INTERNATIONAL TELECOMMUNICATION UNION

TELECOMMUNICATION STANDARDIZATION SECTOR

STUDY PERIOD 2022-2024

FOCUS GROUP ON AI NATIVE NETWORKS

AINN-I-xx

Original: English

Question(s): N/A 28th, September 2024

INPUT DOCUMENT

Source: *BLAZE-IITJ (IIT JODHPUR)*

Title: *A Real Time CQI Prediction Framework for Proactive Resource Scheduling in 5G Enabled Drones Using AI*

Contact:
1) Ankush Chaudhary **E-mail:** m24eei021@iitj.ac.in
2) Aswathy Puthukkulam aswathyp@iitj.ac.in

Submitting Entity:

1. Wireless Communications and Navigation Lab,
   Dept. of Electrical Engineering, Indian Institute of Technology Jodhpur.

Team (BLAZE-IITJ):

1. Ankush Chaudhary, M. Tech 1st Year, Intelligent Communication Systems, Dept. of EE.
2. Aswathy Puthukkulam, Project Associate, Wireless Communications and Navigation Lab, Dept. of EE.

Faculty Mentors:

1. Dr. Sai Kiran M. P. R., Assistant Professor, Dept. of Electrical Engineering
2. Dr. Arun Kumar Singh, Associate Professor, Dept. of Electrical Engineering

1. Abstract:

With the emergence of 5G, new use-cases are being envisioned, including: Ultra-Reliable Low-Latency Communication (uRLLC), Massive Machine-Type Communications (mMTC), Enhanced Mobile Broadband (eMBB). One of the key technologies that will enable the efficient support of these use cases is **network slicing** using **AI technologies**. In relevance to the above discussion, in this hackathon, we propose:

"A Real-Time CQI Prediction Framework for Proactive Resource Scheduling in 5G-Enabled Drones Using AI"

Our goal is to demonstrate the feasibility of dynamic resource scheduling in a 5G O-RAN compliant network to support uRLLC use cases such as 5G-enabled drones using AI for estimating Channel Quality Indicator (CQI). The key activities undertaken and contributions include:

1. **Setting Up a 5G O-RAN Compliant Network**: Utilizing Open Air Interface (OAI) as a sandbox for technology demonstration (both UE and gNB, and we consider the UE as a 5G-enabled drone).
2. **Integration of FlexRIC**: Connecting the RAN Intelligent Controller with the OAI 5G sandbox using the E2 agent.
3. **Establishing the OAI 5G Core Network**: Deployment of OAI 5G containers using Docker containers.
4. **Development of xApp**: Creating a xApp in Python 3 to aggregate and prepare a CQI dataset for AI model training and eventual prediction of the CQI by integrating the trained model.
5. **Utility Tools Development**: Using Matlab and Linux-based Expect scripts to induce automated channel variations into the OAI RF Simulator for CQI dataset collection and model validation.
6. **Creation of a CQI Dataset**: Creation of CQI data using the utility tools developed specifically for AI model training.

7. **AI Model Development and Training**: Developed a Bi-LSTM based model that takes the past CQI values (for the past 400 frames) and predicts the future CQI value for the upcoming frame for a User Equipment (UE).
8. **Real-Time Validation**: Validating the proposed model in integration with the xApp, FlexRIC, OAI gNB, and OAI UE to assess performance.

From our demonstration, we observed that the proposed model achieves:

- **Prediction MAE** (Mean Absolute Error): < 0.5 CQI Units
- **Prediction MSE** (Mean Squared Error): < 2 CQI^2

Future work will focus on the development and demonstration of dynamic resource allocation strategies using the predicted CQI values. We sincerely believe that the demonstrated use-case will have significant potential on the evolving use-cases of 5G.

2. Use-case Introduction:

Unmanned Aerial Vehicles (UAVs), especially drones, have immense applications, including agriculture, payload deliveries, surveillance, search and rescue, coverage extension in telecom networks [1-4]. However, currently, the communication technology which is prominently used for control of drones in ZigBee or WiFi based ad-hoc networking technologies. One of the major limitations of these technologies is the range limitation. Hence, in many cases, the drones when flying beyond line-of-sight (LoS) typically utilize path planning strategies and autonomous navigation. However, many applications such as search-and-rescue, surveillance, etc., cannot rely on path planning as the navigation path may not be known ahead of the flight. In such cases, the real-time control of drones beyond LoS will significantly benefit the applications. Hence, utilizing 5G telecom networks for such applications will be a suitable solution, as the 5G telecom networks in India currently offer wide coverage with multi-cell deployment.

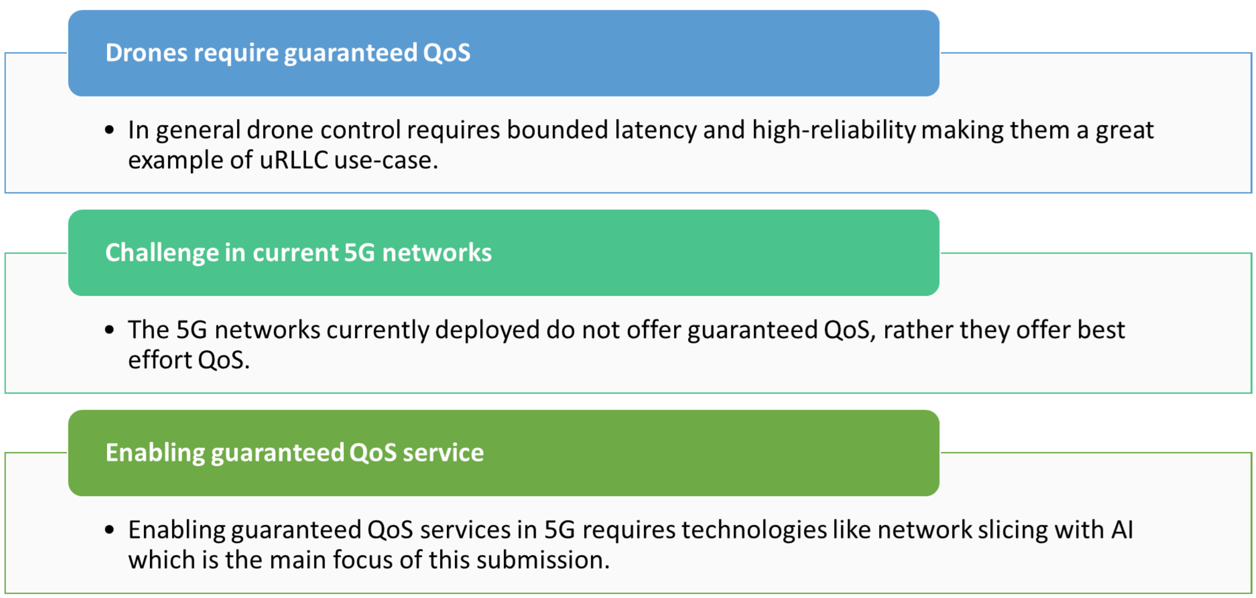

Figure 1: Challenges in enabling control of drones over 5G networks

Nevertheless, there are challenges to enable the control of drones over 5G networks which are summarized in Fig. 1 [2]. Importantly, the challenges include the following:

1) Drones require guaranteed QoS:
    a) Controlling drones require control information in the downlink to be transmitted at a bounded latency such as <30 ms (3GPP, TR 36.777) and with high reliability of

>99% at least. If the service level agreements are not met, then the application will result in catastrophic consequences.

2) Challenge in current 5G and ad-hoc networks:
    a) The current 5G networks are deployed to offer best-effort QoS. Hence, there is no outage present due to the non-consideration of key performance guarantees. This leads to varying latency (in fact can cross more than 100 ms) and reliability (may be less than 99%) leading to their non-suitability to control drones. Whereas, the ad-hoc networks (such as ZigBee, WiFi, etc.) offer limited range limiting the applications of drones or UAVs.
3) Enabling guaranteed QoS:
    a) Enabling guaranteed QoS in 5G networks requires technologies such as network slicing and AI to be integrated with the network. Currently with the evolution of O-RAN architecture, the integration of AI to the 5G and Beyond networks is plausible and provides greater flexibility.

In this hackathon, our objective is to address some of the above challenges and our important contributions include the following:

1) We propose “A Real Time CQI Prediction Framework for Proactive Resource Scheduling in 5G Enabled Drones Using AI.” We aim to demonstrate the feasibility of dynamic resource scheduling in a 5G O-RAN compliant network for supporting uRLLC use-cases such as 5G-enabled drones.
2) Setting up of a 5G O-RAN compliant network using Open Air Interface (OAI) as a sandbox for proposed technology demonstration [5].
3) Integration of FlexRIC (RAN Intelligent Controller) with the OAI 5G sandbox using E2 agent. Setting up of OAI 5G Core Network [5].
4) Development of xApp in Python-3 for aggregating and preparing CQI dataset for AI model training.
5) Development of utility tools using Matlab and Linux-based Expect scripts for inducing automated channel variations into OAI RF Simulator for database collection and model validation subsequently.
6) Creation of CQI dataset for AI model training purpose. Development and training of Bi-LSTM based AI model that takes the past CQI values (for the past 400 frames in this case) and predicts the future CQI value (for the upcoming frame) for a UE.

