LLM-Driven Reward Design for Intelligent RAN Slice Scheduling in Open RAN

Pedro Sousa[1], Frank B. Morte[1], Andrey Oliveira[1], Cleverson Nahum[1], Silvia Lins[2], Andrey Silva[2], Aldebaro Klautau[1]

[1]Instituto de Tecnologia – Universidade Federal do Pará (UFPA) - Brazil

[2]Innovation Center, Ericsson Telecomunicações S.A, Brazil.

{pedro.sousa, frank.bruno, andrey.oliveira}@itec.ufpa.br,
{cleversonahum, aldebaro}@ufpa.br
{silvia.lins, andrey.silva}@ericsson.com

Abstract. *Radio resource scheduling is a fundamental task in network slicing scenarios, where different services impose heterogeneous performance requirements. This paper proposes the use of large language models (LLMs) to automatically generate reward functions for Reinforcement Learning-based radio resource schedulers in Open Radio Acess Network (RAN) environments. By adapting the reward formulation according to the characteristics and requirements of two use cases, the approach increases the flexibility of the scheduling policy. Simulation results demonstrate the feasibility of the method and indicate that LLM-assisted reward design can support efficient resource allocation with stable throughput while satisfying slice-level requirements.*

1. Introduction

Future beyond-5G and emerging 6G networks must simultaneously support highly heterogeneous applications—such as extended reality, cloud gaming, industrial automation, and smart healthcare, each imposing distinct requirements on throughput, latency, and reliability [Quy et al. 2023]. Network slicing has emerged as a key enabler of this diversity by allowing multiple logical networks to coexist over shared infrastructure, each tailored to a specific service-level agreement (SLA) [Afolabi et al. 2018]. Within this context, radio resource scheduling radio resource scheduling (RRS) plays a central role in allocating limited radio resources among slices and user equipments while ensuring SLA compliance.

To address the complexity of such dynamic environments, reinforcement learning (RL) has been widely adopted for RRS, leveraging its ability to learn optimal policies directly from network observations without relying on explicit analytical models [Calabrese et al. 2018]. Prior work has formulated scheduling as a Markov decision process (MDP), where RL agents allocate resource blocks to optimize objectives such as throughput maximization for enhanced mobile broadband (eMBB) and latency minimization for ultra-reliable low latency communication (URLLC) [Polese et al. 2022, Raftopoulos et al. 2024]. Other approaches exploring intent-based schedulers have demonstrated promising performance in both inter-slice and intra-slice resource allocation scenarios [Nahum et al. 2023, Nahum et al. 2026].

Despite these advances, a critical limitation persists in the design of reward functions that guide RL agents. Most existing approaches rely on handcrafted rewards, which

require significant domain expertise and often encode fixed trade-offs between competing performance metrics [Polese et al. 2022, Raftopoulos et al. 2024, Nahum et al. 2023, Nahum et al. 2026, Zangooei et al. 2023]. Such designs tend to be highly scenario-specific, leading to limited generalization when network conditions change, including variations in traffic patterns, user distributions, or channel characteristics. Consequently, RL-based schedulers often struggle to adapt effectively in realistic, dynamic deployments.

Recent developments in large language model (LLM) provide a novel opportunity to overcome this limitation. LLMs have demonstrated strong capabilities in reasoning, structured problem solving, and code generation, and have recently been explored for assisting in the design of RL components, including reward functions [Yu et al. 2023]. In complex systems with multiple interacting objectives, LLMs can dynamically generate or adapt reward formulations based on high-level descriptions of system goals and constraints, offering a more flexible and context-aware alternative to static reward engineering. Motivated by these capabilities, this work proposes leveraging LLMs to automatically design reward functions for RL-based radio resource scheduling within open radio access network (RAN) environments. Unlike conventional approaches that rely on fixed reward structures, the proposed framework dynamically generates reward formulations tailored to current network conditions and slice requirements. By aligning the reward design with evolving SLA objectives, the approach enhances the adaptability and robustness of RL schedulers, enabling more efficient and versatile resource allocation in programmable and heterogeneous next-generation networks.

