localization," IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), 2018.

[12] B. Bertalanič, J. Hribar and C. Fortuna, "Visibility Graph-Based Wireless Anomaly Detection for Digital Twin Edge Networks," IEEE Open Journal of the Communications Society, vol. 5, pp. 3050-3065, 2024.

[13] G. Cerar, H. Yetgin, B. Bertalanic and C. Fortuna, "Learning to Detect Anomalous Wireless Links in IoT Networks," IEEE Access, vol. 5, pp. 212130-212155, 2020.

[14] T. Šolc, C. Fortuna and M. Mohorčič, "Low-cost testbed development and its applications in cognitive radio prototyping," in Cognitive radio and networking for heterogeneous wireless networks, Springer, 2015, pp. 361-405.

[15] T. Gale, T. Šolc, R.-A. Moşoi, M. Mohorčič and C. Fortuna, "Automatic detection of wireless transmissions," IEEE Access, vol. 8, pp. 24370-24384, 2020.

[16] M. Caron, P. Bojanowski, A. Joulin and M. Douze, "Deep clustering for unsupervised learning of visual features," Proceedings of the European Conference on Computer Vision (ECCV), 2018.

[17] T. J. O'Shea, T. Roy and T. C. Clancy, "Over-the-air deep learning based radio signal classification," IEEE Journal of Selected Topics in Signal Processing, vol. 12, no. 1, pp. 168-179, 2018.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga and others, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," Advances in Neural Information Processing Systems, 2019.

[19] M. Agiwal, A. Roy and N. Saxena, "Next Generation 5G Wireless Networks: A Comprehensive Survey," IEEE Communications Surveys & Tutorials, vol. 18, no. 3, pp. 1617-1655, 2016.

[20] M. Alsabah, M. A. Naser, M. Mahmmod, S. Abdulhussain, M. R. Eissa, A. Al-Baidhani, N. K. Noordin, S. M. Sait, K. A. Al-Utaibi and F. Hashim, "6G Wireless Communications Networks: A Comprehensive Survey," IEEE Access, vol. 9, pp. 148191-148243, 2021.

[21] K. Kousias, M. Rajiullah, G. Caso, O. Alay, A. Brunstrom, U. Ali, L. De Nardis, M. Neri and M. G. Di Benedetto, "Empirical Performance Analysis and ML-based Modeling of 5G non-standalone networks," Elsevier Computer Networks, vol. 241, pp. 1-17, 2024.

[22] L. Xie, S. Song and K. B. Letaief, "Networked Sensing With AI-Empowered Interference Management: Exploiting Macro-Diversity and Array Gain in Perceptive Mobile Networks," IEEE Journal on Selected Areas in Communications, vol. 41, no. 12, pp. 3863-3877, 2023.

[23] M. Polese, L. Bonati, S. D'Oro, S. Basangi and T. Melodia, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," IEEE Communications Surveys & Tutorials, vol. 25, no. 2, pp. 1376-1411, 2023.

[24] F. Schaich, T. Wild and R. Ahmed, "Subcarrier Spacing - How to Make Use of This Degree of Freedom," IEEE 83rd Vehicular Technology Conference (VTC Spring), 2016.

[25] S. Oulaourf, E. Haidine and H. Ouahmane, "Review on using game theory in resource allocation for LTE/LTE-Advanced," International Conference on Advanced Communication Systems and Information Security (ACOSIS), 2016.

[26] V. N. Ha, L. B. Le and N. D. Dao, "Cooperative transmission in cloud RAN considering fronthaul capacity and cloud processing constraints," IEEE Wireless Communications and Networking Conference (WCNC), 2014.

[27] Z. Wang, H. Li, H. Wang and S. Ci, "Probability weighted based spectral resources allocation algorithm in Hetnet under Cloud-RAN architecture," IEEE/CIC International Conference on Communications in China - Workshops (CIC/ICCC), Xi'an, China, 2013.

[28] R. Keerthika and S. Abinayaa, Algorithms of Intelligence: Exploring the World of Machine Learning, Inkbound Publishers, 2022.

[29] S. Sun, Z. Cao, H. Zhu and J. Zhao, "A Survey of Optimization Methods from a Machine Learning Perspective," IEEE Transactions on Cybernetics, vol. 50, no. 8, pp. 3668-3681, 2020.

[30] H. M. F. Noman, E. Hanafi, K. Noordin, K. Dimyati, M. N. Hindia, A. Abdrabou and F. Qamar, "Machine Learning Empowered Emerging Wireless Networks in 6G: Recent Advancements, Challenges and Future Trends," IEEE Access, vol. 11, pp. 83017-83051, 2023.

[31] Y. Sun, J. Liu, J. Wang, Y. Cao and N. Kato, "When Machine Learning Meets Privacy in 6G: A Survey," IEEE Communications Surveys & Tutorials, vol. 22, no. 4, pp. 2694-2724, 2020.

[32] Y. Zuo, J. Guo, N. Gao, Y. Zhu, S. Jin and X. Li, "A Survey of Blockchain and Artificial Intelligence for 6G Wireless Communications," IEEE Communications Surveys & Tutorials, vol. 25, no. 4, pp. 2494-2528, 2023.