7) Real-time validation of the proposed model in integration with the xApp, FlexRIC, OAI gNB, OAI UE for understanding the performance.
8) From the demonstration, it is observed that the proposed model achieves a prediction MAE (mean absolute error) of <0.5 CQI Units and prediction MSE (mean square error) of <2 CQI^2 thus proving the feasibility of the CQI prediction and eventual dynamic resource allocation strategies.

2.1 Use-case Flow Diagram:

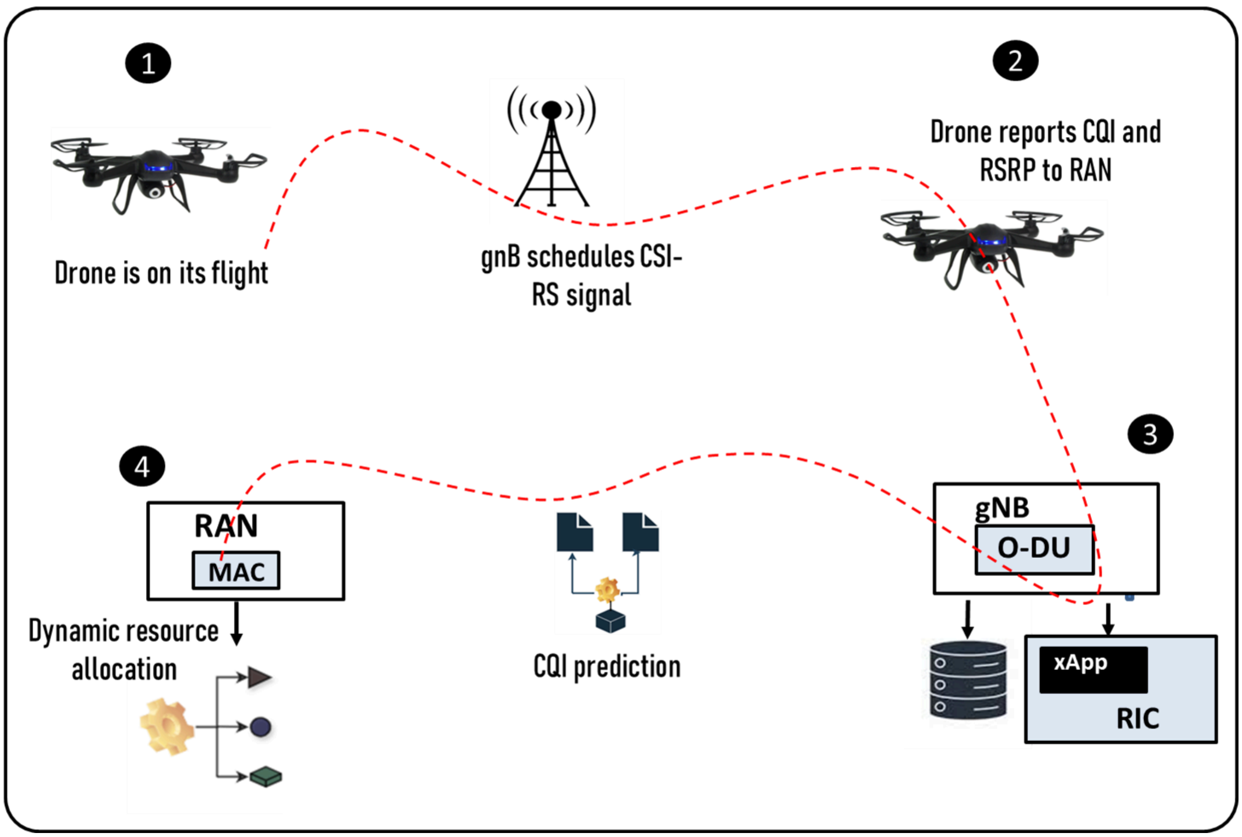

Figure 2: Proposed use-case flow diagram for "A Real Time CQI Prediction Framework for Proactive Resource Scheduling in 5G Enabled Drones Using AI"

Fig. 2 shows the flow diagram for the proposed use-case consisting of a UE (mimicking the behavior of a drone), gNB, RIC and xApp. In the below, we describe the flow diagram in detail:

- Step 1: **The drone registers** with the network using non-access stratum signaling (NAS) with 5G CN.

- Step 2: **PDU session is created** based on the NSSAI (Network Slice Selection Assistance Information) and UE will be serviced in the slice it is configured for
- Step 3: **gNB schedules the CSI-RS** (Channel State Information Reference Signals) in the downlink (DL) as shown in Fig. 3.
- Step 4: **UE performs CSI and computes the CQI** which is sent as a feedback to the gNB.
- Step 5: **gNB communicates the CQI statistics to the FlexRIC** using E2 agent.
- Step 6: **FlexRIC communicates the CQI value to the xApp** using E42 agent.
- Step 7: **xApp stores the data** for training purposes in a SQLite3 database.
- Step 8: Also, xApp uses the new CQI value along with the past CQI values (in this case we are using a history of 400 CQI values corresponding to the previous 400 frames) to **predict the CQI value in the next frame** using the trained AI model**.**
- Step 9: xApp based on the predicted CQI value **determines the scheduling policy** and communicates with the gNB.
- Step 10: **gNB uses the information to allocate resources** including the PRBs and transmit power for the DL transmissions to meet the SLAs.

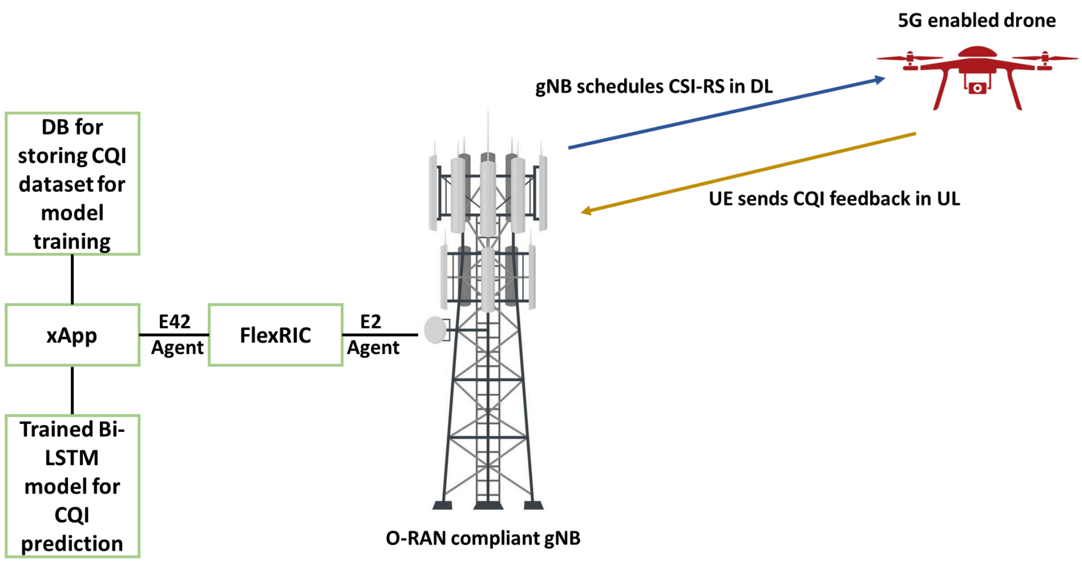

Figure 3: Explaining the CQI measurement using CSI-RS

CQI is an integer generated by UE to indicate the quality of a wireless channel. The UE continuously monitors channel parameters like signal to noise ratio (SNR), fading, etc., and reports CQI which is a quantized version of the CSI back to the gNB. The gNB dynamically adapts the modulation and coding (MCS) scheme based on the received CQI value. A poor channel condition can have a CQI value between 0 and 6. A channel with moderate channel condition will have CQI value between 7 to 10. A network with good channel condition will have a CQI value of 11 to 15 and can enhance transmission rate. A higher order modulation scheme like 64 QAM or 256 QAM will be used by a system with high CQI leading to higher data transmission with less allocated resources. However, if the CQI is less, more resources will have to be allocated to meet the service level agreements (SLAs). An example table of CQI versus MCS mapping and spectral efficiency is shown in Fig. 4

***Table 1**: CQI mapping table [3GPP 38.214, Release 18].*

CQI index	modulation	code rate x 1024	efficiency
0	out of range		
1	QPSK	78	0.1523
2	QPSK	120	0.2344
3	QPSK	193	0.3770
4	QPSK	308	0.6016
5	QPSK	449	0.8770
6	QPSK	602	1.1758
7	16QAM	378	1.4766
8	16QAM	490	1.9141
9	16QAM	616	2.4063
10	64QAM	466	2.7305
11	64QAM	567	3.3223
12	64QAM	666	3.9023
13	64QAM	772	4.5234
14	64QAM	873	5.1152
15	64QAM	948	5.5547

2.2 Use-case Requirements:

The following are the use-case requirements:

- **Requirement 1 (Critical):** Monitoring real-time UE CQI information to prepare training dataset for AI model development and eventual prediction of CQI for next frame in real-time using the trained model.

