Enabling Online Reinforcement Learning Training for Open RAN

Andrea Lacava*†, Tommaso Pietrosanti†, Michele Polese*, Francesca Cuomo†, Tommaso Melodia*
*Institute for the Wireless Internet of Things, Northeastern University, Boston, MA, USA
†Sapienza University of Rome, Rome, Italy
Email: {lacava.a, m.polese, t.melodia}@northeastern.edu
Email: pietrosanti.1994391@studenti.uniroma1.it, francesca.cuomo@uniroma1.it

***Abstract*—The Open Radio Access Network (RAN) architecture has introduced new elements in the RAN, i.e., the RAN Intelligent Controllers (RICs), which allow for closed-loop control of the physical infrastructure through custom, data-driven, intelligent applications. At the near-real-time RIC, *xApps* access RAN nodes through the E2 interface and offer the option to host traditional algorithms or data-driven Deep Reinforcement Learning (DRL) agents to control and optimize RAN functionalities. The O-RAN specifications suggest that Artificial Intelligence (AI) model training should not be done on production RAN deployments to avoid network disruptions and degradation in the Quality of Experience (QoE) of the User Equipments (UEs), suggesting the adoption of offline reinforcement learning. However, this approach limits the exploration phase during training to the static data that has already been collected, potentially affecting the performance of the model and its generalization capabilities. Therefore, a safe environment capable of supporting online reinforcement learning is needed to overcome such constraints and to allow AI agents to perform state explorations freely. In this paper, we present a new control environment based on Gymnasium (gym), a Python library for the creation of reinforcement learning environments, and ns-O-RAN, a software integration between a real-world near-real-time RIC and an Network Simulator 3 (ns-3) simulated RAN, which exposes the RAN Key Performance Indicators (KPIs) through a standardized Application Programming Interface (API) ready to be used by any solving approach. Leveraging ns-O-RAN, we create an environment that dynamically captures the simulated O-RAN telemetry, waits for the agent to compute a decision, receives and delivers such control action to update the RAN configuration in the underlying simulation, allowing the development and test of models in safe and reproducible conditions. Finally, our framework exposes an abstract API interface that can be extended to support custom use cases to design AI agents and to study the Radio Resource Management (RRM) issues of the next generation of cellular networks.**

***Index Terms*—Open RAN, ns-3, deep reinforcement learning, artificial intelligence, gymnasium**

I. Introduction

Despite the performance improvement introduced by 5th generation (5G) cellular systems, the traditional monolithic cellular infrastructure does not offer the flexibility to optimize and control the Radio Access Network (RAN) at the granularity needed by a specific User Equipments (UEs). In recent years, the O-RAN ALLIANCE [1], a community of mobile operators, vendors, and academic institutions, has begun to embrace the Open RAN paradigm and define open specifications focused on agile and disaggregated architectures. These architectures are based on softwarization and virtualization to exploit the re-programmability of network components, thereby improving Radio Resource Management (RRM) and control of the RAN [2]. This new ecosystem has created opportunities for standardized data collection and dynamic control of the RAN parameters through the exchange of messages over the O-RAN interfaces. Such messages enable tight control loops between the RAN and the RAN Intelligent Controller (RIC) to manage the behavior of the system through control applications, called xApps for the near-real-time control granularity [3]. The xApps can implement heuristics or data-driven approaches and are usually created by different vendors [4]. This architecture paves the way for the use of novel approaches, among with Deep Reinforcement Learning (DRL), for various RRM use cases, such as traffic steering, anomaly detection, energy saving, and more. To enable the study of the possibilities introduced by the Open RAN architecture, we created ns-O-RAN [5], the first open-source simulation platform that combines a functional 3rd Generation Partnership Project (3GPP) RAN 5G protocol stack in ns-3 with an O-RAN-compliant E2 interface. In real RAN deployments, the online DRL training is discouraged by the O-RAN specifications [6], which identifies potential risks of outages for the UEs associated with the natural exploration phase of the DRL algorithms. In ns-O-RAN, UEs are simulated, making our framework ideal for developing data-driven algorithms. However, most implementations of these algorithms rely on functions not exposed in ns-O-RAN, complicating their use, especially for controlling the online DRL training process. These functions are implemented through standard frameworks such as Gymnasium (gym) [7], which provides a unified interface for single-agent reinforcement learning environments. In this paper, we expand the ns-O-RAN framework capabilities to support online DRL by introducing a gym environment that acts as an Application Programming Interface (API) interface to ns-O-RAN.[1] To the best of our knowledge, our work is the first gym-based DRL environment for Open RAN powered by 3GPP-based radio channel models, leveraging the

This work was partially supported by the U.S. National Science Foundation under Grants CNS-2120447 and CNS-2112471.

