Using Public Datasets to Train O-RAN Deep Learning Models

Rodrigo S. Couto*, Pedro Cruz*, Miguel Elias M. Campista*, Luís Henrique M. K. Costa*
* GTA/DEL-POLI/PEE-COPPE - Universidade Federal do Rio de Janeiro (UFRJ) - Rio de Janeiro, Brazil
Email: {rodrigo, cruz, miguel, luish}@gta.ufrj.br

Abstract—**The disaggregation promoted by the O-RAN architecture allows for unprecedented flexibility in Radio Access Networks (RANs). The existence of specific components to control the infrastructure, such as RAN Intelligent Controllers (RICs), places intelligence at the center of the management and orchestration mechanisms of these networks. Hence, deep learning plays a crucial role in developing these solutions. As deep learning heavily relies on data for model training and generalization, using public datasets becomes essential to facilitate research and foster advancements in O-RAN. This paper surveys the primary public datasets available online used in O-RAN research. Our goal is to overview these datasets and act as a complement to their documentation.**

I. Introduction

The O-RAN architecture allows the disaggregation of Radio Access Networks (RANs), fostering innovation and reducing vendor lock-in [1]. In this architecture, three separated components implement Base Station (BS) functions: O-RU (O-RAN Radio Unit), O-DU (O-RAN Distributed Unit), and O-CU (O-RAN Central Unit). The existence of specific components to control RANs, such as Non-Real-Time (Non-RT) and Near-Real-Time (Near-RT) RAN Intelligent Controllers (RICs), places intelligence at the core of O-RAN management and orchestration mechanisms, which is a major aspect envisioned for 6G networking [2]. The Non-RT RIC has a global view of the infrastructure and operates through rApps, with timescales larger than one second. The Near-RT RIC controls a specific set of RAN components through xApps, operating at timescales between ten milliseconds and one second [3].

Near-RT RICs generally manage user sessions and medium access [4]. Their xApps may calculate network slicing scheduling strategies, establish load balancing mechanisms, and manage handover policies. For instance, a xApp can run machine learning models to predict the availability of different RAN resources. In this case, the models receive the current state of the RAN, including its Key Performance Indicators (KPIs), and forecast the demand for resource utilization. These resources may include computational aspects like processing, storage, and memory or communication-related aspects like spectral utilization. With its global view of the infrastructure, a Non-RT RIC has rApps that can act by configuring the xApps and defining their policies and parameters.

Deep learning models are highly present in xApps and rApps [5]. However, their training necessitates substantial data input, which may be prohibitive for those researchers that do not dispose of an O-RAN testbed. The need for specific hardware and operation on licensed radio spectrum makes it difficult to build this type of infrastructure. This accentuates the need for diverse and extensive public datasets to foster O-RAN research. In addition, the diversity of O-RAN data offered by public datasets helps to achieve model generalization. Finally, public datasets can be used as benchmarks to compare different proposals.

In this work, we survey the available public datasets that can be used in deep learning models for O-RAN. We select the most important datasets used in the O-RAN literature and describe their organization and collected metrics. Our goal is to shed light on the publicly available data and act as a complement to the datasets' documentation.

This work is organized as follows. Section II describes the related work, while Section III overviews the main tools and testbeds used to build the datasets. Next, Section IV details the datasets, showing their structure and collected metrics. Finally, Section V concludes this work.

II. Related Work

Deep learning techniques are widely present in O-RAN literature. For example, Bonati *et al.* employ Deep Reinforcement Learning (DRL) to choose the best radio link scheduling policies [4]. Baldesi *et al.* use Residual Networks (RNs) and Convolutional Neural Networks (CNNs) to classify the traffic in a RAN. Hojeij *et al.* use a Recurrent Neural Network (RNN) to decide the placement of O-DUs and O-CUs along the computing infrastructure [6]. Rezazadeh *et al.* employ Federated Deep Reinforcement Learning (FDRL) to tune the amount of radio link resources given to each network slice [7]. The works in [5], [8] surveys different proposals that use deep learning and other machine learning approaches in open radio access networks.

The adoption of deep learning algorithms in cellular networks is a topic that can be independent of the work conducted on the O-RAN architecture [9]–[11]. However, one can adapt these solutions by considering O-RAN interfaces and components. Accordingly, O-RAN approaches may employ datasets already used in the general area of mobile networking. The work in [12] overviews these datasets.

Despite considering the available RAN datasets as in [12], we only survey the ones already used in an O-RAN paper. Since we focus on a few datasets, we are able to describe some of their details and act as a complement to their documentation.

III. Tools and testbeds

The public datasets surveyed in this work are collected in testbeds that use Universal Software Radio Peripheral (USRP) to implement User Equipment (UE) and Base Stations (BSs). A USRP is a hardware designed by Ettus Research[1] commonly used in Software-Defined Radio (SDR) applications. The idea of an SDR is to provide a flexible and programmable platform for radio frequency (RF) communications. Due to their flexibility, USRP devices are largely used in testbeds. A USRP architecture is generally composed of two modules. The motherboard is the main hardware component, including a CPU, memory, an FPGA (Field-Programmable Gate Array), and other interfaces that connect to the host computer, enabling data transfer and USRP's control. The motherboard thus processes the baseband signal. In turn, the daughterboard transmits and receives RF signals and contains the RF circuitry, such as mixers, filters, amplifiers, and transceivers.