  Activity performed: We have set up an O-RAN compliant 5G sandbox using Open Air Interface comprising of UE (mimicking as a drone), gNB, core network, RIC (FlexRIC in this case), developed an xApp for data collection into SQLite3 DB. The developed xApp monitors the UE CQI per frame and logs the data into the DB. For inducing the channel variations we have developed a Matlab and Linux based Expect script utility for continuously monitoring channel parameters in OAI RF Simulator. Thus, a novel dataset for AI model training has been developed.

- **Requirement 2 (Critical):** Accurately predicting the CQI values considering the past CQI instances.

  Activity performed: As part of this, we have developed a Bi-LSTM (Bidirectional Long-Short-Term Memory) based model for predicting the CQI values accurately. The model takes an input of past CQI values (corresponding to present and previous 400 frames) and predicts the CQI value in the upcoming frame. The proposed model offers good accuracy while the validation (or) prediction MAE and MSE are <0.5 CQI units and <2 CQI^2 units, respectively. This shows the demonstration feasibility of the proposed idea.

- **Requirement 3 (Added value with good commercialization potential):** xApp will intimate RAN to allocate resources based on the proposed scheduling strategy using the predicted CQI value.

  Activity performed: Currently, the 5G networks utilize proportional-fair or round-robin based scheduling where the resources are allocated fairly across all the users. Hence, guaranteeing QoS using these schedulers is not a suitable option. We propose a prioritized scheduling mechanism, where the UEs falling under the guaranteed QoS requirement will be allocated with the resources (power, PRBs, etc.) first based on the CQI values predicted (and considering the MCS applicable). The remaining resources will be allocated to the UEs under the category best-effort QoS as available. Currently, the work is under progress at the Wireless Communications and Navigation Lab, IIT Jodhpur.

2.3 Mapping of the proposed Use-case to the SDGs:

Our use-case can be mapped to the following Sustainable Development Goals:

- **SDG 7.3** - *Improvement in energy efficiency* . Using the 5G technology, the current case study can design an energy efficient system, by increasing the transmit power when the channel is efficient and reducing the power when the channel seems to be in distress.
- **SDG 11.3** -Facilitate improvement in participatory engagement between human settlements using integrated communication approaches. Using real time data collection, resources can be managed wisely which can contribute to urban planning for the development of resilient and sustainable cities.
- **SDG 9.6** **-**This use case which focuses on directing resources towards fostering innovation, productivity and growth maps to the SDG goal 9 which emphasizes Industry*, Innovation and infrastructure*.

2.4 Mapping of the proposed Use-case to the ITU-T Use-cases:

As per ITU-T Technical Specification : *Use- cases for Autonomous Network*

Use case id : *FG-AN-Use case-032*

Use case name : AI enabled Game theory based mechanism for resource allocation

2.5 Secondary Use-cases:

The proposed work can also be extended to multiple use-cases such as:

- Providing guaranteed Quality of Experience services to applications such as Netflix, Hotstar, and other streaming platforms
- Enabling ROVs (remotely operated vehicles) such as cars, robots, aerial vehicles, etc., which are some of the focus areas of the prestigious Technology Innovation Hubs (TIHs) setup by Dept. of Science and Technology, Govt. of India.
- Enabling tele-surgeries as use-cases also require the guaranteed QoS over 5G networks which is the main objective of this work.

3. PS-1: Pipeline Design for the Use-case “A Real Time CQI Prediction Framework for Proactive Resource Scheduling in 5G Enabled Drones Using AI”

Note: For setting up the sandbox pipeline to replicate the use-case proposed in this document, please use the guide in the GitHub repository: [OAI Setup.md]

3.1 Sandbox Pipeline Designed for the Proposed Use-case:

The proposed sandbox pipeline used for the proposed use-case is provided in Fig. 4. For the purpose of demonstration and idea validation, we use Open Air Interface based 5G protocol stack which is compliant with O-RAN architecture.

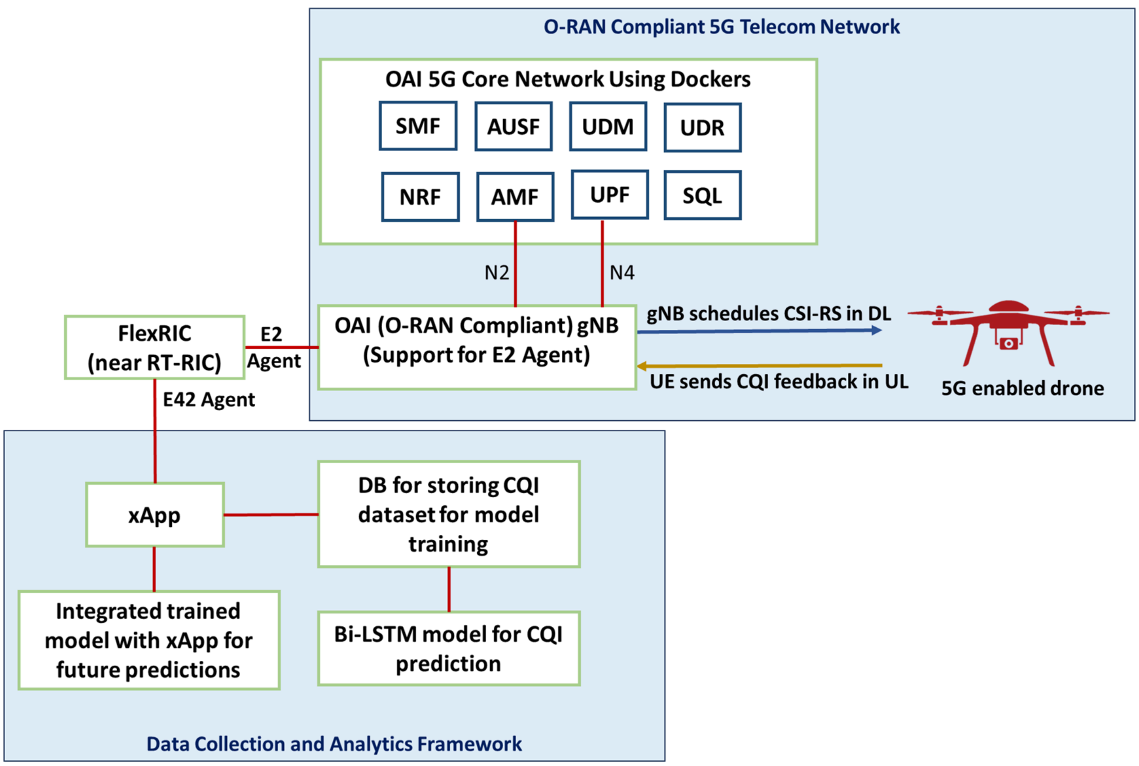

Figure 4: The OAI-based 5G sandbox pipeline used for the proposed use-case

For setting up of the above pipeline, the following steps can be followed:

- **Step 1: Deploying OAI 5G Core Network:**

  In the below, we discuss the deployment aspects of the OAI 5G Core Network. We consider the deployment of the following Core Network Functions:

  a. User Plane Function (UPF)
  b. Access and Mobility Management Function (AMF)
  c. Unified Data Repository (UDR)

d. Session Management Function (SMF)
e. Unified Data Management (UDM)
f. Authentication Server Function (AUSF)
g. Network Repository Function (NRF)
h. SQL (DB for storage)

The above network components enable all the functionalities related to user authentication, registration, and PDU session establishment in 5G. Also, we deploy the 5G Core Network Functions using Docker containers as shown below in Fig. 5.

Note: For setting up the OAI 5G CN discussed here, kindly refer to the setup guide: [OAI Setup.md]

```
kiran@Velocity:~/oai-cn5g$ sudo docker ps -a
[sudo] password for kiran:
CONTAINER ID   IMAGE                                         COMMAND                  CREATED      STATUS                  PORTS                                          NAMES
9c1c75e81182   oaisoftwarealliance/oai-upf:develop           "/openair-upf/bin/oa_"   2 days ago   Up 8 hours (healthy)   2152/udp, 8805/udp                             oai-upf
81dbae4890ee   oaisoftwarealliance/oai-smf:develop           "/openair-smf/bin/oa_"   2 days ago   Up 8 hours (healthy)   80/tcp, 8080/tcp, 8805/udp                     oai-smf
3801c4cf809d   oaisoftwarealliance/oai-amf:develop           "/openair-amf/bin/oa_"   2 days ago   Up 8 hours (healthy)   80/tcp, 8080/tcp, 9090/tcp, 38412/sctp         oai-amf
03e0d9062fb3   oaisoftwarealliance/oai-ausf:develop          "/openair-ausf/bin/o_"   2 days ago   Up 8 hours (healthy)   80/tcp, 8080/tcp                               oai-ausf
137eb01411fc   oaisoftwarealliance/oai-udm:develop           "/openair-udm/bin/oa_"   2 days ago   Up 8 hours (healthy)   80/tcp, 8080/tcp                               oai-udm
7dc9c93e7cad   oaisoftwarealliance/oai-udr:develop           "/openair-udr/bin/oa_"   2 days ago   Up 8 hours (healthy)   80/tcp, 8080/tcp                               oai-udr
386ee3e62b48   oaisoftwarealliance/trf-gen-cn5g:jammy        "/bin/bash -c ' ip r_"   2 days ago   Up 8 hours (healthy)                                                  oai-ext-dn
4344c5f8e804   oaisoftwarealliance/oai-nrf:develop           "/openair-nrf/bin/oa_"   2 days ago   Up 8 hours (healthy)   80/tcp, 8080/tcp, 9090/tcp                     oai-nrf
e0d2d7a1f695   oaisoftwarealliance/ims:latest                "asterisk -fp"           2 days ago   Up 8 hours (healthy)                                                  ims
1d767a55732b   mysql:8.0                                     "docker-entrypoint.s_"   2 days ago   Up 8 hours (healthy)   3306/tcp, 33060/tcp                            mysql
```

