SoftwareX 21 (2023) 101320

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Exploiting Simu5G for generating datasets for training and testing AI models for 5G/6G network applications

Giovanni Nardini [a,b,*], Alessandro Noferi [a], Pietro Ducange [a], Giovanni Stea [a]

[a] Dipartimento di Ingegneria dell'Informazione, University of Pisa, Largo L. Lazzarino 1, 56122 Pisa, Italy
[b] Center for Logistic Systems, University of Pisa, Via dei Pensieri 60, 57142, Livorno, Italy

ARTICLE INFO

Article history:
Received 20 September 2022
Received in revised form 16 January 2023
Accepted 19 January 2023

Keywords:
simu5G
Artificial intelligence
Network simulation
Dataset

ABSTRACT

Researchers working on Artificial Intelligence (AI) need suitable datasets for training and testing their models. When it comes to applications running through a mobile network, these datasets are difficult to obtain, because network operators are hardly willing to expose their network data or to open their network to experimentation. In this paper we show how Simu5G, a popular 5G network simulator based on OMNeT++, can be used to circumvent this problem: it allows users to log data at arbitrary spatial and temporal resolution, belonging to every network layer — from the application to the physical one.

Code metadata

Current code version	*v1.2.1*
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00294, https://github.com/Unipisa/Simu5G/releases/tag/dataset_generator_software-x
Permanent link to reproducible capsule	
Legal code license	*GNU Lesser General Public License V. 3*
Code versioning system used	*git*
Software code languages, tools and services used	*C++, Network Description (NED) language, few python routines.*
Compilation requirements, operating environments and dependencies	*Requires OMNeT++*
If available, link to developer documentation/manual	http://simu5g.org, https://github.com/Unipisa/Simu5G
Support email for questions	giovanni.nardini@unipi.it

1. Motivation and significance

Research on Artificial Intelligence (AI) for networking has seen an enormous upsurge in the last years. AI solutions are now being envisaged to solve complex networking problems, such as optimizing the network configuration and predicting user Quality of Experience (QoE) [1]. Parallel to this, *networked* paradigms for AI algorithms, such as Federated Learning [2], are emerging: their performance (e.g., speed of convergence) depends on network Quality of Service (QoS), and networks themselves have to manage new workload for supporting such new paradigms. This double interplay – i.e., AI *for* networks, on one hand, and *networked* AI, on the other – has at its core cellular access (CA), i.e., 4G, 5G and Beyond-5G (B5G) [3]. Indeed, CA ensures the expected reliability and ubiquitous diffusion, and supports complex communication/computation services via Multi-access Edge Computing (MEC) [4].

Suitable datasets for training and testing AI models for specifics applications may be hard to obtain. For example, an AI model for predicting user QoE needs to correlate data coming from different network (sub)layers and the position data on the mobile users, at the relevant time and space resolution, with the actual level of QoE perceived by users. Network infrastructure providers have their own logging procedures. However, they are (understandably) not always willing to share them, due to a plethora of reasons (e.g., competitive advantage, or the extra work required for GDPR compliance). Moreover, these data may not fit the requirements for the training algorithm: e.g., they may not log some relevant information, or log it at the wrong time/space resolution, or in suboptimal network conditions (e.g., too lightly loaded cells). The same problem occurs – possibly exacerbated – when Federated Learning of AI models is required. In this case,

* Corresponding author at: Dipartimento di Ingegneria dell'Informazione, University of Pisa, Largo L. Lazzarino 1, 56122 Pisa, Italy.
E-mail address: giovanni.nardini@unipi.it (Giovanni Nardini).