Another common component used in O-RAN testbeds is a RAN implementation provided by the srsRAN project[2]. srsRAN has software stacks to implement O-CUs, O-DUs, O-RUs, and UEs. Hence, a BS can be implemented by installing the O-RU srsRAN stack on a USRP and the O-CU/O-DU on a general-purpose server. In the same way, UEs are implemented by installing a UE stack on a USRP.

The Colosseum testbed, which is part of a broader initiative named OpenRAN Gym [13], is an example of a wireless network emulator that employs 128 USRP connected through a Massive Channel Emulator (MCHEM) [13]. This MCHEM allows the emulation of different channel aspects, such as path loss, fading, and user mobility. The RAN used in Colosseum employs the solutions provided by the srsRAN project.

IV. Publicly available datasets used in O-RAN

This section describes the most representative public datasets used in O-RAN papers. We select those datasets which the papers explicitly mention their applicability to O-RAN. However, some of these datasets were not employed in deep learning yet. We consider them in this survey since they can help design novel deep learning models.

We have downloaded and manipulated all the datasets surveyed in this paper to explore some details that were not explicitly described in their documentation. Consequently, our work act as a complement to their original documentation. However, it is advised to carefully read their documentation and associated papers before using the datasets. Furthermore, to use these datasets, please refer to the conditions available on each dataset link.

A. Colosseum O-RAN COMMAG Dataset

This dataset, used in [4], is available on GitHub [14], being a result of a 5G network emulation on Colosseum (Section III). The emulation considers real Base Station (BS) locations in an area of 0.11 km^2 in Rome, Italy. This emulation employs 40 UEs and four BSs. A network slicing scenario is considered in which each BS has three different slices, and the UEs are statically assigned to a given slice.

The UEs generate traffic using the Colosseum Traffic Generator (TGEN) [13] following three traffic types. The first one is enhanced mobile broadband (eMBB), in which the UE generates constant bitrate traffic of 1 Mbps. The other one is machine-type communications (MTC), in which the UE generates 125-byte packets following a Poisson distribution with a rate of 30 pkt/s (packets per second). Finally, the ultra-reliable low-latency communications (URLLC) type considers that the UE generates a traffic of 125-byte packets using a Poisson distribution with a 10 pkt/s rate.

Each BS serves ten UEs on the emulation. Also, a UE is fixedly assigned to a given BS and a slice using two possible assignments. The UEs from a BS are distributed randomly among the three slices for the UE assignment called `slice_mixed`. In contrast, the assignment called `slice_traffic` considers that each slice handles one different traffic type (i.e., eMBB, MTC, and URLLC). Hence, the traffic type of a given UE is known, and the UE is assigned to a given slice depending on its traffic type, which is fixed throughout the emulation. For each BS, there are three UEs generating eMBB traffic, three UEs generating MTC traffic, and four UEs generating URLLC traffic[3]. The dataset's `README.md` file specifies which UEs follow each traffic type.

The dataset is organized using one directory tree for the `slice_mixed` emulation and another tree for the `slice_traffic` one. Figure 1 shows the `slice_traffic` tree organization. The tree for the `slice_mixed`, omitted to conciseness, is similar to the one in Figure 1. There are four different emulation scenarios for `slice_traffic` according to Figure 1. The `rome_slow_close` considers that UEs move at an average speed of 3m/s within 20 m of each BS. The other ones consider that UEs are static, differing according to the distance between UEs and their associated BS. In `rome_static_close`, the UEs are within 20 m of their BS. In `rome_static_medium` and `rome_static_far`, this distance is 50 m and 100 m, respectively. The `slice_mixed` emulation has only `rome_slow_close` and `rome_static_close` scenarios.

The emulation uses different initial configurations of Resource Block Group (RBG) allocations and scheduling policies. A Resource Block Group (RBG) is a group of contiguous PRBs (Physical Resource Blocks). In its turn, the PRB (Physical Resource Block) is the smallest allocation unit of the radio link, consisting of subcarriers allocated at a specific frequency and during a determined time interval. Hence, the RBG allocation defines the amount of radio resources allocated to a given slice. The emulation considers that one RBG has two PRBs. The scheduling policy defines how the

[1]https://www.ettus.com/

[2]https://www.srsran.com/

[3]The paper in [4] describes a slightly different UE distribution for the traffic types. However, after analyzing the data, we can state that the distribution reported here is the actual one used by the public dataset.