Figure 5: OAI 5G CN functions deployed using Docker containers

- **Step 2: Deploying OAI gNB and UE;**

  We use the OAI-5G RAN for the deployment of gNB and UE setup along with RF Simulator and Channel Simulator. In this project, we use 2X2 MIMO setup where gNB is equipped with 2 Tx and Rx antennas, and UE is also equipped with 2 Tx and Rx antennas. The used 2X2 MIMO system can allow 2 spatial streams for multiplexing over the air interface if the CQI is good. We have provided all the configuration files used for the gNB and UE in the GitHub link: CQI-Prediction. For building and setting up of gNB and UE, we request you to please follow the setup guide: OAI Setup.md. Once the gNB and UE are built, the same can be run using the suitable executables as mentioned in the guide. Figs. 6 and 7 show the sample logs gNB and UE provide during execution.

```
[NR_MAC]   Frame.Slot 128.0
UE RNTI 1ea0 CU-UE-ID 1 in-sync PH 52 dB PCMAX 20 dBm, average RSRP -45 (16 meas)
UE 1ea0: CQI 15, RI 2, PMI (0,1)
UE 1ea0: dlsch_rounds 11/0/0/0, dlsch_errors 0, pucch0_DTX 0, BLER 0.08100 MCS (0) 9
UE 1ea0: ulsch_rounds 28/0/0/0, ulsch_errors 0, ulsch_DTX 0, BLER 0.02288 MCS (0) 9 (Qm 2  dB) NPRB 5  SNR 53.0 dB
UE 1ea0: MAC:    TX          2174 RX          1940 bytes
UE 1ea0: LCID 1: TX           194 RX           234 bytes

[NR_MAC]   Frame.Slot 256.0
UE RNTI 1ea0 CU-UE-ID 1 in-sync PH 52 dB PCMAX 20 dBm, average RSRP -45 (16 meas)
UE 1ea0: CQI 15, RI 2, PMI (0,1)
UE 1ea0: dlsch_rounds 12/0/0/0, dlsch_errors 0, pucch0_DTX 0, BLER 0.07290 MCS (0) 9
UE 1ea0: ulsch_rounds 40/0/0/0, ulsch_errors 0, ulsch_DTX 0, BLER 0.00646 MCS (0) 9 (Qm 2  dB) NPRB 5  SNR 53.0 dB
UE 1ea0: MAC:    TX          2415 RX          3332 bytes
UE 1ea0: LCID 1: TX           194 RX           234 bytes
```

Figure 6: OAI gNB logs when a UE is connected (gives CQI information acquired from UE)

```
[NR_PHY]   ==============================================
[NR_PHY]   [UE 0] Harq round stats for Downlink: 24/0/0
[NR_PHY]   ==============================================
[NR_PHY]   RSRP = -45 dBm, RI = 2 i1 = 0.0.0, i2 = 1, SINR = 37 dB, CQI = 15
[NR_PHY]   RSRP = -45 dBm, RI = 2 i1 = 0.0.0, i2 = 1, SINR = 37 dB, CQI = 15
[NR_PHY]   RSRP = -45 dBm, RI = 2 i1 = 0.0.0, i2 = 0, SINR = 37 dB, CQI = 15
[NR_PHY]   RSRP = -45 dBm, RI = 2 i1 = 0.0.0, i2 = 1, SINR = 37 dB, CQI = 15
[NR_PHY]   RSRP = -45 dBm, RI = 2 i1 = 0.0.0, i2 = 1, SINR = 37 dB, CQI = 15
[NR_PHY]   RSRP = -45 dBm, RI = 2 i1 = 0.0.0, i2 = 1, SINR = 37 dB, CQI = 15
[NR_PHY]   RSRP = -45 dBm, RI = 2 i1 = 0.0.0, i2 = 1, SINR = 37 dB, CQI = 15
[NR_PHY]   RSRP = -45 dBm, RI = 2 i1 = 0.0.0, i2 = 1, SINR = 37 dB, CQI = 15
```

Figure 7: OAI UE logs after registering with network (gives CQI information acquired using CSI-RS signals)

- **Step 3: Deploying FlexRIC (near RT-RIC)**
  In this project, we use FlexRIC as the near RT-RIC for both UE specific CQI data collection and prediction. Hence, to set up the FlexRIC, we request you to please consider using the setup guide: OAI Setup.md. Below Fig. 8 shows the FLexRIC deployment, and one can observe that the FlexRIC supports various service models including statistics for PDCP, MAC, RLC, GTP, etc. In this proposed work, we will be using CQI metrics for dataset collection and hence, the MAC SM is of key importance to us. Accordingly, we develop the xApp for acquiring the MAC level statistics. Also when the gNB connects to the FlexRIC, the RIC accepts the functions supported by RAN.

```
[NEAR-RIC]: Initializing
[NEAR-RIC]: Loading SM ID = 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Loading SM ID = 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Loading SM ID = 146 with def = TC_STATS_V0
[NEAR-RIC]: Loading SM ID = 2 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Loading SM ID = 142 with def = MAC_STATS_V0
[NEAR-RIC]: Loading SM ID = 3 with def = ORAN-E2SM-RC
[NEAR-RIC]: Loading SM ID = 143 with def = RLC_STATS_V0
[NEAR-RIC]: Loading SM ID = 148 with def = GTP_STATS_V0
[iApp]: Initializing ...
[iApp]: nearRT-RIC IP Address = 127.0.0.1, PORT = 36422
[NEAR-RIC]: Initializing Task Manager with 2 threads
[E2AP]: E2 SETUP-REQUEST rx from PLMN   1. 1 Node ID 3584 RAN type ngran_gNB
[NEAR-RIC]: Accepting RAN function ID 2 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Accepting RAN function ID 3 with def = ORAN-E2SM-RC
[NEAR-RIC]: Accepting RAN function ID 142 with def = MAC_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 143 with def = RLC_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 146 with def = TC_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 148 with def = GTP_STATS_V0
```

Figure 8: Deployment of FlexRIC

- **Step 4: Preparing the Matlab and Expect script for data collection**

  In this work, we try to do CQI predictions using a suitable AI model (Bidirectional-LSTM in this case). For the same, we will be needing a CQI dataset for performing the model training. Currently, there are a minimal number of datasets providing the CQI information in 5G. Hence, we generate a custom dataset using OAI RF simulator. For the same to introduce channel variations over the time, we need to adjust the noise power and fading parameters. To implement this we have taken a two way approach, where we develop a Matlab script for generating an Linux-based Expect script. The generated Expect script can be run in linux terminal and it connects to the UE over a telnet session to continuously vary the channel parameters. The developed Matlab script is shown in below Fig. 9:

```
clc;
clear all;
close all;
N = 50000; % Number of data points
interval = 0.1; % In seconds
r = [-15, -5]; % Range of noise power in dB
fileID = fopen('channel_parameter_simulator_validation.exp','w');
fprintf(fileID,'#!/usr/bin/expect -f\n');
```

```
fprintf(fileID,'\n\n# Developed by Wireless Communications and Navigation Lab, IIT
Jodhpur\n');
fprintf(fileID,'# Automated expect script generated for channel parameter variation
using MATLAB\n');
fprintf(fileID,'# This script continuously varies the DL noise power in the range
[-15 dB, -5 dB] thus inducing CQI variations\n');
fprintf(fileID,'# This is used for CQI training dataset generation later used for
Bi-LSTM model training\n\n\n');
% To change the channel parameters in DL (at UE)
fprintf(fileID,'spawn telnet 127.0.0.1 9091\n');
fprintf(fileID, 'send "\\r\"\n');
fprintf(fileID,'expect "softmodem_5Gue"\n');
fprintf(fileID,'send "channelmod modify 0 noise_power_dB -50\\r"\n');
fprintf(fileID,'sleep %0.2f\n', interval);
% Generate sample data
for i=1:N
   noise_power_dB = randi(r);
   fprintf(fileID,'send "channelmod modify 0 noise_power_dB
%d\\r"\n',noise_power_dB);
   if(mod(i,1000)==0)
       fprintf(fileID,'puts "Progress: %d/%d (%0.2f %%)"\n', i,N,i*100/N);
   end
   fprintf(fileID,'sleep %0.2f\n', interval);
end
```

Figure 9: Matlab script written to generate the Expect script which can induce the channel variations in OAI RF Simulator