2. System Model

This work considers a single-input and single-output (SISO) configuration with a single base station operating at a carrier frequency f_{c} and providing service to $s = 1, 2, 3, \ldots, S$ RAN slices. The base station simultaneously serves $u = 1, 2, 3, \ldots, U$ users' equipment (UEs), where each UE u is equipped with a single antenna and is associated with exactly one slice s. Each slice s comprises a set of U_s UEs. Although the system model is specified for a SISO configuration, the proposed framework is directly applicable to other RF transmission schemes employing multiple-input and multiple-output (MIMO) without requiring modifications to the underlying system formulation.

The base station operates over a total bandwidth of B MHz, which is discretized into R resource blocks (RBs), representing the minimum radio resource allocation unit in the scheduling framework. The transmission time interval (TTI), denoted by t and measured in ms, defines the minimum temporal granularity considered by the scheduler. The inter-slice and intra-slice schedulers are executed over distinct time scales. The inter-slice scheduler operates with a coarser time-scale periodicity and is responsible for allocating the available RBs to the active network slices. Conversely, the intra-slice scheduler operates at every TTI, assigning the slice-specific RBs (previously allocated by the inter-slice scheduler) to the slice's UEs. In this work, we are going to focus on the inter-slice scheduler while assuming a round-robin algorithm for the intra-slice scheduler. The RRS system with RAN slicing distributes the R available RBs among the active slices S. A downlink RRS configuration is considered, in which the base station determines the set of RBs assigned to each UE. Nonetheless, the proposed methodology is directly applicable to the uplink direction as well.

2.1. RAN slicing

The 3rd Generation Partnership Project (3GPP) defines the radio resource management (RRM) policy structure to control the RAN slicing resources for each slice in the inter-slice allocation [3GPP 2024]. The *RRMPolicyRatio* structure defines how radio resources are shared or dedicated to specific network slices, identified by the single–network slice selection assistance information (S-NSSAI), within a gNB-DU. The *RRMPolicyRatio* structure defines the inter-slice scheduler dynamics through the attributes *rRMPolicyDedicatedRatio*, *rRMPolicyMinRatio*, and *rRMPolicyMaxRatio*. The attribute *rRMPolicyDedicatedRatio* defines the dedicated resource usage quota for the active RAN slices in the network, defining the dedicated radio resources in the network. These dedicated resources are exclusive to their allocated RAN slices, meaning that they can not be shared with other RAN slices, even if the target RAN slice is not using these dedicated resources. The dedicated resource quota Q is

$$\begin{aligned} Q_{\mathsf{ded}} &= \sum_{s=1}^{S} Q^{s}_{\mathsf{ded}} \\ \text{subject to} \quad & 0\% \leq Q_{\mathsf{ded}} \leq 100\%, \end{aligned} \tag{1}$$

where Q^{s}_{ded} is the dedicated resource quota for slice s determined by the inter-slice scheduler. As these resources are dedicated and exclusive to each allocated slice, the total dedicated resource quota cannot exceed 100%.

The attribute *rRMPolicyMinRatio* defines the minimum resource usage quota for RAN slices, defining the prioritized resources in the network. The prioritized resources are preferentially used by the associated RAN slices when they need to use them. When not used, these resources may be used by other RAN slices in the network. The minimum resource quota Q_{min} is

$$\begin{aligned} Q_{\mathsf{min}} &= \sum_{s=1}^{S} Q^{s}_{\mathsf{min}} \\ \text{subject to} \quad & Q^{s}_{\mathsf{ded}} \leq Q^{s}_{\mathsf{min}} \leq 100\%, \end{aligned} \tag{2}$$

where Q^{s}_{min} represents the minimum resource quota for slice s assigned by the inter-slice scheduler. Similarly to the dedicated resource quota, the total minimum resource quota cannot exceed 100%. Moreover, the minimum resource quota Q^{s}_{min} for slice s has to be greater or equal to the dedicated resource quota. Therefore, the prioritized resources are represented as $Q_{\mathsf{min}} - Q_{\mathsf{ded}}$.