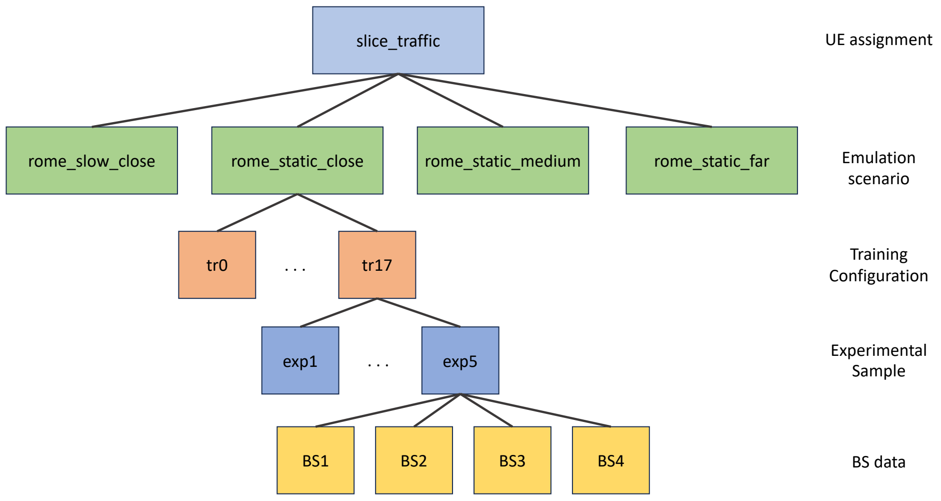

Fig. 1. Directory tree for the Colosseum O-RAN COMMAG Dataset.

available RBGs are scheduled between UEs within a network slice. The emulation considers three different policies: Round-robin (RR), Waterfilling (WF), and Proportionally fair (PF).

The emulation enables a diverse model training with 18 different initial combinations of RBG allocation and scheduling policies, named tr0 to tr17, as shown in Figure 1 for the `rome_static_close` scenario. For example, for tr0, the eMBB slice has one RBG and employs the PF policy, the MTC slice has two RBGs and the RR policy, and the URLLC slice has four RBGs and the PF policy. The RBG allocation and scheduling policy for each configuration is specified on the dataset's `README.md` file. Although Figure 1 details only the `rome_static_close` case, the 18 training configurations are employed in all scenarios.

The emulation has 480 seconds. The initial RBG allocation, defined in the training configuration, is kept during the first 30 seconds. After that, the RBG allocation varies while the policy is kept fixed. This variation is detailed on the dataset's `README.md` file. The RBG variation allows the evaluation of algorithms that dynamically choose the scheduling policy, as performed in [4].

For each training configuration, the experiment is repeated five times. Hence, as shown in Figure 1 for the case of tr17 in the `rome_static_close` scenario, each configuration has five directories, named from `exp1` to `exp5`.

An experimental sample (i.e., `exp1` to `exp5`) has four directories to describe the performance metrics for each BS. Figure 2 shows the directory tree that describes the BS data, using BS1 as an example. The directory has a CSV file for each UE connected to the BS, one CSV file related to the BS itself, and a directory that stores CSV files for each UE describing slice allocation metrics. The CSV files can be easily read by libraries such as pandas[4].

The columns of the UE or BS CSV files describe metrics

[4]https://pandas.pydata.org/

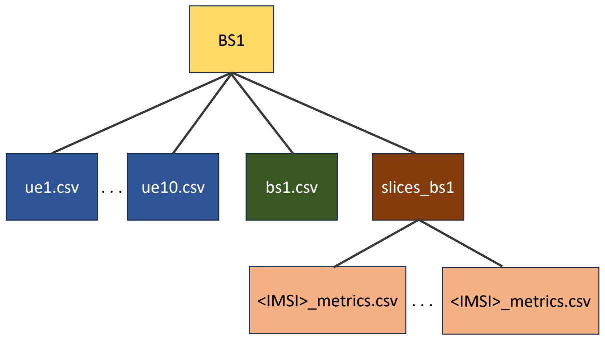

Fig. 2. BS data for the Colosseum O-RAN COMMAG Dataset.

reported by the srsRAN tool. Each row contains the metrics for a given timestamp. The main columns of a UE CSV file (e.g., the ue10.csv file in Figure 2) are described in Table I. To conciseness, this table and the following ones have names with "dl(ul)" indicating that there is a value for the downlink and the uplink. A detailed description of each metric is available on srsRAN documentation. Table II describes the columns of the BS CSV file (e.g., the bs1.csv file in Figure 2).

The slice metrics CSV files for each UE are grouped in a directory (e.g., `slices_bs1` in Figure 2). The name of each file has the UE's IMSI (International Mobile Subscriber Identity), a unique identifier for a UE. These files are the ones employed in [4] to train the agents that choose the slice scheduling policy. The main columns of the CSV file for slice metrics are described in Table III. Note that there are metrics in common with the UE's CSV files.

The work in [4] uses the described dataset to propose xApps based on Deep Reinforcement Learning (DRL) that dynamically chooses the best scheduling policy among RR, WF, and PF. The algorithm is trained using the dataset, which has different performance metrics according to fixed

TABLE I
MAIN COLUMNS OF THE UE CSV FILE IN COLOSSEUM O-RAN COMMAG DATASET.

Name	Description
time	Timestamp in milliseconds
cc	Component Carrier (LTE)
pci	Physical Cell Identifier
earfcn	Frequency Channel
rsrp	Reference Signal Receive Power (dBm)
pl	Pathloss (dB)
cfo	Carrier Frequency Offset (Hz)
dl(ul)_mcs	Downlink (uplink) - Modulation and coding scheme
dl(ul)_brate	Downlink (uplink) - Bitrate (bits/sec)
dl(ul)_bler	Downlink (uplink) - Block error rate
dl_snr	Downlink - Signal-to-Noise (SNR) Ratio (dB)
ul_buff	Uplink - Buffer status (i.e., enqueued bytes)

TABLE II
COLUMNS OF THE BS CSV FILE IN COLOSSEUM O-RAN COMMAG DATASET.