The above matlab script generates an output file of xxxx.exp, which can be then executed for inducing the channel parameter changes in the OAI RF simulator. The sample Expect scripts generated for training purposes and validation purposes can be viewed using the links: [channel_parameter_simulator.exp]() and [channel_parameter_simulator_validation.exp](), respectively. Also, the Fig. 10 shows a portion of the raw Expect file where we are changing the noise power every 100ms duration in the downlink.

```
#!/usr/bin/expect -f


# Developed by Wireless Communications and Navigation Lab, IIT Jodhpur
# Automated expect script generated for channel parameter variation using
MATLAB
```

```
# This script continuously varies the DL noise power in the range [-15 dB,
-5 dB] thus inducing CQI variations
# This is used for CQI training dataset generation later used for Bi-LSTM
model training


spawn telnet 127.0.0.1 9091
send "\r"
expect "softmodem_5Gue"
send "channelmod modify 0 noise_power_dB -50\r"
sleep 0.10
send "channelmod modify 0 noise_power_dB -7\r"
sleep 0.10
send "channelmod modify 0 noise_power_dB -8\r"
sleep 0.10
send "channelmod modify 0 noise_power_dB -10\r"
sleep 0.10
send "channelmod modify 0 noise_power_dB -7\r"
sleep 0.10
send "channelmod modify 0 noise_power_dB -5\r"
sleep 0.10
send "channelmod modify 0 noise_power_dB -15\r"
sleep 0.10
```

Figure 10: A sample Expect script developed using Matlab to induce the channel variations in OAI RF Simulator

- **Step 5: Deploying the xApp for CQI dataset collection**
  For the purpose of the CQI dataset collection, we developed an xApp which subscribes to the MAC SM. The xApp interacts with the FlexRIC using E42 agent to receive MAC indication messages communicated by gNB. The relevant code used for the subscription is shown below:

```
# Initialize the RIC SDK
ric.init()

# Check the connected nodes (gNB in this case)
conn = ric.conn_e2_nodes()
assert(len(conn) > 0)

# Print the E2 nodes information , if any
for i in range(0, len(conn)):
```

```
    print("Global E2 Node [" + str(i) + "]: PLMN MCC = " +
str(conn[i].id.plmn.mcc))
    print("Global E2 Node [" + str(i) + "]: PLMN MNC = " +
str(conn[i].id.plmn.mnc))

####################
#### CONFIGURING MAC INDICATION CALLBACK FOR MONITORING AND PREDICTION
####################

# Configure the MAC SM indications with 10 ms intervals (corresponding to
every frame) along with callback config
mac_hndlr = []
for i in range(0, len(conn)):
    mac_cb = MACCallback()
    hndlr = ric.report_mac_sm(conn[i].id, ric.Interval_ms_10, mac_cb)
    mac_hndlr.append(hndlr)
    time.sleep(1)
```

Figure 11: xApp code for subscribing to the MAC indication messages

Initially, we begin with initializing the RIC SDK and then fetch the E2 nodes connected to the RIC. AFter the E2 nodes information is acquired, we subscribe to the MAC indication messages. Upon running the xApp, we can infer that the data collected is being stored in a SQLite DB as mentioned in the below Fig. 12.

```
[NEAR-RIC]: Loading SM ID = 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Loading SM ID = 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Loading SM ID = 146 with def = TC_STATS_V0
[NEAR-RIC]: Loading SM ID = 2 with def = ORAN-E2SM-KPM
[NEAR-RIC]: Loading SM ID = 142 with def = MAC_STATS_V0
[NEAR-RIC]: Loading SM ID = 3 with def = ORAN-E2SM-RC
[NEAR-RIC]: Loading SM ID = 143 with def = RLC_STATS_V0
[NEAR-RIC]: Loading SM ID = 148 with def = GTP_STATS_V0
[xApp]: DB filename = /tmp/xapp_db_1727448995861362
[xApp]: E42 SETUP-REQUEST tx
[xApp]: E42 SETUP-RESPONSE rx
[xApp]: xApp ID = 10
[xApp]: Registered E2 Nodes = 1
Global E2 Node [0]: PLMN MCC = 1
Global E2 Node [0]: PLMN MNC = 1
Model: "sequential"
```

Figure 12: Logs of xApp indicating the DB name used for data storage

For viewing the full xApp developed in this work, one can refer to: xapp_mac_stats_prediction.py. Also, for accessing the training data used in this work, one can refer to the DB: CQI_DATASET shared in the GitHub repository. To view the data, use the DB Browser application provided by SQLite3.

3.2 Relevance with ITU Y.3172 specifications

The proposed reference architecture in ITU Y.3172 specifications is shown in below Fig. 13:

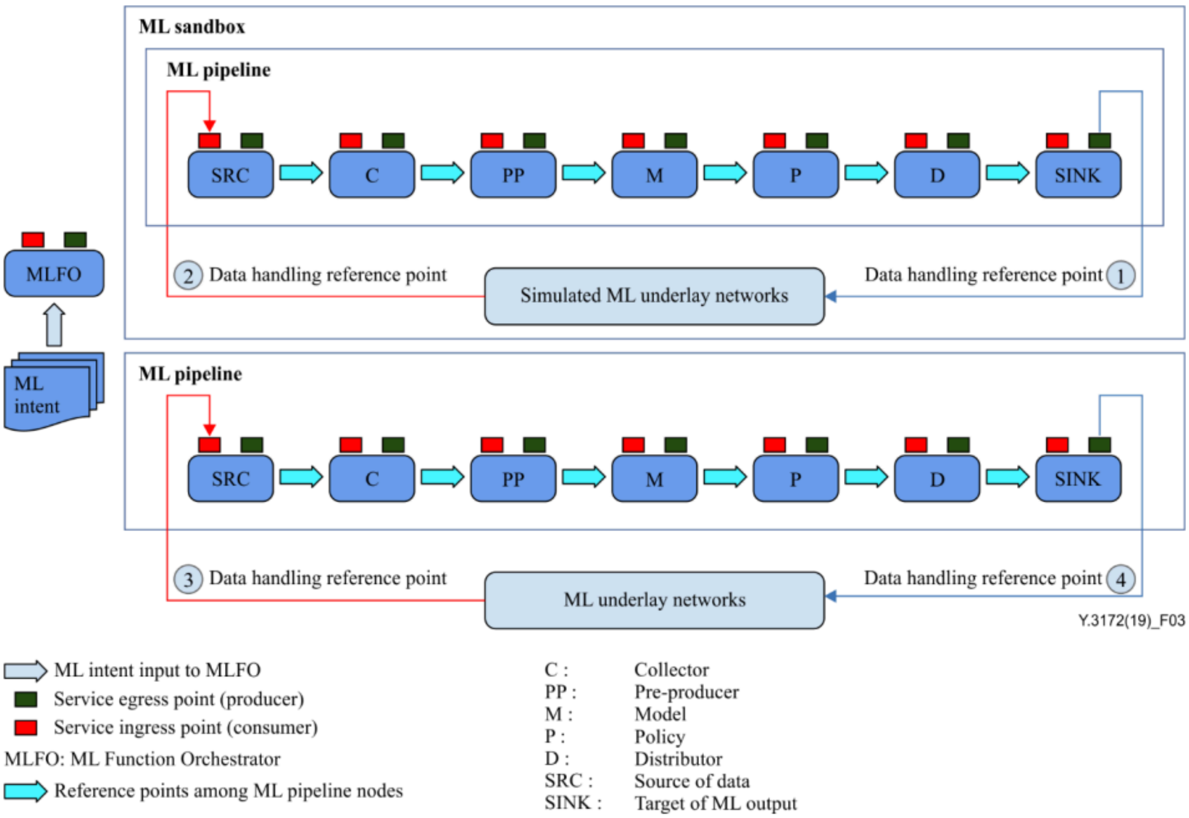

Figure 13: High-level architectural components proposed in ITU Y.3172 specifications

In reference to the above, we map the different components considered in this proposed work as shown in Fig. 14 below.