https://doi.org/10.1016/j.softx.2023.101320

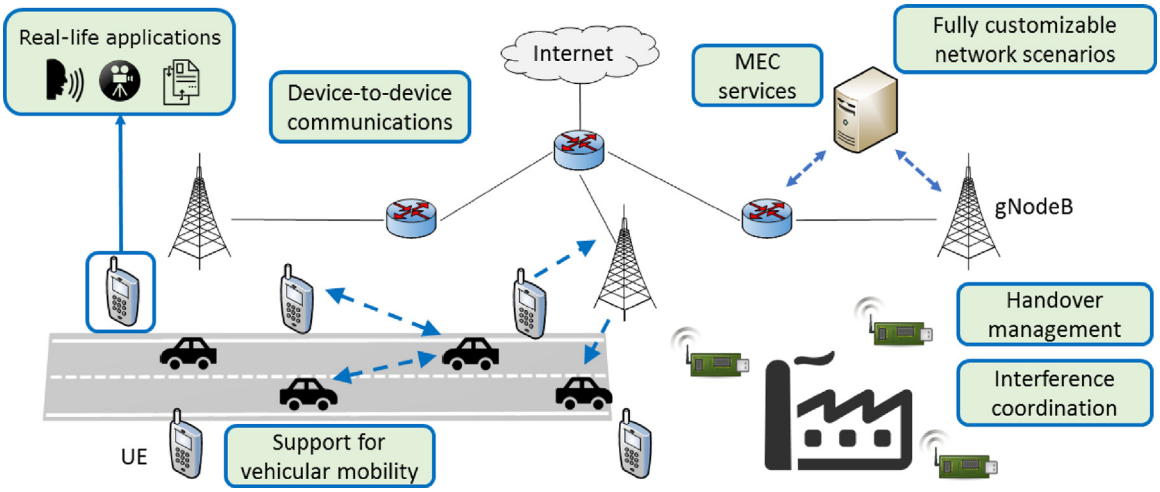

Fig. 1. Overview of Simu5G's functionalities, which cover both application- and network-level modeling of end-to-end applications, as well as flexible configuration of users mobility.

datasets should in fact include input data to all the federated entities (typically, users of cellular networks).

In this paper, we discuss how these issues can be solved by using Simu5G, a popular 4G/5G cellular network simulation library [5–7]. Simu5G is an *end-to-end* simulator for evaluating how the network affects application-layer metrics, and how applications impact network performance. It allows users to generate arbitrary network scenarios, which include user mobility, handover, real applications, and to set arbitrary *probes* in the code, to measure what they need at the relevant time/space granularity, with full control over experimental conditions. Simu5G can therefore be used to generate flexible datasets for training and testing AI models. We describe how to set probes in Simu5G and how to generate datasets, with reference to a real-life example for tele-operated driving. AI models learn their structure, namely their parameters, from available data. Accordingly, the higher the capability of a simulator to generate realistic data, the better the AI models will be able to act in real world applications. Simu5G's physical layer reporting has been validated according to 3GPP guidelines [5], and its MEC model has been validated in [6]. While this does not constitute any a-priori guarantee, it is however encouraging.

Simu5G is not the only software that simulates 5G networks. Authors of [8,9] discuss *physical-layer simulators* for evaluating physical-layer design (e.g., antenna performance, transmission schemes, spectral efficiency). These simulators often lack upper network protocols and cannot support real applications. Other *end-to-end* simulators include some – but not all – Simu5G's features [10,11]. For instance, 5G-LENA [10] is an ns3 library [12] that, to the best of our knowledge, lacks dual connectivity, network-controlled device-to-device communications, ETSI MEC modeling, and real-time capabilities.

2. Software description

Simu5G is a discrete-event simulation *library* for 4G/5G New Radio networks based on OMNeT++ [13]. As shown in Fig. 1, Simu5G is interoperable with all the OMNeT++-based libraries, e.g., the INET library, which includes a wealth of TCP/IP network elements [14], libraries for vehicular mobility (e.g., using [15]), etc. Moreover, Simu5G also models the ETSI MEC standard [4, 16]. Users can therefore setup scenarios where user applications instantiate MEC applications through a 5G network, and MEC applications use the services provided by the underlying 5G network (e.g., the Radio Network Interface service). Applications can be modeled within the simulator, or they can be external applications *interfaced* with it. Simu5G can also run in real time [7]: packets from an external application are injected into Simu5G, undergo coherent forwarding treatment (routing, delay, losses, etc.) and come out at the other end, to be conveyed to their external destination.