Name	Description
time	Timestamp in UNIX time (milliseconds)
nof_ue	Number of UEs associated with the BS
dl(ul)_brate	Downlink (uplink) - Bitrate (bits/sec)

scheduling policies, as described in Table III. The dataset link also contains the algorithm implementation.

B. Colosseum ColO-RAN Dataset

This dataset, used in [15], is available on GitHub [16]. The emulation uses Colosseum and the same BS environment in Rome, as described in Section IV-A. However, it considers only the `slice_traffic` assignment (i.e., the UEs are assigned to one of the three slices based on their traffic types). Also, the dataset uses only the `rome_static_medium` scenario, in which static UEs are distributed within 50 m of their BS. The emulation employs 42 UEs and seven BSs.

Figure 3 shows the directory tree for this dataset. Note that the structure is similar to that of Section IV-A. However, the ColO-RAN dataset has only one emulation scenario (i.e., `rome_static_medium`). Also, instead of varying the scheduling policy in the training configuration, there is one directory for each scheduling policy. Hence, in `sched_0`, all

TABLE III
MAIN COLUMNS OF THE CSV FILE FOR SLICE METRICS OF EACH UE IN COLOSSEUM O-RAN COMMAG DATASET.

Name	Description
time	Timestamp in UNIX time (milliseconds)
slice_id	Slice assigned to this UE (eMBB=0, MTC=1, and URLLC=2)
dl_buffer [bytes]	Downlink - Buffer size (i.e., transmission queue size)
tx_brate downlink [Mbps]	Downlink - Transmission bit rate between the BS and the UE
sum_requested_prbs	Number of PRBs requested by the UE
sum_granted_prbs	Number of PRBs granted to the UE
scheduling_policy	Slice's initial scheduling policy (RR=0, WF=1, and PF=2)
slice_prb	Number of PRBs initially attributed to the slice

slices use a RR policy, while in `sched_1` and `sched_2` they use WF and PF, respectively. Consequently, the training configuration, named from tr0 to tr27, only varies the number of RBGs allocated per slice. This variation is detailed on the dataset's `README.md` file.

In the same way, as in Section IV-A, each training configuration has five experimental samples. The directory of a given experimental sample has one directory for each considered BS. The BS directory has the same structure described before in Figure 2, with the same columns for the CSV files, described in Tables I, II, and III.

The work in [15] uses the described dataset to propose xApps that use DRL to select the scheduling and slicing policies. The scheduling policies are the same as described before in Section IV-A, and thus, the xApp must choose between RR, WF, and PF. The slicing policy is the number of PRBs for each slice.

C. Computing, Energy, and Application Dataset

The dataset described in [17] reports computing and energy metrics of a vRAN (virtual RAN) testbed. This testbed employs srsRAN to implement UEs and BSs on USRP B210 boards. The UE software runs on commodity laptops. The BS software functions run on a computing pool, implemented as containers executing as vBSs (virtual Base Stations). This computing pool uses off-the-shelf hardware, such as servers equipped with Intel i7-7700K CPUs with eight cores. Also, an edge application server is connected to the BS through the mobile core and used to generate traffic or run an object recognition application. The dataset is available on IEEE Dataport™ [18]. This repository is not widely open as GitHub but can be accessed for free by IEEE Society Members or through a paid institutional or individual subscription.

Figure 4 shows the dataset's directory tree. The `energy_datasets` has two CSV files reporting energy consumption metrics on vBSs. The `dataset_ul.csv` file results from an experiment with only uplink traffic, while the `dataset_dlul.csv` file considers uplink and downlink traffic. Each row of these files represents the metrics obtained during one minute for a given configuration. The first columns of these files represent the configuration parameters of the testbed, while the other ones represent the metrics. Table IV shows the main configuration parameters, and Table V shows the main metrics. These metrics were obtained by software or hardware. The software measurements estimate CPU consumption using the Running Average Power Limit (RAPL) functionality via Linux `turbostat` tool. Also, they employ a digital power meter to capture the energy consumption of the entire server and the radio head.

The `computing_datasets` in Figure 4 reports the server's CPU utilization using different configuration parameters and numbers of containers hosting vBS functions. The CSV files are grouped in two directories. The `datasets_unpin` directory has five CSV files named `dataset_x.csv`, in which `x` is the number of concurrent vBS container instances. In this experiment, the default Linux