[1] https://github.com/wineslab/ns-o-ran-gym-environment

flexibility of ns-3. This allows researchers to compare the effectiveness of various reinforcement learning algorithms in a standardized manner across the same network optimization problems. This integration offers several advantages. First, it provides standardized access to the ns-O-RAN framework since the gym APIs are the *de facto* standard in the scientific community for reinforcement learning, facilitating the usage of the ns-O-RAN framework with state-of-the-art DRL algorithm frameworks. Second, the use of gym facilitates the adoption of the environment by those familiar with its APIs, even if they are not interested in the specific internals of ns-O-RAN, thus providing abstraction and generalization to the whole framework. Finally, the different classes implemented in this integration improve and modularize data management during the simulation, with specific tasks such as updating the environment and synchronization between the agent and ns-3. This ensures that the ns-O-RAN framework can be upgraded without requiring changes to the model used for studying its use cases. The main contributions of this paper are the following:

- We describe the changes made in the ns-O-RAN codebase to enable the synchronization between ns-3 and external software;
- We present the *NsOranBase* environment, explain how it is interfaced with the rest of the ns-3 framework, and how can be extended to develop custom use cases;
- We implement a proof-of-work online environment for the Traffic Steering use-case based on the problem formulation of [8];
- We benchmark our implementation demonstrating that the overhead caused by our code is minimal compared to the usual performance of ns-O-RAN.

The remainder of the paper is organized as follows. Section II reviews the state of the art for DRL and simulated control environments for cellular networks. Section III introduces the general architecture of the ns-O-RAN Gym base environment, how it works and its general capabilities. Section IV introduces the environment created for the Traffic Steering use case and its features implemented, including an analysis of the performance overhead. Finally, Section V presents possible future works based on such framework and the overall conclusion.

II. State of the Art

A. The Open RAN Architecture

Key principles of O-RAN are the disaggregation of the standard network functions into different entities and the on-demand re-programmability of these functions using open standardized interfaces. The management of the different entities in O-RAN takes place in the RIC, which acts as the centralized abstraction of the network and is connected with all the Base Stations (BSs) of a specific geographic area. The O-RAN Alliance defines two different types of RIC: the non-real-time RIC, which operates on a time scale longer than 1 s, and the near-real-time RIC, which operates on a time scale between 10 ms and 1 s [9]. From a research point of view, the collection of the Key Performance Indicators (KPIs) from all the RAN borders in one place paves the way for the construction of standardized virtual applications to create Artificial Intelligence (AI)-enabled RRM agents to be deployed through xApps.

B. Reinforcement Learning and Wireless Network Simulators

Deep Reinforcement Learning (DRL) is a class of Machine Learning (ML) problems where an *agent* optimizes a mathematical function known as the *reward*, which quantifies the performance of a system of interest referred to as the *environment*. The agent's goal is to derive an association, named *policy*, between the different observations, named *states*, of the environment and its reward, and to modify the environment through its *actions*. Due to its adaptability, DRL has become popular among wireless network researchers [10]. One of the major challenges of DRL is the availability of datasets and reliable baselines and the lack of commonly accepted APIs makes reproducibility and validation of results difficult [11]. This is what motivated the creation of foundational frameworks such as gym [7], which creates a standardized interface that allows common access to heterogeneous environments, from video games [12] to drones, and more,[2] as well as Stable-Baselines 3, a Python library that implements some of the most important state-of-the-art DRL algorithms [13]. The same challenge persists for DRL-based optimization problems in wireless networks and especially in O-RAN, where the deployment of untrained AI agents may be detrimental to the final user experience and is, therefore, discouraged [6]. Similar frameworks in the literature are mostly non-based on the Open RAN E2 Service Model (SM) and are hard to transition from a prototype to a real xApp. In [14], Schneider et al. propose a minimal gym environment for autonomous coordination in wireless mobile networks. Despite its versatility, this environment does not represent the wireless radio channels, especially compared to other well-established network simulators such as ns-3 and Omnet++. For this work, we chose ns-3 as our simulator due to its accurate radio channel and protocol stack models, as well as its extensibility. Some works in the past have connected the power of ns-3 with the Gym interface [15], [16]. These frameworks require major changes within the simulation scenarios and are only for specific versions of ns-3, making them not supported by ns-O-RAN.