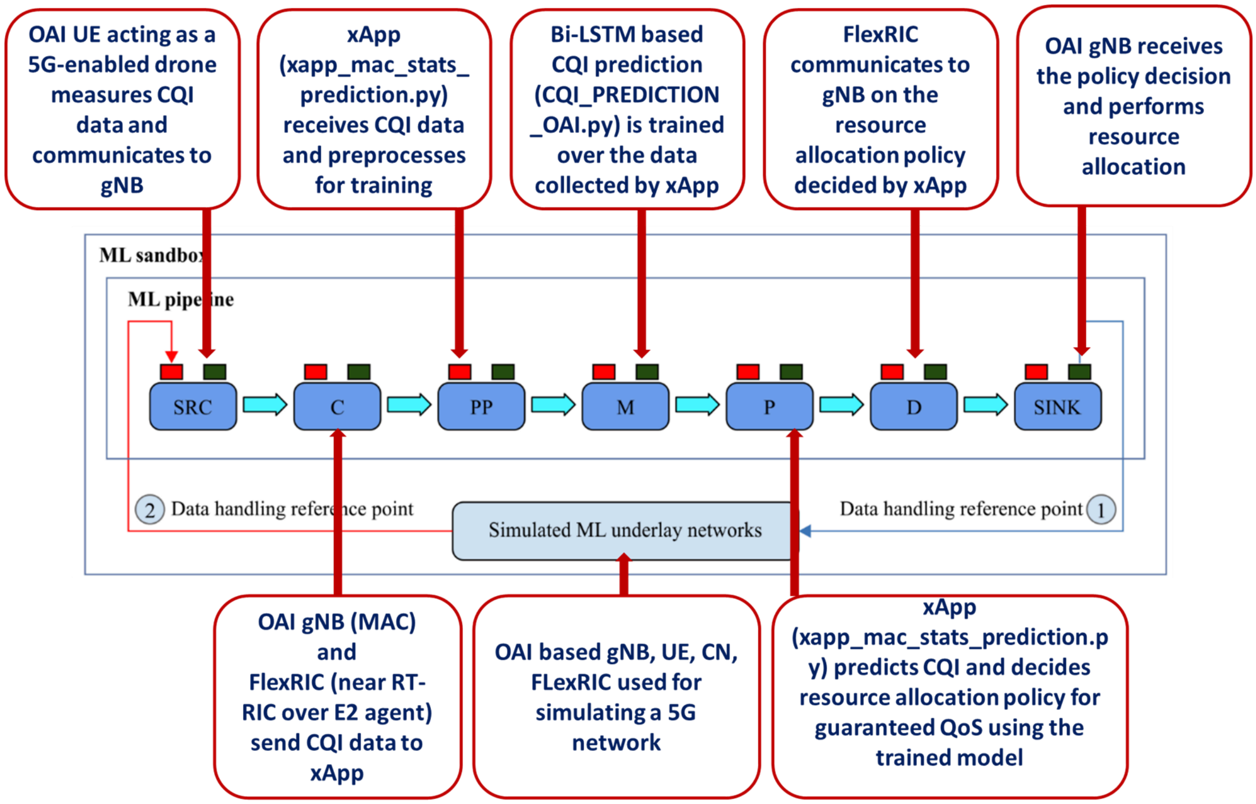

Figure 14: Mapping of the developed architectural components in this hackathon with respect ITU Y.3172 architecture

4. PS-2: xApp Design for the Use-case "A Real Time CQI Prediction Framework for Proactive Resource Scheduling in 5G Enabled Drones Using AI"

The proposed use-case targets real-time prediction of the CQI data on a per-frame basis which can be used for dynamic resource allocation in 5G for supporting guaranteed QoS services such as 5G-enabled Drones, Tele-surgeries, ROVs, etc. (an example of uRLLC use-cases). As part of this, we utilize Open Air Interface 5G protocol stack for simulating the sandbox consisting of gNB, CN, UE, and FlexRIC. The proposed architecture is shown in Fig. 4.

In this work, we have developed an xApp (xapp_mac_stats_prediction.py) which can perform the following functions:

a. **CQI data collection:** The xApp interacts with FlexRIC over E42 agent and aggregates the MAC indication messages sent by the gNB. The MAC indication messages will comprise of the CQI data per UE per frame. The aggregated data is collected into a SQLite3 database which can be used for training purposes.

b. **Model Training:** From the collected CQI dataset, we perform offline AI model training. The proposed AI model in this work is based on Bi-LSTM model with SELU activation units (CQI_PREDICTION_OAI.py) [6,7]. The trained model (trained_model.keras) along with the MinMaxScaler (scaler_training.bin) used for training purposes are also exported as part of this. The architecture of the proposed Bi-LSTM model with SELU activation is summarized below.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirection  (None, 400, 50)           5400
 al)

 bidirectional_1 (Bidirecti  (None, 50)                15200
 onal)

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 20651 (80.67 KB)
Trainable params: 20651 (80.67 KB)
Non-trainable params: 0 (0.00 Byte)
```

c. **Prediction Step:** The trained model is then used by the xApp (xapp_mac_stats_prediction.py) for predicting the CQI values for the future frames.

4.1 xApp code description

For the full code of the xApp, please refer to the link: xapp_mac_stats_prediction.py. Also, the data collection modality is already described in the Section 3.1. Hence, in the following, we primarily focus on the prediction part of the xApp. The different steps involved in designing the xApp include the following

- Initializing the RIC SDK and connection with the FlexRIC over E42 agent

```
# Initialize the RIC
ric.init()
```

- Connecting to E2 agents (in this case OAI gNB)

```
# Check the connected nodes (gNB in this case)
conn = ric.conn_e2_nodes()
# Print the E2 nodes information , if any
for i in range(0, len(conn)):
    print("Global E2 Node [" + str(i) + "]: PLMN MCC = " +
str(conn[i].id.plmn.mcc))
    print("Global E2 Node [" + str(i) + "]: PLMN MNC = " +
str(conn[i].id.plmn.mnc))
```

- Create MAC indication callbacks to handle the incoming MAC indication messages sent by gNB

```
####################
#### CONFIGURING MAC INDICATION CALLBACK FOR MONITORING AND PREDICTION
####################

# Configure the MAC SM indications with 10 ms intervals (corresponding to
every frame) along with callback config
mac_hndlr = []
for i in range(0, len(conn)):
    mac_cb = MACCallback()
    hndlr = ric.report_mac_sm(conn[i].id, ric.Interval_ms_10, mac_cb)
    mac_hndlr.append(hndlr)
    time.sleep(1)
```

- Create the MAC callback class with required variables for prediction

```
####################
#### MAC INDICATION CALLBACK & AI MODEL FOR CQI PREDICTION
####################
```

```
#  MACCallback class is defined and derived from C++ class mac_cb
#  Whenever the RIC indication message is received, this callback function
is called
#  Once we receive the CQI update, we use the recent CQI data corresponding
to 400 frames and make prediction for the next frame CQI

class MACCallback(ric.mac_cb):
    # Variables required for preparing input data. Trained model is already
saved
    prev_frame = 0
    ready = 0

    # Required for storing past 400 CQI values for predicting the next CQI
value
    input = np.empty(400)

    # Required for loading the trained model
    model = Model()

    # Required for loading the MinMaxScaler used during training
    scaler = MinMaxScaler(feature_range=(0, 1))

    # Predicted CQI and number of CQIs predicted
    pred_CQI = 0
    pred_count = 0;

    # Variables for statistics of the model performance
    accuracy = 0;
    mae = 0;
    mse = 0;
    t = time.time()
    pred_log = '';
    # Define Python class 'constructor'
```

- Create the constructor for the MACCallback class and load the trained model and scaler

```
 # Define Python class 'constructor'
   def __init__(self):

       # Call C++ base class constructor
       ric.mac_cb.__init__(self)
```

```
        # Load the trained model for real-time prediction
        self.model = tf.keras.models.load_model('trained_model.keras')
        self.model.summary()

        # Load the scaler used during the training (MinMaxScaler)
        self.scaler = load('scaler_training.bin')
        print('Min: %f, Max: %f' % (self.scaler.data_min_,
self.scaler.data_max_))
```

- Prepare the main logic for prediction in the handle function. The important steps that will occur are as follows:
    - The Bi-LSTM model used for predicting CQI values requires CQI values corresponding to present and past 400 frames.
    - Hence, wait until the input array is filled before making any predictions
    - Once, the input array is filled, we use it to make the first prediction.
    - Later on as soon as we receive an actual CQI update from the MAC indication message, we use a sliding window mechanism to update the input array and then make further predictions of the next frame.
    - Note that the prediction can be extended to anytime in ths future with suitable training. Predicting 1 frame ahead is a decision taken considering the latency constraints of uRLLC use-cases which are in the order of (10 ms to 30 ms).
    -

```
 # Override C++ method: virtual void handle(swig_mac_ind_msg_t a) = 0;
    def handle(self, ind):
        # Print swig_mac_ind_msg_t
        if len(ind.ue_stats) > 0:
           if (ind.ue_stats[0].frame != self.prev_frame):

              # Wait until the input buffer is filled. Our input for
prediction is recent CQI values for 400 frames
              if (self.ready < 400):
                 self.input[self.ready] = ind.ue_stats[0].wb_cqi
                 self.ready = self.ready + 1

              # Once the input is ready, predict the CQI value of upcoming
frame every time
              else:
```

```
                 # Perform left shift to insert the recent CQI at the end
of the input
                 self.input[0:398] = self.input[1:399]
                 self.input[399] = ind.ue_stats[0].wb_cqi
                 try:
                    self.pred_log = self.pred_log +
'['+str(self.prev_frame) + ',' + str(int(self.input[-1])).rjust(2, '0') +
',' + str(int(self.pred_CQI)).rjust(2, '0')>
                    if(self.pred_count%10==0):
                       self.pred_log = self.pred_log + '\n'
                    if(self.pred_count>0):
                       if(self.input[-1]==self.pred_CQI):
                          self.accuracy = self.accuracy+1
                       error = np.abs(self.input[-1]-self.pred_CQI)
                       self.mae = self.mae+ error
#                     print('Frame: ' + str(self.prev_frame) + ', Real CQI:
'+ str(self.input[-1]) + ', Predicted CQI: ' + str(self.pred_CQI))
                       if(self.pred_count%100==0):
                          error_mae =
round(self.mae.item()/self.pred_count,2)
                          error_mse =
round(self.mse.item()/self.pred_count,2)
                          acc = round(self.accuracy*100/self.pred_count,2)
                          print(self.pred_log)
                          print('Stats Summary (100 frames) - Time Elapsed:
'+ str(round(time.time()-self.t,2)) + ' Sec, MAE: ' + str(error_mae) + '
(CQI), MSE: ', str(e>