2.1. Software architecture

Simu5G provides a set of modules that implement the main entities of 5G New-Radio (NR) networks, such as Base Stations (BSs) and User Equipments (UEs). Following the OMNeT++ philosophy, each entity is described by a Network Description (NED) file, a declarative language that exploits inheritance and interfaces, fully convertible into XML. It defines the parameters of the module and the submodules that compose its internal architecture, as well as the gates and the connections that allows it to exchange messages with other modules. Parameters to be used in a simulation instance can be configured via an initialization (INI) file. Fig. 2 shows the architecture of the *gNodeB* and the *NrUe* modules. They both include a NR Network Interface Card (NIC) submodule, which in turn is composed of submodules representing the different layers of the NR protocol stack, from PDCP to the physical layer. The behavior of each submodule is specified by a C++ class, which defines the actions that the module performs to handle events, such as the reception of messages from another module or the expiration of a self-scheduled timer. Indeed, data-plane network packets are implemented as messages that flow through the submodules of a BS or a UE, where each layer applies its 3GPP-compliant processing before passing the messages to the downstream/upstream layer (or sending it to another entity, e.g., over the radio channel). Many NR features are also supported, such as carrier aggregation, multiple numerologies, frequency- and (flexible) time-division duplexing.

Fig. 3 reports a portion of the class diagram describing the entities performing physical-layer operations. *LtePhyBase* is the class providing the base functionalities for a node, which are specialized by specific classes at BS and UE side. Note that the same inheritance mechanism is employed for all the submodules of the NR NIC shown in Fig. 2. *LtePhyBase* references one or more channel models, i.e. one for each component carrier supported by the node. The channel model is implemented by one of the classes implementing the *LteChannelModel* interface. Each channel model is associated to one *ComponentCarrier* class, which includes the parameters of the component carrier, such as carrier frequency, numerology index, etc. An instance of *ComponentCarrier* can be used by zero or more nodes in the simulation.

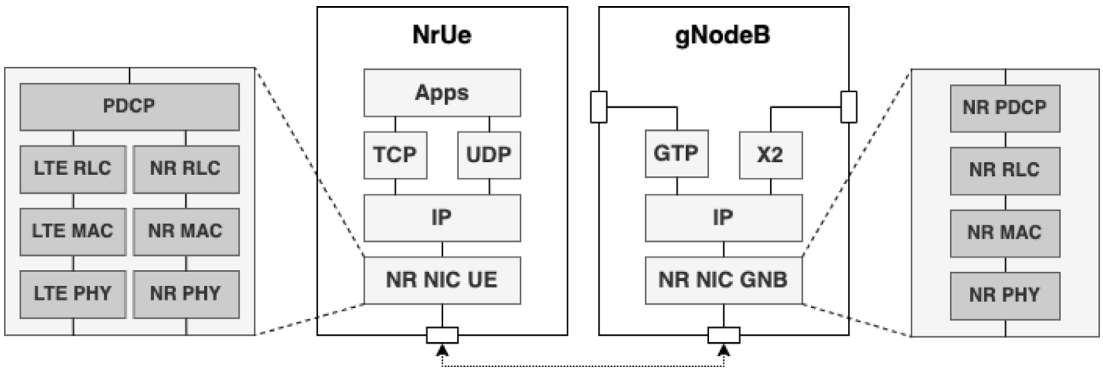

Fig. 2. High-level architecture of Simu5G's main modules, namely the NrUe and the gNodeB modules, with special focus on the internal modeling of the NR NIC, which includes all the NR protocol layers.