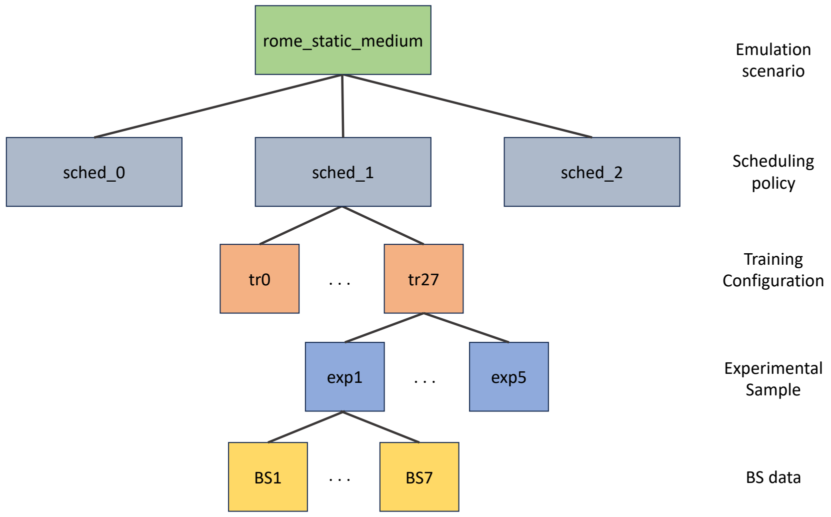

Fig. 3. Directory tree for the Colosseum ColO-RAN Dataset.

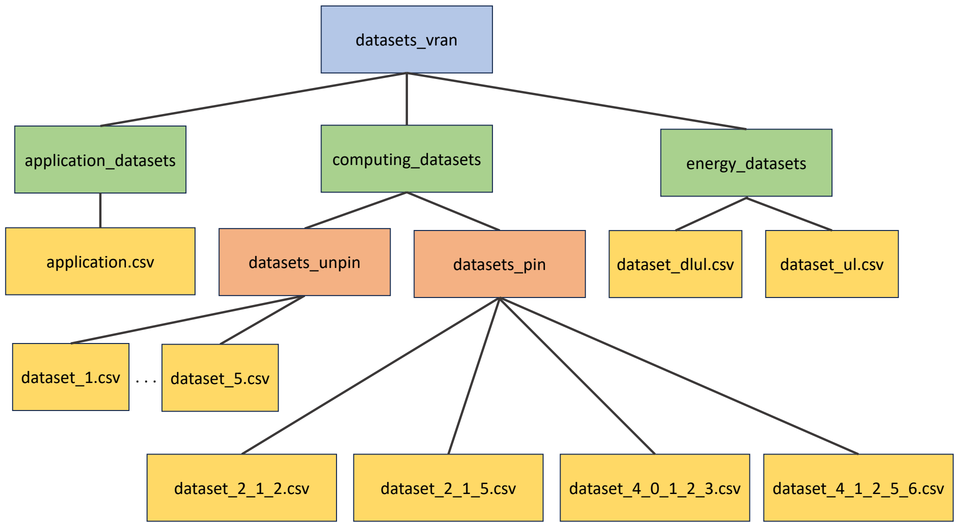

Fig. 4. Directory tree for the Computing, Energy, and Application Dataset.

TABLE IV
Configurations of the `energy_datasets` CSV files in the Computing, Energy, and Application Dataset.

Name	Description
date	Timestamp using ISO 8601 date format
BW	Bandwidth in terms of PRBs
traffic_load_dl(ul)	Downlink (uplink) traffic load
txgain_dl(ul)	Downlink (uplink) transmission gain configured on the USRP board to evaluate different SNR patterns
selected_mcs_dl(ul)	Modulation and coding scheme (MCS) used in the downlink (uplink)
selected_airtime_dl(ul)	Airtime used in the downlink (uplink), configured using the maximum number of PRBs per subframe

TABLE V
Main energy consumption metrics in the Computing, Energy, and Application Dataset.

Name	Description
thr_dl(ul)	Average downlink (uplink) throughput
pm_power	Average power consumed by the vBS measured using the power meter
pm_var(median)	Variance (or median) of the power consumed by the vBS measured using the power meter
rapl_power	Average power consumed by the vBS's CPU
rapl_var	Variance of the power consumed by the vBS's CPU

CPU scheduling policy chooses which CPU core runs a given instance. On the hand, the `datasets_pin` refers to experiments in which each container is pinned to a specific CPU core.

For the CPU pin case, the `dataset_2_1_2.csv` reports the results of an experiment with two vBS instances. One instance is pinned to the CPU core #1, and the other to the CPU core #2. In this case, the instances share the same L3 cache but use different L1 and L2 caches [17]. For the dataset_2_1_5.csv file, one instance is pinned to the CPU core #1, and the other one to the CPU core #5. In this configuration, the instances share all CPU caches since core #5 is the hyperthread of core #1 (i.e., both cores are on the same physical core). This same reasoning applies to the dataset_4_0_1_2_3.csv file, which consists of four instances pinned to cores #0, #1, #2, and #3, respectively. In this case, the cores are on different physical cores, sharing only the L3 cache. Finally, for the dataset_4_1_2_5_6.csv file, the instances on cores #1 and #5 share all the caches. This same pattern occurs among instances on cores #2 and #6. All the instances share the L3 cache. Table VI shows the configuration parameters for the computing_datasets CSV files, while Table VII describes the metrics.

TABLE VI
CONFIGURATIONS OF THE COMPUTING_DATASETS CSV FILES IN THE COMPUTING, ENERGY, AND APPLICATION DATASET.