C. ns-O-RAN primer

ns-O-RAN is the first open-source simulation platform combining a functional LTE/5G protocol stack in ns-3 with an O-RAN-compliant E2 interface [5]. This platform, alongside the gym environment implemented in our work, forms part of the OpenRAN Gym framework[3] [17], an experimental toolbox for end-to-end design, data collection, and testing workflows for intelligent control in next-generation Open RAN systems. Thanks to these, researchers and network operators can test

[2]https://gymnasium.farama.org/environments/third_party_environments/
[3]https://openrangym.com/

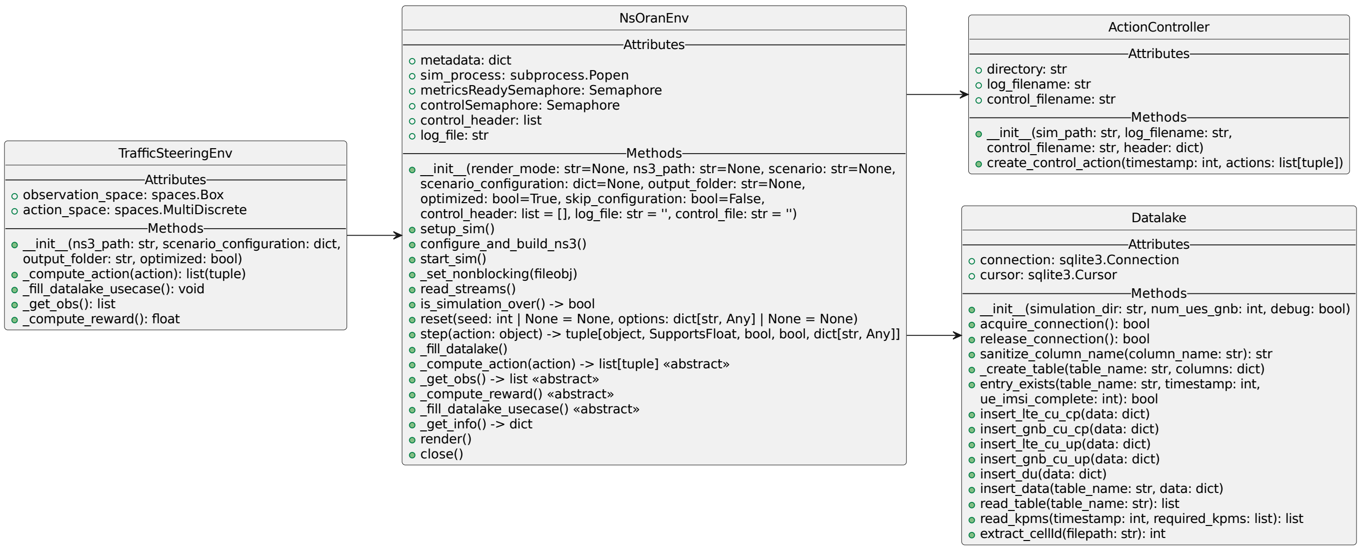

TrafficSteeringEnv
Attributes
observation_space: spaces.Box
action_space: spaces.MultiDiscrete
Methods
__init__(ns3_path: str, scenario_configuration: dict, output_folder: str, optimized: bool)
_compute_action(action): list(tuple)
_fill_datalake_usecase(): void
_get_obs(): list
_compute_reward(): float
NsOranEnv
Attributes
metadata: dict
sim_process: subprocess.Popen
metricsReadySemaphore: Semaphore
controlSemaphore: Semaphore
control_header: list
log_file: str
Methods
__init__(render_mode: str=None, ns3_path: str=None, scenario: str=None, scenario_configuration: dict=None, output_folder: str=None, optimized: bool=True, skip_configuration: bool=False, control_header: list = [], log_file: str = '', control_file: str = '')
setup_sim()
configure_and_build_ns3()
start_sim()
_set_nonblocking(fileobj)
read_streams()
is_simulation_over() -> bool
reset(seed: int | None = None, options: dict[str, Any] | None = None)
step(action: object) -> tuple[object, SupportsFloat, bool, bool, dict[str, Any]]
_fill_datalake()
_compute_action(action) -> list[tuple] «abstract»
_get_obs() -> list «abstract»
_compute_reward() «abstract»
_fill_datalake_usecase() «abstract»
_get_info() -> dict
render()
close()
ActionController
Attributes
directory: str
log_filename: str
control_filename: str
Methods
__init__(sim_path: str, log_filename: str, control_filename: str, header: dict)
create_control_action(timestamp: int, actions: list[tuple])
Datalake
Attributes
connection: sqlite3.Connection
cursor: sqlite3.Cursor
Methods
__init__(simulation_dir: str, num_ues_gnb: int, debug: bool)
acquire_connection(): bool
release_connection(): bool
sanitize_column_name(column_name: str): str
_create_table(table_name: str, columns: dict)
entry_exists(table_name: str, timestamp: int, ue_imsi_complete: int): bool
insert_lte_cu_cp(data: dict)
insert_gnb_cu_cp(data: dict)
insert_lte_cu_up(data: dict)
insert_gnb_cu_up(data: dict)
insert_du(data: dict)
insert_data(table_name: str, data: dict)
read_table(table_name: str): list
read_kpms(timestamp: int, required_kpms: list): list
extract_cellId(filepath: str): int

Fig. 1: Simplified Unified Modeling Language (UML) Diagram of the ns-O-RAN and the Traffic Steering environments