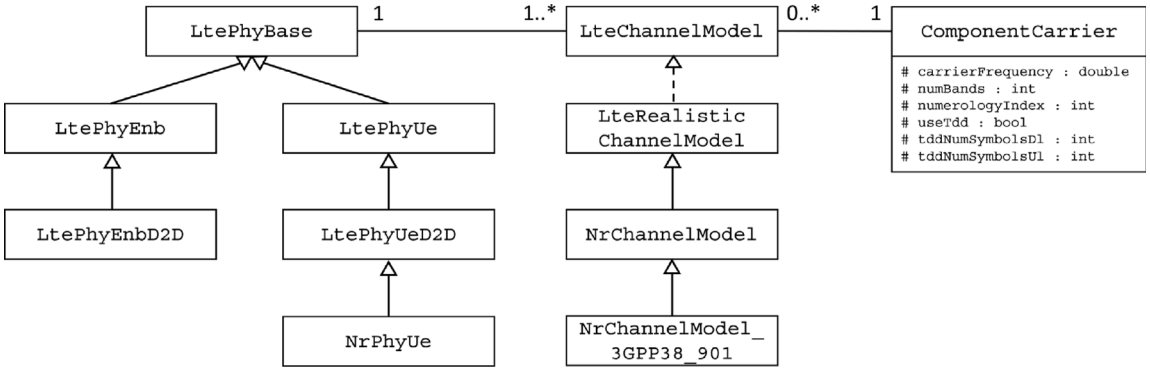

Fig. 3. Partial view of the class diagram representing the relationship between classes at the physical layer. Inheritance is widely used to implement common functionalities and reuse as much code as possible.

2.2. Software functionalities

Simu5G can simulate arbitrary 5G network scenarios, as well as mixed 4G/5G and dual connectivity deployments. Scenarios can be constructed by composing modules – from Simu5G, INET and, possibly, other OMNeT++ libraries – using the NED language. The latter allows one to write parametric simulation scenarios, e.g., a multicell 5G network with a variable number of BSs and UEs. Actual simulation *parameters*, e.g., traffic generation rate of an application, number of UEs served by a BS, etc., can be configured separately in INI files, so that a set of simulation experiments can be constructed by computing the cartesian product of all the factors.

Users willing to obtain a dataset for a particular service or application will need to configure a simulation scenario where UEs run their application of interest. Simu5G comes with a set of built-in applications (i.e., application-layer modules) representing both real-life applications – such as VoIP – and generators of synthetic traffic – e.g., Constant Bit Rate. Occasionally, users may need custom applications. In this regard, the modular nature of Simu5G allows one to easily plug new application modules. This can be done by defining the NED and C++ files for that application and configuring the INI so that UEs run that application.

In addition, users need to identify the statistics of interest that will form the dataset. Simu5G already provides a large set of predefined metrics at both BS and UE granularity, like cell-wise/UE throughput and latency measurements at multiple levels of the network protocol stack. For instance, the end-to-end latency of transmitted packets can be obtained at both the application and the MAC level. Custom statistics can also be defined. Statistics are declared within modules, namely in the NED file describing them, and probes capturing the samples are implemented within the corresponding C++ files. Statistics recording is based on *signals*. The latter are generated by a module and received by whichever other component of the simulation registered a *listener* for them. To record metrics, when a module computes a new sample, it emits a signal carrying it, which is then received by a *recorder* that stores and elaborates it to build complex statistics. Listings 1–4 exemplify the code that must be implemented to add a new statistic recording the end-to-end delay of packets received by a custom application module.

By default, each simulation instance produces two output files storing the statistics, namely a *.sca* file including all scalar-type statistics and a *.vec* file including all vector-type statistics. Since not all statistics may be of interest to the user in a particular scenario, the actual set of statistics that a simulation produces can be fine-tuned via the INI file. If statistic *statA* is defined in submodule *Y* of module *X*, a user can disable all scalars related to it by configuring:

```
**.<moduleX>.<submoduleY>.statA.scalar-recording=false.
```

Likewise, the `vector-recording` parameter can be used to control vector statistics. The hierarchical structure of the simulation scenario allows one to disable all statistics generated by a module or by the whole simulation — by using wildcards.

Output files usually need to be parsed to generate the final dataset. Being raw text files, they can be parsed using any kind of scripting. However, OMNeT++ includes tools that facilitate data extraction, e.g. an analysis tool that allows the user to browse statistics, filter them (e.g., by name, module, replicas, type) and create plots. Moreover, it exports selected statistics to JSON or CSV files (e.g., for further analysis through spreadsheet applications), or other formats readable (e.g.) by R and Matlab.