#                         To print accuracy enable the below line and
comment the above line
#                          print('Time Elapsed: '+
str(round(time.time()-self.t,2)) + ' Sec, Frame: ' + str(self.prev_frame) +
', Accuracy: '+ str(acc) + '%, MAE: ' + st>


print('---------------------------------------------------------------------
----------------')
                          self.pred_log = '\n\nFrame Level Predictions
[Frame Number, Actual CQI, Predicted CQI]\n\n'

                    self.prev_frame = ind.ue_stats[0].frame

                    # Scale the input using the MinMaxScaler fitted during
```

```
the training procedure. Scales from [0,15] -> [0,1]
                    normalized_in =
self.scaler.transform(self.input.reshape(len(self.input),1))


                    # Make the prediction and do inverse of scaling to get
the CQI mapping of [0,15] from scaled prediction [0,1]
                    self.pred_CQI =
np.rint(self.scaler.inverse_transform(self.model.predict(normalized_in.resh
ape((1, 400, 1)), verbose=0)))
                    if(self.pred_CQI>15): self.pred_CQI = 15
                    self.pred_count = self.pred_count + 1
                 except Exception as e: print(e)
```

4.2 Relevance with ITU Y.3172 specifications

Fig. 14 shows the relevance of the proposed use-case with the autonomous network architecture proposed in ITU Y.3172 specifications by logically mapping the important components.

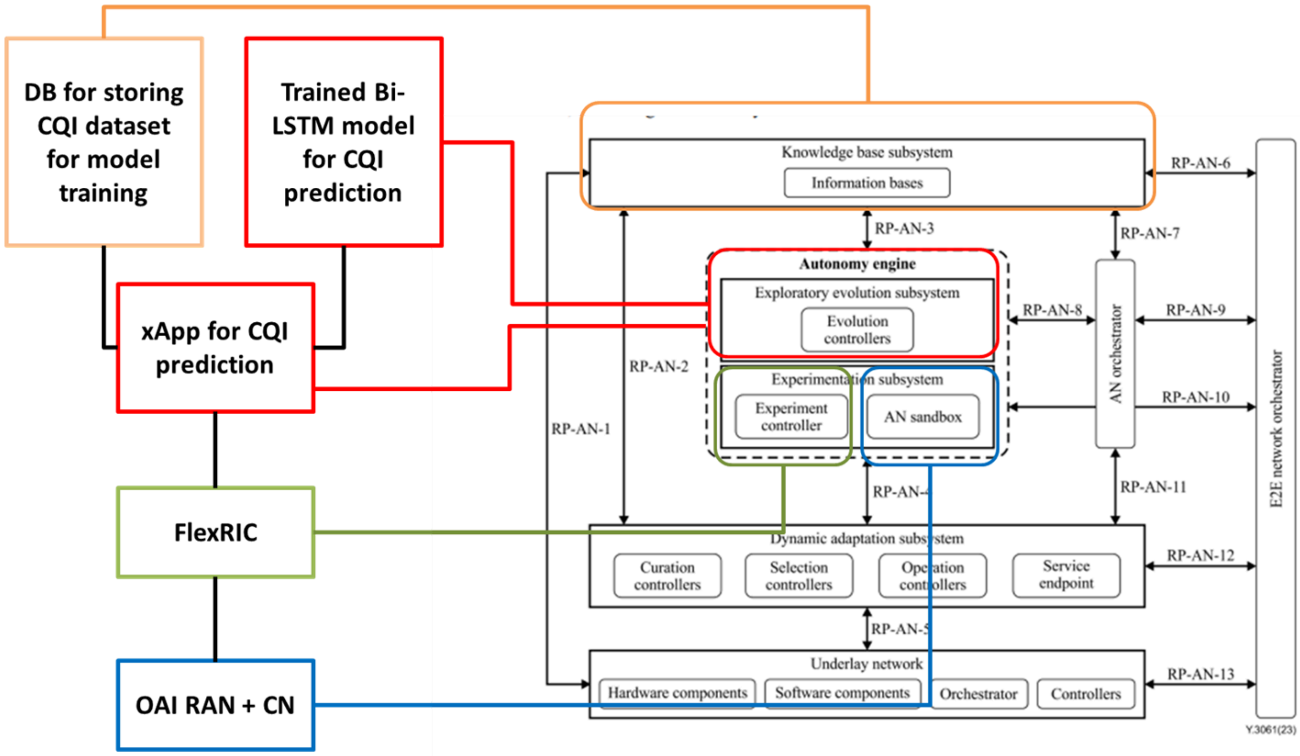

Figure 15: Mapping of the developed architectural components in this hackathon with respect ITU Y.3061 architecture

5. Relation to Standards:

The proposed architecture is compatible with the following standards:

a. OAI CU & DU are O-RAN compliant disaggregated baseband units. The CU contains both CU-C & CU-U functionality and supports PDCP,GTPU,RRC & S1AP protocols along with S1, F1 & E2 interfaces. The DU has High-PHY(FAPI), MAC, RLC & RRC (for handing RRC Config messages from CU) protocols along with F1 interface support. Both CU & DU implements O-RAN's E2AP interface. Hence, the proposed use-case can be integrated with any O-RAN compliant 5G telecom network.

b. The FlexRIC used in the proposed use-case is compatible with E2 interface and hence, the xApp developed can also be integrated with any O-RAN compliant 5G telecom network.

c. OAI is compliant with 3GPP Rel. 15 and 16.

d. The proposed xApp architecture is also compatible with the ITU Y.3172 and Y.3061 specifications as discussed earlier.

In addition to the above, the proposed use-case is in compliance with multiple ITU specifications as mentioned in the below Fig. 16.

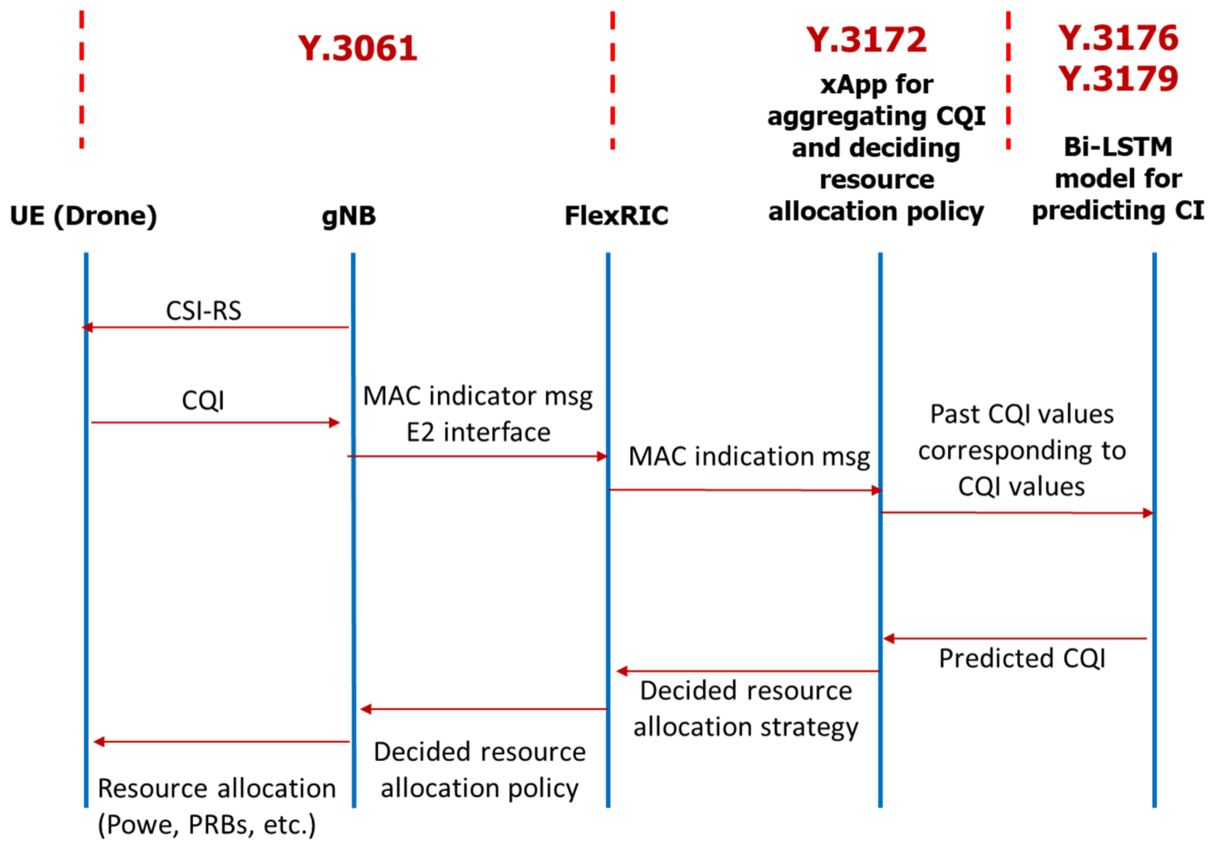

Figure 16: Mapping of the proposed use-case relevance to various ITU specifications

6. Code Submission Details:

The complete code (with multiple files) along with the necessary documentation is uploaded into the following GitHub repository: mprsk/CQI-Prediction (github.com)

The description of the important files used for the demonstration of the proposed use-case include the following:

1. DataPreparation.m: A MATLAB script written for preparing an expect script that can automate varying channel parameters (noise power in this case) over telnet session in OAI RF Simulator periodically. This will help us in generating a CQI database (CQI_DATASET) that can be used for model training.
2. channel_parameter_simulator.exp: A sample expect script generated using DataPreparation.m where noise power is modified every 100ms in the range of [-15 dB, -5 dB].
3. channel_parameter_simulator_validation.exp: Another expect script generated using DataPreparation.m where noise power is modified every 100ms in the range of [-15 dB, -5 dB] used for validation. This is created for testing the developed AI model performance with unseen data during training.
4. CQI_DATASET: A sample CQI dataset (SQLite3 DB) generated using the channel_parameter_simulator.exp script. The dataset is acquired using the xApp (xapp_mac_stats_prediction.py)
5. xapp_mac_stats_prediction.py: xAPP compatible with FlexRIC and OAI 5G Protocol Stack used for CQI dataset collection and real-time prediction. The ML model is based on Bi-LSTM with SeLu activation units. The xApp simultaneously lodges the CQI data collected into SQLite3 DB and performs prediction. During validation of the model, the channel variations can be induced using a new expect script generated using channel_parameter_simulator.exp and DataPreparation.m
6. CQI_PREDICTION_OAI.py: Bi-LSTM model with SELU activation units developed for prediction of CQI in Python 3. The model takes the input of CQI values for the past 400 frames and predicts the CQI value of the upcoming frame.
7. trained_model.keras: The trained model saved using Keras libraries and can be used for prediction or validation. This is used in the xApp (xapp_mac_stats_prediction.py) for validation of the model and prediction of CQI
8. scaler_training.bin: MinMaxScaler used during training. This is required for prediction in xApp (xapp_mac_stats_prediction.py)

9. gnb.sa.band78.fr1.106PRB.2x2.usrpn300.conf: OAI gNB configuration file for supporting 2x2 MIMO with 40 MHz bandwidth, 30 KHz SCS, and TDD configuration
10. ue.conf: OAI UE configuration file for supporting 2x2 MIMO
11. channelmod_rfsimu.conf: Channel model configuration for OAI RF Simulator

7. Self Testing Results

As part of this, the two major activities performed include the following:

a. **AI model training:** The Bi-LSTM model training over 15 epochs took approximately 30 minutes on a computing hardware with Intel i9 processor with 16 GB RAM. The mean absolute error (MAE, considered as loss function) at the end of the training is well within 0.006 (after MinMaxScaler based scaling of input data with fit range [0,1]) as shown in Fig. 17.