Name	Description
mcs_dl(ul)_i	Modulation and coding scheme in the downlink (uplink) for vBS i
dl(ul)_kbps_i	Downlink (uplink) load for vBS i
cpu_set_i	Set of CPU cores that can be used by vBS i

TABLE VII
COMPUTING METRICS IN THE COMPUTING, ENERGY, AND APPLICATION DATASET.

Name	Description
cpu_j	Average utilization of CPU core j, between 0 and 1
explode	Indicates if the experiment should be discarded due to running problems

Finally, the application_datasets in Figure 4 has a CSV file with metrics related to an object recognition application in which a UE sends an image to an edge server using its vBS. The server thus performs object recognition tasks in this image and sends the result back to the UE through the vBS. The metrics available in the dataset include power consumption and CPU utilization measurements and application-specific ones. Due to conciseness, we omit their description, which can be found in [17] or in the dataset link [18].

The work in [17] discusses some applications of the proposed dataset. For example, they show that the vBS CPU utilization varies according to the radio link quality. Hence, a deep learning model can choose the CPU resource allocation according to the SNR sensed in the link.

D. ChARM Dataset

The dataset used in [19] consists of spectrum data collected using the Colosseum testbed. The data consists of I/Q samples collected from LTE and WiFi traffic and background noise. An I/Q sample represents an instantaneous snapshot of the modulated signal, containing both the In-phase (I) and Quadrature (Q) components. The I value represents the amplitude of the baseband signal in-phase with the I carrier (0-degree phase). The Q value represents the amplitude of the baseband signal in-phase with the Q carrier (90-degree phase).

The data was collected at the Colosseum testbed, using the central frequency of 5.2 GHz and a 20MHz bandwidth. For data collection, GNURadio[5] and a USRP x310 were employed. WiFi data generation was performed using OpenWiFi and two Xilinx zc706 boards, acting as an access point and a client. LTE data generation utilized srsRAN and two USRP x310 boards, acting as a BS and a UE. The dataset is available at the Northeastern University Digital Repository Service (DRS) [20]. It consists of nine files:

- LTE_ZT - LTE traces with idle traffic (i.e., considering only control traffic from the BS);
- LTE_1M - LTE traces with a 1Mbps flow between the BS and the UE, generated using iperf3;
- LTE_FLOOD - LTE traces with ping flooding with 1KB packets between the BS and the UE. These traces represent bursty traffic with high throughput;
- LTE_PINGs300 - LTE traces with ping communication with 300-byte packets between the BS and the UE. These traces represent bursty traffic with low throughput;
- WIFI_ZT, WIFI_1M, WIFI_FLOOD, and WIFI_PINGs300 - WiFi traces with the same traffic pattern as their LTE counterparts.
- CLEAR - Background noise, collected when there is no LTE or WiFi communication.

Each file described before is a binary one containing a sequence of 32-bit floating-point number pairs in the little-endian order. Each pair is an I/Q sample, starting from a 32-bit floating number representing the I value followed by a 32-bit floating number representing the corresponding Q value. We have provided a code[6] in GitHub exemplifying how to read the files. This code is a simplified version of the implementation provided by the authors in [19], which can also be found on GitHub[7].

This dataset is employed in [19] to train a residual network (RN) and a convolutional neural network (CNN) that classifies the traffic type when receiving a stream of I/Q samples. The neural network can classify the traffic in Clear, WiFi, LTE, and Unknown (i.e., when the classification has low confidence). This classification is used in the proposed Channel-Aware Reactive Mechanism (ChARM) system, which allows WiFi and LTE traffic to coexist in O-RAN infrastructures.

E. Ultra Dense Indoor mMIMO CSI Dataset

The dataset used in [21] is available on IEEE Dataport™ [22]. Differently from the dataset surveyed in Section IV-C, which is also on IEEE Dataport™, this dataset is available in an open-access manner.

The dataset consists of Channel State Information (CSI) samples collected in a room with 64 antennas deployed using USRPs. CSI represents information about the wireless communication channel between a transmitter and a receiver. For

[5]https://www.gnuradio.org/
[6]https://github.com/GTA-UFRJ/useorandatasets/blob/main/readCharm.py
[7]https://github.com/lucabaldesi/charm_trainer/blob/master/read_IQ.py

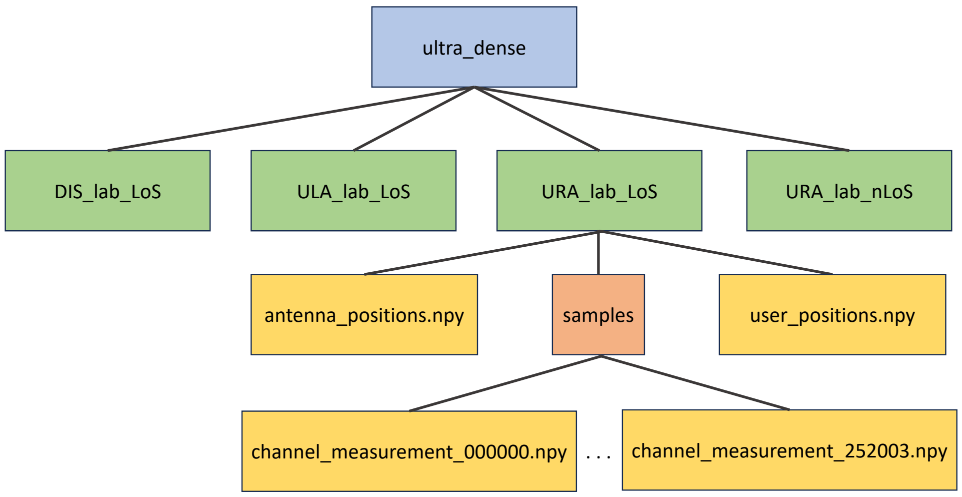

Fig. 5. Directory tree for the Ultra Dense Indoor mMIMO CSI Dataset.