applications in a safe environment, collect data on a large scale, and design and train data-driven approaches for the RRM. ns-O-RAN has been specifically designed to integrate the O-RAN RIC with large-scale 5G simulations using 3GPP radio channel models. This integration enables the collection of RAN Key Performance Measurements (KPMs) across various simulated scenarios and applications, such as multimedia streaming, web browsing, wireless virtual reality, and more. ns-O-RAN supports E2SM-KPMs monitoring [18] and E2SM-RAN Control (RC) [19] functionalities. The framework is composed of three main software:

- the `e2sim` module, which is responsible of the management of the connection between ns-3 and RIC, the encoding and decoding of the E2 Application Protocol (AP), and the delivery of E2 SM messages;
- the `ns-O-RAN` module, which is responsible of interconnect ns-3 with `e2sim`;
- the `mmWave` module, which we extended to extract the RAN features compliant with the O-RAN specifications.

ns-O-RAN enables the simulated BSs to exchange messages with a near-real-time RIC that follows the same E2 protocol. ns-O-RAN has two modes of operation: (i) bridged, where an active E2 connection to a near-real-time RIC is needed, and (ii) stand-alone, where simulation-generated metrics can be collected by logging into files and no connection is required. The RC can be achieved in both modes. In the bridged mode, the E2 SM RC messages are created in the RIC and delivered to the e2sim RAN termination that will subsequently trigger a callback based on the RAN function identifier. In the stand-alone mode, the same control is file-based, thus an external process is supposed to write the actions to be implemented in the network in a dedicated file, specific per use case. In this work, we exploit the stand-alone mode of ns-O-RAN to create a gym environment that enables the possibility of performing online training and testing of data-driven solutions for network and RRM optimization problems.

III. System overview

A. *`NsOranEnv`: the ns-O-RAN base environment*

ns-O-RAN was not designed as a DRL framework for a specific RRM scenario, but as a library that can support different use cases with different observation spaces, control types, and rewards. Such flexibility cannot be provided directly by a single environment, thus we design an abstract middle layer responsible for handling all communications between the controller and the ns-3 module. The `NsOranEnv` class is an abstract base class for creating environments compliant with the gym framework, specifically designed for ns-O-RAN simulations. This class provides a structured way to initialize, configure, and manage the ns-3 simulations, facilitating the integration of DRL algorithms. A review of the main classes implemented is shown in the UML diagram of Fig. 1. The `NsOranEnv` class leverages the `Datalake` and `ActionController` classes, which are responsible for extracting data from the simulation and delivering actions generated by the controller, respectively.