[33] D. Raca, A. H. Zahran, C. J. Sreenan, R. K. Sinha, E. Halepovic, R. Jana and .. V. Gopalakrishnan, "On Leveraging Machine and Deep Learning for Throughput Prediction in Cellular Networks: Design, Performance, and Challenges," IEEE Communications Magazine, vol. 58, no. 3, pp. 11-17, 2020.

[34] J. Kaur, M. A. Khan, M. Iftikhar, M. Imran and Q. Emad Ul Haq, "Machine Learning Techniques for 5G and Beyond," IEEE Access, vol. 9, pp. 23472-23488, 2021.

[35] B. Adhikari, M. Jaseemuddin and A. Anpalagan, "esource Allocation for Co-Existence of eMBB and URLLC Services in 6G Wireless Networks: A Survey," IEEE Access, vol. 12, pp. 552-581, 2024.

[36] X. Yang, Z. Zho and B. Huang, "URLLC Key Technologies and Standardization for 6G Power Internet of Things," IEEE Communications Standards Magazine, vol. 12, pp. 552-581, 2021.

[37] O. Elgarhy, M. Reggiani, A. M. M, A. Zoha, R. Ahmad and A. Kuusik, "Energy Efficiency and Latency Optimization for IoT URLLC and mMTC Use Cases," IEEE Access, vol. 12, pp. 23132-23148, 2024.

[38] A. Narayanan, E. Ramadan, R. Mehta, X. Hu, Q. Liu, R. A. K. Fezeu, U. A. Dayalan, S. Verma, P. Ji, T. Li, F. Qian and Z. L. Zhang, "Lumos5G: Mapping and Predicting Commercial mmWave 5G Throughput," IEEE DataPort, 2022.

[39] C. Oliveira, "Interpolation," in Practical C++20 Financial Programming , Springer, 2021 , pp. 235-250.

[40] S. W. Smith, "Moving Average Filters," in The Scientist and Engineer's Guide to Digital Signal Processing, San Diego, CA, California Technical Publishing, 1997, pp. 277-284.

[41] I. Sutskever and O. Vinyals, "Sequence to Sequence Learning with Neural Networks," in Advances in Neural Information Processing Systems, 2021.

[42] M. T. Luong, H. Pham and M. C.D, "ffective Approaches to Attention-based Neural Machine Translation," Conference on Empirical Methods in Natural Language Processing, 2015.

[43] J. Lelli, C. Scordino, L. Abeni and D. Faggioli, "Deadline scheduling in the Linux kernel," Software: Practice and Experience, vol. 46, no. 6, p. 18, 2016.

[44] S. Fiori, L. Abeni and T. Cucinotta, "RT-Kubernetes, Containerized Real-Time Cloud Computing," in ACM SAC, 2022.

[45] S. Insik and L. Insup, "Compositional real-time scheduling framework," in 25th IEEE International Real-Time Systems Symposium, 2004.

[46] L. Abeni, A. Biondi and E. Bini, "Hierarchical scheduling of real-time tasks over Linux-based virtual machines," Journal of Systems and Software, vol. 149, p. 15, 2019.

[47] A. Hazarika and M. Rahmati, "Towards an Evolved Immersive Experience: Exploring 5G- and Beyond-Enabled Ultra-Low-Latency Communications for Augmented and Virtual Reality," Sensors, vol. 23, no. 7, p. 3682, 2023.

[48] R. Costa, "The Internet of Moving Things [Industry View]," IEEE Technology and Society Magazine, vol. 37, no. 1, pp. 13-14, March 2018.

[49] R. Sanchez-Iborra, L. Bernal-Escobedo and J. Santa, "Machine learning-based radio access technology selection in the Internet of moving things," China Communications, vol. 18, no. 7, pp. 13-24, July 2021.

[50] G. Naik, B. Choudhury and J. Park, "IEEE 802.11bd & 5G NR V2X: Evolution of Radio Access Technologies for V2X Communications," IEEE Access, vol. 7, pp. 70169-70184, 2019.

[51] K. Mikhaylov and a. et., "Energy Efficiency of Multi-Radio Massive Machine-Type Communication (MR-MMTC): Applications, Challenges, and Solutions," IEEE Communications Magazine, vol. 57, no. 6, pp. 100-106, June 2019.

[52] J. Merkert, M. Müller and M. Hubl, "A Survey of the Application of Machine Learning in Decision Support," ECIS 2015 Completed Research Papers, 2015.

[53] P. Warden and D. Situnayake, Tinyml: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers, O'Reilly UK Ltd., 2019.

[54] R. Sanchez-Iborra and . A. Skarmeta, "TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities," IEEE Circuits and Systems Magazine, vol. 20, no. 3, pp. 4-18, 2020.

[55] S. D'Oro, L. Bonati, M. Polese and T. Melodia, "OrchestRAN: Orchestrating Network Intelligence in the Open RAN," IEEE Transactions on Mobile Computing, vol. 23, no. 7, pp. 7952-7968, 2024.

[56] S. D'Oro, B. Leonardi and T. Melodia, "OrchestRAN: Orchestrating Network Intelligence in," IEEE Transactions on Mobile Computing, vol. 23, no. 7, pp. 7952-7968, July 2024.