The attribute *rRMPolicyMaxRatio* defines the maximum resource usage quota for the active RAN slices, defining the shared resources in the network. The shared resources do not provide any resource guarantees since they are not prioritized or dedicated to any RAN slice. The maximum resource quota is

$$\begin{aligned} Q_{\mathsf{max}} &= \sum_{s=1}^{S} Q^{s}_{\mathsf{max}} \\ \text{subject to} \quad & Q^{s}_{\mathsf{min}} \leq Q^{s}_{\mathsf{max}} \leq 100\%, \end{aligned} \tag{3}$$

where Q^s_{max} represents the maximum resource quota for slice s defined by the inter-slice scheduler in step n. The maximum resource quota Q^s_{max} has to be greater than or equal to the minimum resource quota Q^s_{min}. Unlike the dedicated and minimum resource quota, the maximum resource quota Q_{max} is not constrained to 100%.

The inter-slice scheduler is responsible for defining the dedicated, minimum, and maximum resource quota (Q^s_{ded}, Q^s_{min} and Q^s_{max}) for each active RAN slice in the network. Therefore, the inter-slice scheduler decision $\mathbf{Q}^{\mathsf{inter}}$ can be represented as a vector

$$\mathbf{Q}^{\mathsf{inter}} = \left[Q^1_{\mathsf{ded}}, Q^1_{\mathsf{min}}, Q^1_{\mathsf{max}}, \ldots, Q^S_{\mathsf{ded}}, Q^S_{\mathsf{min}}, Q^S_{\mathsf{max}}\right]. \tag{4}$$

3. LLM-Driven Reward Design for Intelligent RAN Slice Scheduling

The RL agents have been used in different inter-slice scheduler approaches for dealing with RAN slicing scenarios due to their flexibility to deal with different network objectives. These approaches design RL agents for specific scenarios, such as maximizing network metrics, fulfilling slice requirements, or giving higher priorities for predefined slices [Polese et al. 2022, Raftopoulos et al. 2024, Nahum et al. 2023, Nahum et al. 2026, Zangooei et al. 2023]. Therefore, there is a need for a different RL agent for each network scenario and objectives, with the network operator being responsible for selecting the correct scheduler approach for each scenario. In order to automate the identification of the network goals and generate RL agents for performing the inter-slice scheduler without human intervention, we propose an LLM-driven reward design for RL-based RAN slicing schedulers. We utilize an LLM agent to generate the reward functions for RL inter-slice schedulers based on operator network intents given in natural language.

Fig. 1 illustrates the overall process of inter-slice scheduling in a 5G Open RAN network using our proposed method. Initially, the network operator provides an input in natural language to an LLM agent, located at the non-RT RAN intelligent controller (RIC), describing the network intents for each slice, such as the network key performance indicators (KPIs) that each network slice should fulfill. The LLM agent reads the network operator input together with mobile network information and generates a reward function to an RL agent aiming to fulfill the network intents. Once the reward function is ready, our proposed method deploys an xApp with an RL agent implementing the designed LLM-driven reward function for controlling the inter-slice scheduler process. The RL agent xApp has a control loop with the E2 termination of the CU and DU functions, where it receives key performance measures from the mobile network through RIC indication reports and generate RRM policies for controlling the RAN slice resources through control messages.

The proposed method is composed of the LLM agent, which will be responsible for identifying the use case described in the network operator input, capturing the KPIs for each slice and generating a reward function that is capable of reaching the realized goals, but also can be successfully learned by the RL agent that is going to perform the inter-slice scheduler. Therefore, the proposed method needs to build a cooperation between the LLM and RL agents in order to successfully achieve the expressed network operator intents.