this dataset, a CSI sample consists of complex-valued matrices that span 64 rows (representing the number of BS antennas) and 100 columns (representing the number of subcarriers). Hence, the CSI matrix, along with the channel noise, can be used to estimate the receiving signal given the transmitted one [23].

Figure 5 shows the dataset's directory structure. There are four different directories storing measurements for different antenna array topologies. The dataset link [22] has a figure showing these topologies, described next:

- `URA_lab_LoS` - Uniform Rectangular Array (URA) with Line of Sight (LoS). In this case, the antennas are arranged in an 8x8 grid without obstacles on the communication paths. The grid rows (i.e., a group of 8 antennas) are in the same (x,y) coordinate. The rows' positions differ in their height. Hence, considering an (x,y,z) coordinate, each 8-antenna group has a different value on the Z-axis.
- `URA_lab_nLoS` - The same array as in `URA_lab_LoS` but using a metal blocker to create an obstacle on the communication paths.
- `ULA_lab_LoS` - Uniform Linear Array (ULA) with LoS. In this case, the 64 antennas are arranged in a linear array without obstacles. Hence, considering an (x,y,z) coordinate, the antennas differ only regarding their X-axis.
- `DIS_lab_LoS` - Distributed Antenna System (DIS) with LoS. In this case, the antennas are grouped in linear arrays of eight antennas. These groups are distributed across the room, and there are no obstacles. All the groups are at the same height.

Each one of the described directories has the same structure presented in Figure 5 for the `ULA_lab_LoS` one. All the data files are pickled arrays from NumPy, which can be easily accessed by the `numpy.load()` method[8].

[8]https://numpy.org/doc/stable/reference/generated/numpy.load.html

The `antenna_positions.npy` file stores the antenna placement across the room, considering their corresponding topology described before. The file has an array with shape (64,3), in which each row corresponds to a BS antenna, and the columns are `float64` values representing their x, y, and z coordinates in millimeters. Likewise, the `antenna_positions.npy` file stores the UE placement across the room. The experiment involves moving the UE using a CNC (Computer Numerical Control) XY-table, a mechanical platform that allows movement along the X and Y axes. This movement occurs by zigzagging across a nine-squared meter area in steps of 5mm, resulting in 252,004 samples. Accordingly, the `antenna_positions.npy` file stores an array with shape (252004, 3), in which each row is a sample (i.e., UE position), and the columns are `int64` values representing their x, y, and z coordinates in millimeters.

As shown in Figure 5, each topology directory has a subdirectory named `samples`, which stores a NumPy pickled array for each position sample. Hence, each `samples` subdirectory has 252,004 files. Each file has an array with shape (64, 100) representing the CSI matrix. Each row of this array corresponds to an antenna, and a column is a complex number (i.e., a `complex128` value) representing a given subcarrier. The element of the CSI matrix is thus a complex number that can be used to obtain the amplitude gain (i.e., indicating signal strength) and phase gain (i.e., representing phase shift) for a specific subcarrier and antenna combination.

The work in [21] uses this dataset to analyze the performance of cell-free deployments in the context of O-RAN. Cell-free networks consist of wireless infrastructures where multiple BS can serve a given UE. The architecture of O-RAN facilitates the implementation of cell-free networks since, for example, different O-RUs can cooperate if the same O-DU serves them.

F. O-RAN Software Community Dataset

The work in [24] uses a dataset and an anomaly detection xApp provided on GitHub by the O-RAN Software Community[9] [25]. The dataset consists of a single CSV file containing throughput values sampled at 10 ms intervals in an environment with 20 UEs and four BSs. Each row represents a sample for a given UE. The timestamp and the downlink throughput values are reported in the column `measTimeStampRf` and `DRB.UEThpDl`, respectively. As of the writing of this article, there were no further details available about this dataset, despite a brief description in [24]. Since the focus of the code provided by the O-RAN Software Community is the xApp itself, there is no information on how the measurement was performed or if it consists of actual experiments or simulations. Hence, in this paper, we are not able to provide more details about this dataset.

V. Conclusions

The application of deep learning models in O-RAN is a reality. However, building testbeds for radio access networks is challenging, and deep learning heavily relies on extensive data. Our paper has surveyed datasets already employed by O-RAN mechanisms, providing descriptive details to enhance their accessibility for research.