The `Datalake` is a Python wrapper for a SQLite database that stores the KPMs generated by ns-O-RAN and works on an independent thread. This database is utilized to leverage the agent's control action based on the current simulation state. Each time the environment is reset, a new simulation starts, and thus a new database is created. All implemented KPMs in ns-O-RAN are saved in the database, regardless of the environment's observation set. The list of KPMs related to a specific use case can be accessed using the `read_kpms()` method, which takes as input the timestamp and the list of required features and returns the per-UE KPMs.

The agent then writes the action on the shared control file through the `ActionController`, which creates and updates the file that will be read and parsed by the stand-alone mode of ns-O-RAN. The control action is specific per use case. Indeed, the abstract `NsOranEnv` class provides four methods that another class must override, e.g., `TrafficSteeringEnv` presented in Section IV, for the environment to function properly. These methods implement a specific use case since the data they return depends on the optimization problem being studied: (i) `_get_obs()`, which returns the observation state; (ii) `_compute_reward()`, which returns the reward function based on the observation state; (iii) `_compute_action()`, which is a helper function that converts the agent's action defined in gym into the format required by ns-O-RAN before triggering the `ActionController` class, and (iv) `_fill_datalake_usecase()`, which is a function used to capture additional data from ns-O-RAN and store it in the `Datalake`. Any additional data definition must always use the ns-O-RAN timestamp and the UE International Mobile Subscriber Identity (IMSI) as primary keys. This last function is optional, so it does not throw an exception if not implemented. Finally, the `NsOranEnv` can reconfigure the ns-3 framework, allowing the user to decide whether to use an optimized or a debug build of ns-3. Part of the codebase that handles the reconfiguration and the start of the simulation has been ported from Simulation Execution Manager (SEM), a framework that enables large-scale data collection campaigns for ns-3 [20].