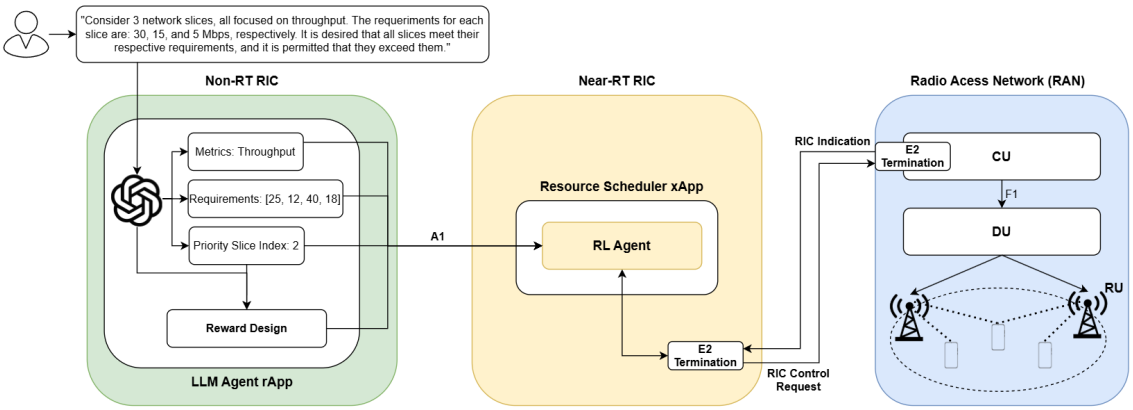


Figure 1. LLM-driven reward design for RAN slicing scheduler in the Open RAN scenario.

3.1. RL Scheduler

The RL agent operates as an intelligent decision-making entity embedded within the RAN control loop, continuously interacting with the network environment. At each scheduling interval, the agent observes the current state of the RAN, which reflects the performance experienced by active users and network slices, and takes actions that define the inter-slice resource quotas for each slice as defined in Eq. 4. This interaction follows the classical RL paradigm of perception, decision, and feedback, where the RAN responds to the agent's actions by updating its state in the next time step. Within this context, the RL agent is responsible for determining how radio resources are distributed across multiple RAN slices, ensuring that heterogeneous service requirements are adequately addressed.

We employ a proximal policy optimization (PPO)-based RL approach, a widely used algorithm for continuous control problems that combines the robustness and stability of trust-region techniques with a more accessible implementation [Schulman et al. 2017]. For the RL-based inter-slice scheduler, we model the problem as a MDP defined by the tuple $(S, A, \mathsf{RW}, P, \rho_0)$. Here, S denotes the set of all feasible states, each encoding information about relevant network metrics. The set A contains all admissible actions of the inter-slice scheduler, RW specifies its reward function, P is the state transition probability function, and ρ_0 is the initial state distribution [Schulman et al. 2017]. At each time step t, the agent observes the current state s_t, selects an action a_t that corresponds to an inter-slice resource allocation decision, and transitions to the next state s_{t+1} while receiving a reward RW. This reward is a scalar signal evaluating how effective the chosen action was in meeting scheduler goals, and the agent's objective is to maximize the expected long-term cumulative reward [Schulman et al. 2017].

3.1.1. Observation space

We define the observation space as a composition of performance indicators associated with each RAN slice, specifically throughput and latency, which jointly characterize the quality of service currently being delivered. The observation space is

$$\mathbf{O} = [e_1, e_2, \ldots, e_S, \ell_1, \ell_2, \ldots, \ell_S], \tag{5}$$

where e_s and ℓ_s are the throughput and the latency of the s-th slice, respectively. The observation space has a $2S$-dimension, with S being the number of active slices in the network. In the scope of this work, we are focusing on throughput and latency slice metrics since they encompass the most important metrics utilized in related work [Nahum et al. 2026]. Still, the observation space could be expanded to contain additional slice metrics such as the packet loss rate.

3.1.2. Action space

The action space is the RL agent output that represents the percentage of available resource blocks assigned to each slice at a given time interval. By mapping observed network conditions to resource allocation decisions, the agent learns a policy that adapts to traffic fluctuations and varying slice demands. The action space is an S-dimensional array defined as

$$A = [p_1, p_2, ..., p_S], \ \sum_{s=1}^{S} p_s = 1, \tag{6}$$

where p_s is the percentage of resource blocks given to the s-th slice at a given time. The RL agent actions set the inter-slice scheduler decision $\mathbf{Q}^{\mathsf{inter}}$ values for each slice in order to produce an *RRMPolicyRatio* structure to be implemented in the gNB scheduler.