Notably, most datasets described in this work are currently confined to their original research papers, limiting their broader utilization. In addition, some deep learning models might need more data than that available in the existing datasets. However, this paper's insights aim to pave the way for broader adoption and enrichment of these datasets by researchers and practitioners in the deep learning O-RAN community.

Acknowledgement

This study was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, CNPq, PR2/UFRJ, FAPERJ Grants E-26/010.002174/2019, and E-26/201.300/2021, and FAPESP Grant 15/24494-8.

References

[1] M. Moussaoui, E. Bertin, and N. Crespi, "Telecom business models for beyond 5G and 6G networks: Towards disaggregation?" in *1st International Conference on 6G Networking (6GNet)*, 2022, pp. 1–8.

[2] M. A. Lopez, G. N. N. Barbosa, and D. M. F. Mattos, "New barriers on 6G networking: An exploratory study on the security, privacy and opportunities for aerial networks," in *1st International Conference on 6G Networking (6GNet)*, 2022, pp. 1–6.

[3] M. Polese, L. Bonati, S. D'Oro, S. Basagni, 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.

[4] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and Learning in O-RAN for Data-Driven NextG Cellular Networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 21–27, 2021.

[5] B. Brik, K. Boutiba, and A. Ksentini, "Deep Learning for B5G Open Radio Access Network: Evolution, Survey, Case Studies, and Challenges," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 228–250, 2022.

[6] H. Hojeij, M. Sharara, S. Hoteit, and V. Vèque, "Dynamic placement of O-CU and O-DU functionalities in open-ran architecture," in *IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2023.

[7] F. Rezazadeh, L. Zanzi, F. Devoti, H. Chergui, X. Costa-Pérez, and C. Verikoukis, "On the Specialization of FDRL Agents for Scalable and Distributed 6G RAN Slicing Orchestration," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 3, pp. 3473–3487, 2023.

[8] A. Arnaz, J. Lipman, M. Abolhasan, and M. Hiltunen, "Toward Integrating Intelligence and Programmability in Open Radio Access Networks: A Comprehensive Survey," *IEEE Access*, vol. 10, pp. 67 747–67 770, 2022.

[9] 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.

[10] C.-W. Huang, C.-T. Chiang, and Q. Li, "A study of deep learning networks on mobile traffic forecasting," in *IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017, pp. 1–6.

[11] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[12] M. Amini, R. Stanica, and C. Rosenberg, "Where are the (cellular) data?" *ACM Computing Surveys*, vol. 56, no. 2, 2023.

[13] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "OpenRAN Gym: AI/ML development, data Collection, and Testing for O-RAN on PAWR Platforms," *Computer Networks*, vol. 220, p. 109502, 2023.

[14] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Colosseum O-RAN COMMAG dataset," Available: https://github.com/wineslab/colosseum-oran-commag-dataset, 2022.

[15] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing Machine Learning-Based xApps for Open RAN Closed-Loop Control on Programmable Experimental Platforms," *IEEE Transactions on Mobile Computing*, 2022.

[16] ——, "Colosseum O-RAN ColORAN dataset," Available: https://github.com/wineslab/colosseum-oran-coloran-dataset, 2022.

[17] J. X. Salvat Lozano, J. A. Ayala-Romero, L. Zanzi, A. Garcia-Saavedra, and X. Costa-Perez, "Open radio access networks O-RAN experimentation platform: Design and datasets," *IEEE Communications Magazine*, pp. 1–7, 2023, accepted for publication.

[18] ——, "O-RAN experimental evaluation datasets," Available: https://dx.doi.org/10.21227/64s5-q431, 2022.

[19] L. Baldesi, F. Restuccia, and T. Melodia, "ChARM: NextG Spectrum Sharing through Data-Driven Real-Time O-RAN Dynamic Control," in *IEEE Conference on Computer Communications (INFOCOM)*, 2022, pp. 240–249.

[20] ——, "ChARM (Channel-Aware Reactive Mechanism) dataset," Available: http://hdl.handle.net/2047/D20423481, 2021.

[21] V. Ranjbar, A. Girycki, M. A. Rahman, S. Pollin, M. Moonen, and E. Vinogradov, "Cell-free mMIMO support in the O-RAN architecture: A PHY layer perspective for 5G and beyond networks," *IEEE Communications Standards Magazine*, vol. 6, no. 1, pp. 28–34, 2022.

[22] S. De Bast and S. Pollin, "Ultra dense indoor MaMIMO CSI dataset," Available: https://dx.doi.org/10.21227/nr6k-8r78, 2021.

[23] Y. Ma, G. Zhou, and S. Wang, "WiFi sensing with channel state information: A survey," *ACM Computing Surveys*, vol. 52, no. 3, jun 2019.

[24] V.-Q. Pham, H.-T. Thieu, A. Kak, and N. Choi, "HexRIC: Building a better near-real time network controller for the Open RAN ecosystem," in *The International Workshop on Mobile Computing Systems and Applications (HotMobile '23)*, 2023, p. 15–21.

[25] O-RAN Software Community (SC), "Anomaly detection," Available: https://github.com/o-ran-sc/ric-app-ad, 2022.

[9]The dataset is available inside the xApp's source code: https://github.com/o-ran-sc/ric-app-ad/blob/master/src/ue.csv