B. *Synchronization of the Gymnasium environment with ns-3*

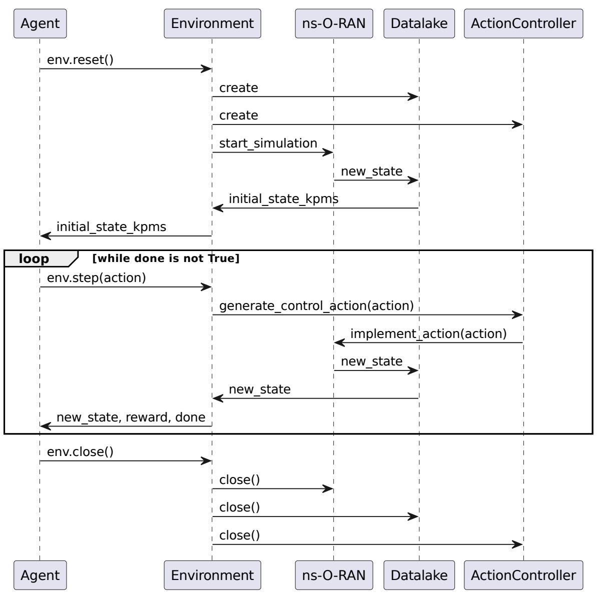


Fig. 2: Activity diagram with the agent and the environment

The ns-O-RAN framework is based mainly on ns-3, which is a discrete-time simulator where each network event is associated with a function and a timestamp that indicates when the event should be executed. Events with the same timestamp are executed with a First In First Out (FIFO) mechanism, preserving the happened-before relationship in ns-3. Moreover, the simulator is a single-core process that can be extended with callbacks based on the C++ multi-threading system, but it was not designed to accept external inputs to ensure the reproducibility of each single simulation upon the same random seed. This restriction does not apply to DRL since the reproducibility of a result in a real or pseudo-real environment is based on a statistical evaluation and not on individual simulations, except in the case where the environment is deterministic. Fig. 2 illustrates the interactions between an external agent and all components of the environment. Each time the agent resets the environment, a new simulation is initiated and continues until the first set of KPMs is generated. The periodicity of the KPMs generation coincides with the E2 SM KPM Indication Message Periodicity [18]. After retrieving the data, the environment parses it based on the observation space and sends it to the agent, which then computes an action according to the defined action space. In our gym environment, the `NsOranEnv` creates and manages the simulation. The scenario to be simulated is defined on the ns-3 side, while the parameters of the single simulation can be changed directly from the environment. In this way, we have two different processes performing I/O operations over the same files, so it is important to synchronize them to avoid possible race conditions. Such synchronization is achieved through the use of two different named semaphores, one that signals when the metrics are ready to be read from the environment and written into the `Datalake`, and the other that signals to the simulation when the `ActionController` has finished writing the E2 SM RC Control Message to the control file, thus delivering the action to ns-O-RAN. Both the semaphores are created at the beginning of the simulation and destroyed when the environment is closed, either by calling the `close()` or the `reset()` methods. The semaphore for Control Action delivery is blocking, while the other is non-blocking and embedded in a loop that waits for new metrics to become available. Inside the loop, the semaphore attempts to access the shared files within a 10-second window. If access is granted, the loop breaks and the metrics are extracted. If the window times out, the environment checks if the simulation process is still active. If the simulation has ended, it transitions to the final state and initiates closing procedures. Following the gym APIs, each call to the `step()` method returns two additional boolean values: one indicating if the environment is closed and another specifying whether the closure was due to truncation or natural termination of the simulation. Since there is no natural closure condition in a RAN optimization problem, we always return the truncated boolean variable as true, and since ns-3 imposes a simulation time limit that can be dynamically changed by the user [21]. In the literature, another case that predicts truncation as true occurs when the conditions of the Markovian Decision Process (MDP) are violated, which is what we return if the simulation ends with an error. In this scenario, the environment sets the termination variable to false and displays the error and the simulation's return code.

IV. Traffic Steering implementation

In this section, we present an example of a gym environment that extends the `NsOranEnv` class to a specific use case by defining all the necessary components that form a MDP for an Open RAN-based RRM optimization problem, thus proving our framework as a practical way to design and study DRL problems using online algorithms. Furthermore, we use the same environment to benchmark the performance of the framework and show that the time overhead introduced by the gym APIs is negligible compared to the computation time of the simulation. We implement the environment for the O-RAN Traffic Steering use case, the optimization problem definition and the reference scenario from [8].

A. Simulation Scenario

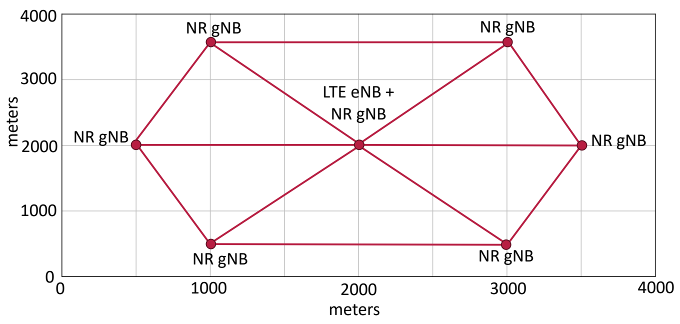


Fig. 3: Simulation scenario implemented for `TrafficSteeringEnv` class

We reproduce a 5G Non Stand Alone (NSA) scenario with one primary evolved Node Base (eNB) cell and seven secondary Next Generation Node Bases (gNBs) cells, as shown in Fig. 3. The eNB is placed at the center of the scenario and is co-located with one gNB, while the remaining gNBs are placed at a fixed inter-site distance of 1000 m from the center. Each UE is initially located close to a gNB, with exact coordinates defined using a uniform distribution. Notable tunable parameters in the scenario configuration include the E2 Indication Periodicity, simulation time, number of UEs per-gNB, and size of the Physical Resource Block (PRB) buffer.

To benchmark our environment, we make the UEs request downlink traffic from a remote server with a mixture of four traffic models: Full Buffer, Bursty, Half Full/Half Bursty, and Mixed. The Full Buffer model simulates Maximum Bit Rate (MBR) traffic with a maximum data rate of 20 Mbit/s, representing file transfer or synchronization with cloud services. The Bursty model represents traffic with an average data rate of 3 Mbit/s, modeling video streaming applications. The Half Full/Half Bursty model includes 50% of UEs with Full buffer traffic and 50% with Bursty traffic. The Mixed model includes 25% Full Buffer, 25% Bursty at 3 Mbps, 25% Bursty at 750 Kbit/s, and 25% Bursty at 150 Kbit/s, representing a range of applications from web browsing and instant messaging to phone calls. More details about the implemented scenario can be found in the original paper [8].