Besides the observation and action space, the reward function plays a central role in guiding the agent's learning and ultimately shaping its behavior. By translating high-level performance objectives into a scalar feedback, the reward incentivizes the agent to favor certain trade-offs, such as fulfilling slice intents. Flexibility in reward design is therefore crucial, as different operational goals may require fundamentally different optimization strategies. However, most existing works in the literature rely on static, fixed reward formulations, which limits the agent's ability to cope with diverse requirements, evolving network conditions, or conflicting objectives. Such rigid designs often fail to identify the most appropriate scheduling strategy for a given scenario, highlighting the importance of adaptable reward mechanisms that can better align the RL agent's decisions with dynamic network and operator goals.

3.2. LLM agent

In this framework, the integration of LLMs is justified by their ability to interpret and process high-level semantic information. The LLM agent is utilized as a translation interface, taking input prompts that encompass network configurations, performance metrics, and requirements expressed in natural language by the network operator. Based on this textual input, the model autonomously generates executable Python code that defines the reward function. This code is then dynamically invoked within the RL environment, enabling the RL agent to optimize resource allocation and slicing policies in the O-RAN architecture according to the specified high-level objectives.

To create this customized reward function, we utilize an LLM agent to analyse the network operator input text. Internally, the LLM converts the input text into vectors and generates new text probabilistically [Radford et al. 2019], given by

$$P(\mathbf{y} \mid \mathbf{x}) = \prod_{\mathbf{t}} P(y_t \mid \mathbf{x}, \mathbf{y}_{<\mathbf{t}}), \tag{7}$$

where $\mathbf{x}$ are the input text vectors, $\mathbf{y}$ are the complete output text vectors, and $\mathbf{y_{<t}}$ are the output text vectors prior to the t-th iteration of the text generation. The selection of each new text y_t within the model's vocabulary is governed by vector semantics. The LLM maps tokens into a high-dimensional embedding space. During inference, the choice of the next term depends on the similarity between the current hidden state and the candidate vectors in the vocabulary. The fundamental metric for this alignment is the cosine similarity, calculated between the context vector generated by the input and previous outputs (u) and the vector of each candidate token (v):

$$\text{sim}(u, v) = \frac{u \cdot v}{|u||v|}. \tag{8}$$

Therefore, the output is dynamically constructed, and the operator's prompt defines the initial context, while each previously generated token influences the probability of the subsequent ones. This ensures that the final reward function code is syntactically coherent and strictly aligned with the slice's requirements extracted from operator input.

The reward function generated by the LLM serves as a communication bridge between high-level network objectives and the RL agent's operational behavior by translating qualitative requirements into a quantitative mathematical formulation. Unlike static, predefined reward functions, this approach introduces dynamism into the system, enabling the generation of tailored reward structures for specific scenarios, such as penalizing throughput violations and giving extra resources to slices with higher priorities. Consequently, when a new request from the network operator is received, the LLM agent interprets the natural language input and synthesizes the corresponding custom code. This function is then seamlessly passed to the Resource Scheduler xApp, which integrates this reward function into the RL agent's training process using real-time data from the physical network to align the agent's policy with the operator's intent.

Fig. 2 shows the reward function generation process divided into four-stages. First, the LLM performs use-case classification to identify the operational profile that best aligns with the user's intent. If the operator's objective is strictly to satisfy standard network requirements, the model classifies the input as case 1. Conversely, if the intent involves prioritizing a specific slice over others, the model assigns it to case 2. Second, it executes parameter extraction, where the model identifies latency and throughput requirements from the prompt. These metrics are essential for both formulating the reward function and monitoring whether the agent's performance converges toward the requested targets. Furthermore, the model identifies an auxiliary control variable, denoted as K. In Case 2, this variable acts as a scalar that identifies and parameterizes the specific slice to be prioritized during resource allocation. Conversely, for Case 1, where explicit prioritization is not required, this variable is omitted. Finally, by integrating these technical constraints and variables, the LLM synthesizes the reward function tailored for the reinforcement learning training process.