Listing 1: Definition of the signal in the NED file

```
@signal[endToEndDelaySignal](type=simtime_t);
@statistic[endToEndDelay](title="End-to-end delay"; source="endToEndDelaySignal";
record=mean,vector);
```

Listing 2: Declaration of the signal in the C++ class header file

```
omnetpp::simsignal_t endToEndDelaySignal_;
```

Listing 3: Registration of the signal at the simulation initialization in the C++ class implementation

```
endToEndDelaySignal_ = registerSignal("endToEndDelaySignal");
```

Listing 4: Emitting one sample value of the signal in the C++ class implementation

```
simtime_t delay_val = rxTime - txTime;
emit(endToEndDelaySignal_, delayVal);
```

Alternatively (e.g., if the user does not have access to a GUI), the *opp_scavetool* program included with OMNeT++ allows one to extract statistics from the command line.

3. Illustrative examples

In this example, we consider the use case of Tele-operated Driving (ToD), where vehicles can be remotely driven by a human operator. With ToD, the operator must receive a high-definition, real-time video streaming from the vehicle through the mobile network, so that he/she can be aware of the environment around the vehicle and control the latter safely. In this scenario, QoE plays a crucial role, as the ToD functionality could not be exploited if the video presents impairments at the receiving side (i.e., the remote driver). In this context, AI models may be trained for predicting possible QoE degradations in the near future, hence allowing the remote driver to take proper countermeasures, such as parking the vehicle in a safe location or requesting the passenger to take control of the vehicle. The Simu5G software can be exploited to simulate the above scenario and generate a dataset that will be used by AI experts to properly design the parameters of specific AI models. Unless specified otherwise, in the following we refer to files provided within folder *simulations/NR/videostreaming_dataset_generator*.

With reference to Fig. 4, the NED file defines a simulated network including three gNBs, namely *gnb1*, *gnb2* and *gnb3*. The latter provide the radio coverage over a main road intersected by three secondary roads. Traffic lights regulate the traffic at crossroads. Five UEs are deployed in the floorplan, each of them sending a real-time video stream to their corresponding receiving application residing at the MEC host. Eight *background* gNBs (i.e., gNBs whose only purpose is to produce realistic interference on *foreground* gNBs) complete the picture.

The above scenario is parametrized by the *omnetpp.ini* file, which is organized in several sections identified by the label in square brackets. The *General* section specifies the NED file containing the network and network-wise 5G parameters, such as carrier aggregation, handover, transmit powers and MEC system configurations. Moreover, it specifies the simulation duration (set to 900s) and the number of scenario repetitions (set to 10): the OMNeT++ environment will automatically generate ten independent replicas, each 15-minutes long and using different seeds for the pseudo-random generator. The *UrbanNetwork* section, instead, defines the parameters specific to the simulated network. For example, it configures the position of both foreground and background gNBs, as well as the number of UEs (set to five) and their mobility type. In this scenario, UEs use a custom mobility module called `TrafficLightMobility`, (see folder *src/mobility/trafficLightMobility*) which makes UEs move in a straight line at constant speed, randomly turn at crossroads, and stop when they encounter a red traffic light. This section of the INI file also defines the application module that is run by the UEs.

For the use-case presented in this paper, we implemented a custom module that models the behavior of a real-time video-streaming application at both sender and receiver side, the latter being implemented as a MEC app. The implementation can be found in folder *src/apps/RealTimeVideoStreamingApp*. The sender application generates video frames according to a trace file, each line of which specifies the size of a video frame. In our simulations, trace files were obtained from dash-camera videos using the *FFmpeg* library. For each frame, the sender generates a number of fragments and sends them via the UDP protocol. At the receiving side, the application tries to reconstruct the complete frame and play them out at the intended time. The relevant statistics for the QoE prediction – such as the inter-arrival time, frame and fragment size, percentage of frame displayed, and so on – were declared in the NED file of the sender application, i.e. *RTVideoStreamingSender.ned*. In order to generate a dataset representing the values of such metrics over time, statistics are declared as *vector* in the NED file.