B. Extending the `NsOranEnv` Environment

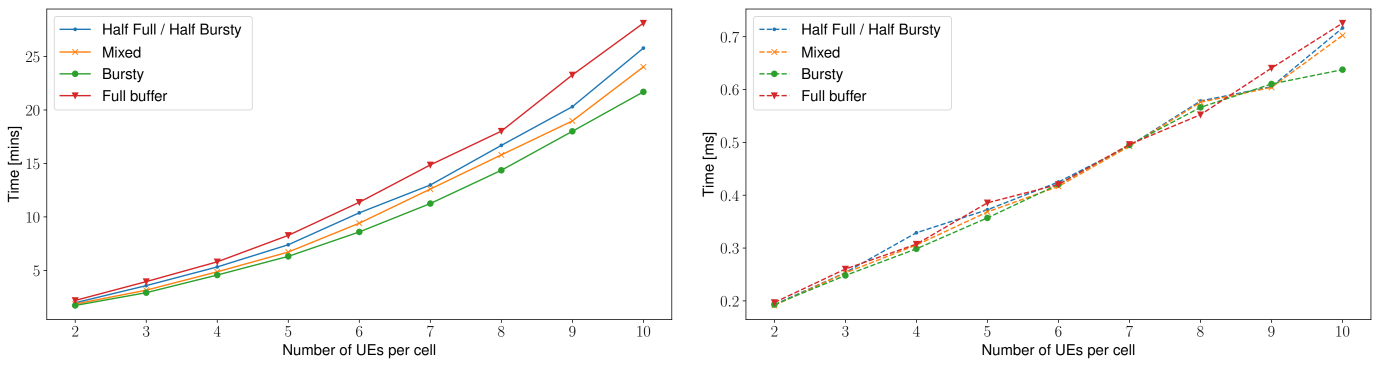
The `NsOranEnv` must be extended with additional information to shape a specific MDP network optimization problem. In particular, the extended entity shall describe:

1) the action space, which represents the set of possible commands that the agent can execute on the network;
2) the observation space, which describes the elements that the agent can analyze to generate the action;
3) the reward that returns feedback to the agent of the impact that the action it previously sent has on the system.

Following these principles, we present `TrafficSteeringEnv`, a class implementing the Traffic Steering use case. In this environment, we extend the abstract methods of `NsOranEnv` according to the problem specifications. Indeed, we define the action space as a multi-discrete variable $[U, A]$, where U is the number of the UEs in the simulation and A represents the action each UE can take at each E2 indication periodicity. Specifically, A is a discrete variable with cardinality equal to the number of gNBs in the scenario minus two. In this way, each value $[u, a]$ of A, represents the association between the u-th UE with the $a+2$-th gNB id, including the solution where $a = 0$, meaning that no handover shall be performed for that particular UE. The action space can be converted into a discrete variable with $U \times A$ cardinality, where each value is mapped to a single tuple of the original variable. The observation space is a set of selected per-UE and per-cell KPMs that are extracted using the `Datalake`. Finally, the reward for UE u is the logarithmic throughput between the actual and the last E2 Indication periodicity minus a fixed cost factor representing the mobility overhead. The reward is positive, if the improvement in the $\log$-throughput is higher than its cost, and negative, otherwise.

C. Performance evaluation

We study the impact of our environment on the overall simulation. The implementation of additional code and the management of the environment brings a cost in the performance of ns-3. We define the elapsed time as the total duration from when the environment is reset to when it is closed, and the overhead as the additional time spent by the program on operations related to the gym environment and semaphore sleeps. Fig. 4a illustrates the time required in minutes for different numbers of UEs per-cell across these traffic models. As the number of UEs increases, the time required also increases. The Full buffer model consistently requires more time, followed by the Half Full/Half Bursty and Mixed models, with the Bursty model requiring the least time. Fig. 4b shows the time overhead introduced with the gym environment. In this case, the y-axis is in milliseconds and clearly shows that as the number of UEs in the simulation increases, the overhead increases proportionally, with no significant variation related to the traffic model. This behavior is consistent with expectations, as an increasing number of UEs in the simulation



(a) Total elapsed time for executing a simulation

(b) Time spent outside the simulation due to the processes of the gym environment

Fig. 4: Measurements of the simulation time in minutes compared to the overhead caused by the gym environment in milliseconds

results in more data generated by the framework for per-UE KPM reports, thus increasing the workload handled by the environment. Overall, the overhead is significantly lower (nanoseconds vs. minutes) compared to the time required for the simulation, making the impact of the gym environment on performance negligible.