```
-----------------------------------------------------------------
Epoch 1/15
928/928 [==============================] - 119s 127ms/step - loss: 0.0148
Epoch 2/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0095
Epoch 3/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0080
Epoch 4/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0070
Epoch 5/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0063
Epoch 6/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0059
Epoch 7/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0058
Epoch 8/15
928/928 [==============================] - 117s 127ms/step - loss: 0.0059
Epoch 9/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0057
Epoch 10/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0056
Epoch 11/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0056
Epoch 12/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0056
Epoch 13/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0055
Epoch 14/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0055
Epoch 15/15
928/928 [==============================] - 118s 127ms/step - loss: 0.0055
Training Time: 1767.8 seconds.
```

Figure 17: Training of the proposed Bi-LSTM model for predicting the CQI

Also, the testing accuracy on the unseen data at the end of the training process is observed to be as shown in Fig. 18.

```
9400 / 9598
9500 / 9598
0.3452802667222338
Accuracy:  0.8788289226922276
MAE: 0.35
MSE: 1.23
```

Figure 18: Testing accuracy of the proposed model

b. **Validation of xApp:** The xApp developed is validated in integration with the proposed AI model. The validation flow consisted of OAI gNB, CN, UE, and FlexRIC running. FOr inducing the channel variations, we have used the Expect script [channel_parameter_simulator_validation.exp]() which was generated with a different seed in Matlab thus ensuring to verify if the model training is generalized and no overfitting happens. The real-time logs generated by the xApp along with aggregated MAE and MSE is shown in Fig. 19 below. It can be observed that the MAE and MSE are less than 0.5 CQI units and 2 CQI^2 units, respectively.

```
Frame Level Predictions [Frame Number, Actual CQI, Predicted CQI]

[294,12,12],   [295,12,12],   [296,12,12],   [297,12,12],   [298,12,12],   [299,12,12],   [300,12,12],   [301,12,12],   [302,12,12],   [303,12,12],
[304,08,12],   [305,08,09],   [306,08,09],   [307,08,08],   [308,08,08],   [309,08,08],   [310,08,08],   [311,08,08],   [312,07,08],   [313,07,07],
[314,07,07],   [315,07,07],   [316,07,07],   [317,07,07],   [318,07,07],   [319,07,07],   [320,07,07],   [321,07,07],   [322,07,07],   [323,07,07],
[324,07,07],   [325,07,07],   [326,07,07],   [327,07,07],   [328,12,07],   [329,12,12],   [330,12,12],   [331,12,12],   [332,12,12],   [333,12,12],
[334,12,12],   [335,12,12],   [336,11,12],   [337,11,11],   [338,11,11],   [339,11,11],   [340,11,11],   [341,11,11],   [342,11,11],   [343,11,11],
[344,12,11],   [345,12,12],   [346,12,12],   [347,12,12],   [348,12,12],   [349,12,12],   [350,12,12],   [351,12,12],   [352,06,12],   [353,06,06],
[354,06,06],   [355,06,06],   [356,06,06],   [357,06,06],   [358,06,06],   [359,06,06],   [360,10,06],   [361,10,11],   [362,10,11],   [363,10,11],
[364,10,11],   [365,10,10],   [366,10,10],   [367,10,10],   [368,12,11],   [369,12,12],   [370,12,12],   [371,12,12],   [372,12,12],   [373,12,12],
[374,12,12],   [375,12,12],   [376,07,12],   [377,07,07],   [378,07,07],   [379,07,07],   [380,07,07],   [381,07,07],   [382,07,07],   [383,07,07],
[384,13,07],   [385,13,13],   [386,13,13],   [387,13,13],   [388,13,13],   [389,13,13],   [390,13,13],   [391,13,13],   [392,08,13],   [393,08,08],

Stats Summary (100 frames) - Time Elapsed: 24.17 Sec, MAE: 0.48 (CQI), MSE:  1.57 (CQI^2)
-----------------------------------------------------------------------------------------
```

Figure 19: Real-time prediction output generated by xApp along with sliding MAE and MSE

Also, one thing to note here is the proposed AI model has an inference latency of <5 ms with a total number of 20651 trainable parameters making it light-weight. Hence, the proposed use-case is suitable for real-time implementation in O-RAN compliant 5G telecom networks.

8. References

1. OIG 2020, Next Generation Connectivity: Postal Service Roles in 5G and Broadband RISC report, Deployment, https://www.uspsoig.gov/sites/default/files/reports/2023-01/RISC-WP-20-007.pdf
2. Nameer Hashim Qasim, Aqeel Mahmood Jawad, 5G-enabled UAVs for energy-efficient opportunistic networking, Heliyon 10 (2024) e32660
3. QCOMMR 2024, What's the role of sensing for next-generation wireless networks? Accessed from https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/Whats-the-role-of-sensing-for-next-generation-wireless-networks.pdf
4. 5GAMERCAS, 2023, 5G Use Cases, November 2023, White Paper Accessed from https://www.5gamericas.org/wp-content/uploads/2023/11/5G-Use-Cases.pdf
5. OpenAirInterface – 5G software alliance for democratising wireless innovation
6. ML-based Traffic Steering for Heterogeneous Ultra-dense beyond-5G Networks (https://ieeexplore.ieee.org/abstract/document/10118923)
7. Which ML Model to Choose? Experimental Evaluation for a Beyond-5G Traffic Steering Case (https://ieeexplore.ieee.org/abstract/document/10279485)
8. Julio Diez-Tomillo, Jose M. Alcaraz-Calero, Qi Wang, Face Verification Algorithms for UAV Applications: An Empirical Comparative Analysis, Journal of Communications Software and Systems, Vol. 20, No. 1, p. 1-12.
9. ITU-T Recommendation Y.3061, Autonomous networks – Architecture framework.
10. ITU-T Recommendation Y.3172, Autonomous networks – Architecture framework.
11. ITU-T Recommendation Y.3176, Autonomous networks – Architecture framework.
12. ITU-T Recommendation Y.3179, Autonomous networks – Architecture framework.