In the C++ class, the corresponding signal were declared in the *.h* file, and registered (through the `registerSignal()` function) in the `handleUeMessage()` function in the *.cc* file. Then, statistics are emitted in the proper function handling the reception of fragments and the playout of frames. For example, the `playoutFrame()` is invoked when a frame has to be played out, it retrieves the number of received segment belonging to that frame and computes the percentage of the frame correctly received by dividing the total size of the received segments by the total size of the frame. The resulting value is emitted as a sample of the `frameDisplayed` statistic by invoking `ueAppModule_->emit(frameDisplayed_, percentage);`.

With reference to Fig. 5, the *omnetpp.ini* can be configured to handle which statistics are produced and where they are saved.

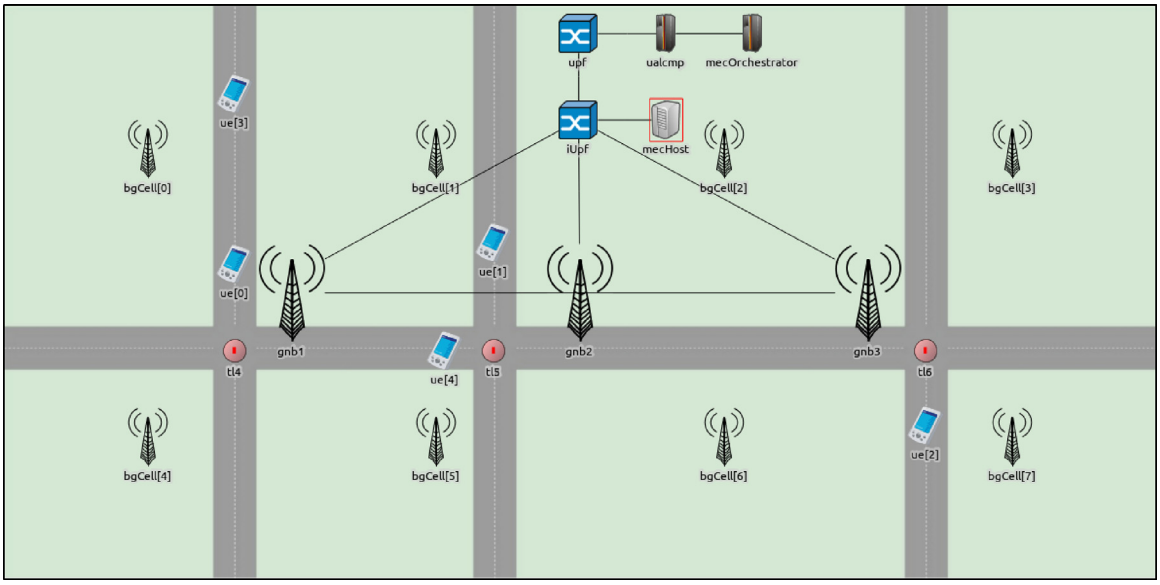

Fig. 4. Simulated scenario: 5G-enabled vehicles move along the roads deployed in the floorplan while sending a video stream using the Real-time Transport Protocol (RTP) over the radio network to the MEC Host, which is connected to the 5G core network.

```
################ Statistics ##################
**.rtVideoStreaming*.vector-recording = true
**.avgServedBlocksUl*.vector-recording = true
**.averageCqiUl*.vector-recording = true
**.rcvdSinrUl*.vector-recording = true
**.measuredSinrUl*.vector-recording = true
**.servingCell*.vector-recording = true

**.scalar-recording = false
**.vector-recording = false

output-scalar-file = ${resultdir}/${configname}/${iterationvars}-${repetition}.sca
output-vector-file = ${resultdir}/${configname}/${iterationvars}-${repetition}.vec
```

Selection of relevant metrics to be recorded. All other metrics are not recorded to save disk space and speed-up the simulations

Path where statistics are saved at the end of the simulation campaign. OMNeT++ variables are used to construct the path or file name

Fig. 5. Statistics filtering in the omnetpp.ini file: the blue box at the top includes the settings required to filter which metrics are recorded by the simulation, whereas the orange box at the bottom includes parameters that specify the output path where statistics can be retrieved at the end of the simulation.