V. Conclusions

In this paper, we presented a novel gym Environment that embeds ns-O-RAN and exposes a standardized APIs capable of supporting the current state-of-the-art frameworks for DRL. Future work includes extending this framework with new use cases such as energy saving, network slicing, and others, to establish ns-O-RAN as the definitive testing benchmark for intelligent cellular networks within the O-RAN architecture.

References

[1] C. Ziqi and I. Chih-Lin, “O-ran alliance vertical industry whitepaper,” Dec. 2023. [Online]. Available: https://mediastorage.o-ran.org/white-papers/O-RAN.WG1.Vertical-Industry-White-Paper-2023-12.pdf

[2] L. Bonati, M. Polese, S. D’Oro, S. Basagni, and T. Melodia, “Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead,” *Computer Networks*, vol. 182, p. 107516, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620311786

[3] 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*, pp. 1–14, July 2022.

[4] O-RAN Working Group 1, “O-RAN Architecture Description 5.00,” O-RAN.WG1.O-RAN-Architecture-Description-v05.00 Technical Specification, July 2021.

[5] A. Lacava, M. Bordin, M. Polese, R. Sivaraj, T. Zugno, F. Cuomo, and T. Melodia, “ns-o-ran: Simulating o-ran 5g systems in ns-3,” in *Proceedings of the 2023 Workshop on ns-3*, 2023, pp. 35–44.

[6] O-RAN Working Group 2, “O-RAN AI/ML workflow description and requirements 1.03,” O-RAN.WG2.AIML-v01.03 Technical Specification, July 2021.

[7] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, “Gymnasium,” Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[8] A. Lacava, M. Polese, R. Sivaraj, R. Soundrarajan, B. S. Bhati, T. Singh, T. Zugno, F. Cuomo, and T. Melodia, “Programmable and customized intelligence for traffic steering in 5g networks using open ran architectures,” *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2882–2897, 2024.

[9] 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.

[10] S. Ergun, I. Sammour, and G. Chalhoub, “A survey on how network simulators serve reinforcement learning in wireless networks,” *Computer Networks*, vol. 234, p. 109934, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128623003791

[11] F. Felten, L. N. Alegre, A. Nowe, A. Bazzan, E. G. Talbi, G. Danoy, and B. C. da Silva, “A toolkit for reliable benchmarking and research in multi-objective reinforcement learning,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 23 671–23 700.

[12] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[13] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[14] S. Schneider, S. Werner, R. Khalili, A. Hecker, and H. Karl, “mobile-env: An open platform for reinforcement learning in wireless mobile networks,” in *Network Operations and Management Symposium (NOMS)*. IEEE/IFIP, 2022.

[15] P. Gawłowicz and A. Zubow, “Ns-3 meets openai gym: The playground for machine learning in networking research,” in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2019, pp. 113–120.

[16] Yin, Hao and Liu, Pengyu and Liu, Keshu and Cao, Liu and Zhang, Lytianyang and Gao, Yayu and Hei, Xiaojun, “Ns3-ai: Fostering artificial intelligence algorithms for networking research,” in *Proceedings of the 2020 Workshop on Ns-3*, ser. WNS3 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 57–64. [Online]. Available: https://doi.org/10.1145/3389400.3389404

[17] L. Bonati, M. Polese, S. D’Oro, S. Basagni, and T. Melodia, “Openran gym: An open toolbox for data collection and experimentation with ai in o-ran,” in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2022, pp. 518–523.

[18] O-RAN Working Group 3, “O-RAN near-real-time RAN intelligent controller E2 service model (E2SM) KPM 2.0,” ORAN-WG3.E2SM-KPM-v02.00 Technical Specification, July 2021.

[19] ——, “O-RAN near-real-time RAN intelligent controller E2 service model, ran control 1.0,” ORAN-WG3.E2SM-RC-v01.00 Technical Specification, July 2021.

[20] D. Magrin, D. Zhou, and M. Zorzi, “A simulation execution manager for ns-3: Encouraging reproducibility and simplifying statistical analysis of ns-3 simulations,” in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2019, pp. 121–125.

[21] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev, “Time limits in reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 4045–4054.