Once the scenario is setup, we can launch the simulation campaign by typing `opp_runall simu5 g -u Cmdenv -c UrbanNetwork` from the command line. Raw statistics are saved in the files specified by the `output-vector-file` parameter in *omnetpp.ini* file, as shown in Fig. 5. In order to obtain usable data, we must parse the above files, extract the relevant statistics and produce a dataset with the desired format. Parsing and extraction of the statistics is done via the *opp_scavetool* program. This produces an intermediate CSV file, which can be parsed further to generate the final dataset. To do so, we implemented a python script, namely *DatasetExtractor.py* that filters unnecessary columns and cleans some text in order to reduce the size of the generated dataset.

The final dataset is a set of tuples with the following format:
`<run, module, metric, timestamps, values>`
Where:

- *run* is the unique identifier of the simulation that generated the metric in the tuple;
- *module* is the entity that produced the metric, e.g., *ue[0]*;
- *metric* is the name of the collected metric;
- *timestamps* and *values* are two arrays. Their ith elements, t_i and m_i concur to represent point $(t_i, metric(t_i))$.

The final rough dataset may be re-elaborated for creating a typical dataset for training and testing AI model. Usually, the AI-dataset is a collection of input–output tuples. Each tuple includes a target value to be predicted and a list of input values. Thus, specific input variable and the output variable must be extracted from the rough dataset.

4. Impact

Despite its code having been publicly available for less than 18 months, Simu5G is already being used by a large community of researchers: it has been downloaded more than 4100 times so far, from all over the world — see Fig. 6. Among the 45+ papers already citing [5],[1] 12 are from (other) research groups that use it to validate their research, some of which involve AI techniques [17–20].

Simu5G has been developed in the framework of a joint research project between the University of Pisa and Intel. It is currently being used within the Hexa-X project, EU's 6G flagship project [3], where it supports research, development and demonstration activities for Federated Learning of Explainable AI (XAI) models for QoE prediction. Recently, a preliminary performance

[1] Google Scholar search, Sept. 6, 2022

Fig. 6. Google analytics report of the Simu5G website [1] from the release of the code (Apr 15, 2021) to Sep 6, 2022. The rightmost graph shows the number of downloads per country.

analysis was presented in [1], where rough data generated using Simu5G were elaborated to create a dataset for estimating QoE values using statistics extracted from network metrics. The dataset is publicly available at [21].

Simu5G is also listed among the tools for MEC app developers in the ETSI MEC ecosystem [22] – in fact, MEC app developers can use it as a cradle to test their production-level code for performance, in realistic and customizable 5G scenarios.

5. Conclusions

This paper discussed how datasets for AI research on 4G/5G/B5G networking can be generated by using Simu5G, a popular open-source simulation library. In fact, Simu5G – that simulates all the networking stack, from the application to the physical layer, includes application-level features like, e.g., MEC hosting, and supports integration with OMNeT++-based libraries like, e.g., Veins for vehicular mobility [23] – allows users to record metrics of heterogeneous provenance, that can be used to train and test machine-learning algorithms. We have shown how a user can add arbitrary probes in the code, to record any metrics she needs in her dataset, with the desired time/space resolution.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code and the resulting data have been published in the Simu5G GitHub repository

Acknowledgments

Work partially supported by the Italian Ministry of Education and Research, Italy in the framework of the CrossLab project (Departments of Excellence), and by the European Commission through the H2020 projects Hexa-X (Grant Agreement no. 101015956). We thank our colleagues Francesco Marcelloni and Alessandro Renda for their useful discussions on this paper.

References

[1] Corcuera Bárcena JL, Ducange P, Marcelloni F, Nardini G, Noferi A, Renda A, et al. Towards trustworthy AI for QoE prediction in B5G/6G networks. In: First international workshop on artificial intelligence in beyond 5G and 6G wireless networks. Padua, IT; 2022, p. 18–23.
[2] Renda A, et al. Federated learning of explainable AI models in 6G systems: Towards secure and automated vehicle networking. Information 2022;13(8):395. http://dx.doi.org/10.3390/info13080395.
[3] Hexa-X project. 2023, website: https://hexa-x.eu [Accessed January 2023].
[4] ETSI white paper (28) MEC in 5G networks. 2018, link: https://bit.ly/3bzCLFI [Accessed January 2023].
[5] Nardini G, Sabella D, Stea G, Thakkar P, Virdis A. Simu5G–an OMNeT++ library for end-to-end performance evaluation of 5G networks. IEEE Access 2020;8:181176–91. http://dx.doi.org/10.1109/ACCESS.2020.3028550.
[6] Noferi A, Nardini G, Stea G, Virdis A. Rapid prototyping and performance evaluation of ETSI MEC-based applications. Elsevier Simul Model Pract Theory 2023;123:102700. http://dx.doi.org/10.1016/j.simpat.2022.102700.
[7] Nardini G, Stea G, Virdis A. Scalable real-time emulation of 5G networks with Simu5G. IEEE Access 2021;9:148504–20. http://dx.doi.org/10.1109/ACCESS.2021.3123873.
[8] Kim Y, et al. 5G K-simulator: 5G system simulator for performance evaluation. In: 2018 IEEE international symposium on dynamic spectrum access networks. Seoul, Korea (South; 2018, p. 1–2. http://dx.doi.org/10.1109/DySPAN.2018.8610404.
[9] Müller M, Ademaj F, Dittrich T, et al. Flexible multi-node simulation of cellular mobile communications: The vienna 5G system level simulator. J Wireless Com Netw 2018 2018;227. http://dx.doi.org/10.1186/s13638-018-1238-7.
[10] Patriciello N, Lagen S, Bojovic B, Giupponi L. An E2E simulator for 5G NR networks. Simul Model Pract Theory 2019;96:101933. http://dx.doi.org/10.1016/j.simpat.2019.101933.
[11] Martiradonna S, Grassi A, Piro G, Boggia G. 5G-air-simulator: An open-source tool modeling the 5G air interface. Comput Netw 2020;173(22):107151. http://dx.doi.org/10.1016/j.comnet.2020.107151.
[12] Ns3 network simulator. 2023, Website: https://www.nsnam.org [Accessed January 2023].
[13] OMNeT++ discrete event simulator. 2023, Website: https://omnetpp.org [Accessed January 2023].
[14] INET framework. 2023, Website: https://inet.omnetpp.org [Accessed January 2023].
[15] Sommer C, German R, Dressler F. Bidirectionally coupled network and road traffic simulation for improved IVC analysis. IEEE Trans Mob Comput 2011;10(1):3–15. http://dx.doi.org/10.1109/TMC.2010.133.
[16] ETSI GS MEC 013 v2.2.1. In: Multi-access edge computing. Location API; 2022.
[17] Batista JOR, da Silva DC, Martucci M, Silveira RM, Cugnasca CE. A multi-provider end-to-end dynamic orchestration architecture approach for 5G and future communication systems. Appl Sci 2021;11(24):11914. http://dx.doi.org/10.3390/app112411914.
[18] Nguyen AC, Pamuklu T, Syed A, Kennedy WS, Erol-Kantarci M. Reinforcement learning-based deadline and battery-aware offloading in smart farm IoT-UAV networks. In: ICC 2022 - IEEE international conference on communications. Seoul, Korea, Republic of; 2022, p. 189–94. http://dx.doi.org/10.1109/ICC45855.2022.9838500.

[19] Antonio G-P, Maria-Dolores C. AIM5la: A latency-aware deep reinforcement learning-based autonomous intersection management system for 5G communication networks. Sensors 2022;22(6):2217. http://dx.doi.org/10.3390/s22062217.

[20] Rehman A, Haseeb K, Saba T, Lloret J, Sendra S. An optimization model with network edges for multimedia sensors using artificial intelligence of things. Sensors 2021;21(21):7103. http://dx.doi.org/10.3390/s21217103.

[21] Dataset for QoE prediction in B5G/6G networks. 2023, [Accessed January 2023].

[22] MEC ecosystem wiki. 2022, [Accessed May 2022].

[23] Nardini G, Virdis A, Stea G. Simulating cellular communications in vehicular networks: Making SimuLTE interoperable with Veins. In: Proc. of the 4th OMNeT++ community summit. Bremen, DE; 2017, p. 7–8. http://dx.doi.org/10.48550/arXiv.1709.02208.