The architecture translates high-level user directives into reward function parameters for the RL agent. This work formulates two different use cases to exemplify the different scenarios that the LLM agent could handle to personalize the RL agent for inter-slice scheduling, The first use-case focuses on fulfilling the RAN slicing requirements, performing scheduler actions to meet the KPIs requirements expressed by the network operator. The second use-case also focuses on fulfilling slice requirements while, addi-

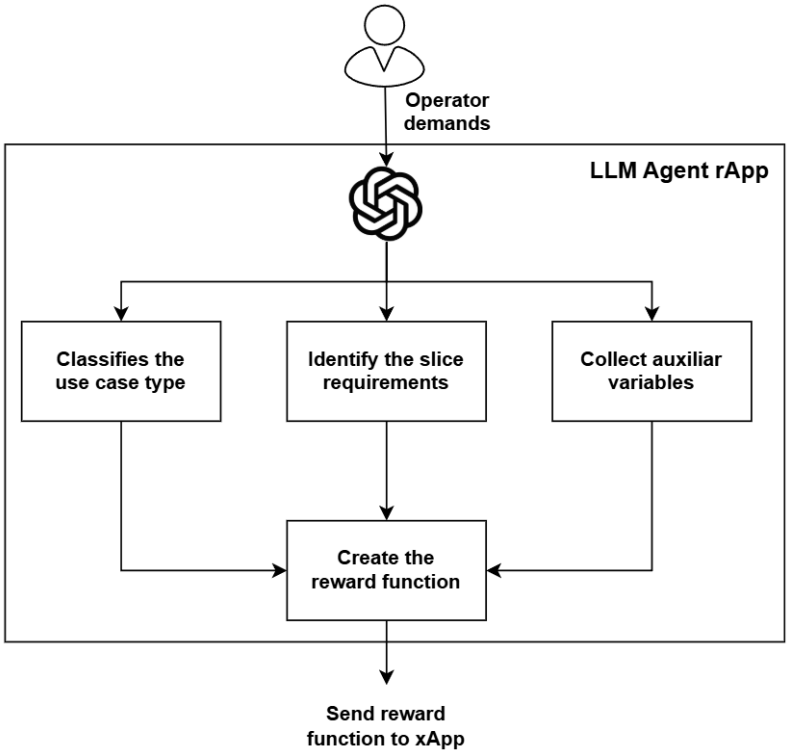


Figure 2. Representation of the steps performed by LLM to generate the reward function.

tionaly, provides higher priorities for specific network slices. In this use case, the slice with higher priority should receive extra resources when all the other slices are already fulfilling their requirements.

3.2.1. Use case 1 - Fulfilling the requirements

A common scenario in radio resource scheduling consists of allocating all available resource blocks in such a way that the requirements of all network slices are satisfied [Nahum et al. 2026, Nahum et al. 2023]. In this context, the primary objective of the scheduler is feasibility, that is, guaranteeing that each slice meets its predefined performance constraints. The allocation process is therefore driven by the need to ensure compliance with service requirements rather than by strict optimization of resource efficiency beyond those requirements. Under this assumption, there is no explicit concern regarding the excess of resources allocated to a given slice once its requirements have been fulfilled. For instance, if a slice specifies a minimum throughput requirement of 10 Mbps, this use case does not distinguish between an observed throughput of 12 Mbps or 13 Mbps, since both values exceed the required threshold. Any additional performance above the minimum requirement is treated equivalently and does not contribute further benefits in the scheduling decision process.

For this work, the reward functions are calculated based on the metrics of the observation space $\mathbf{O}$ and its requirements, defined as

$$\mathbf{\Psi} = [\psi_1, \psi_2, ..., \psi_S]. \tag{9}$$

where ψ_s is the throughput requirement of the s-th slice. The normalized difference between the observed throughput and the slice requirement is called the performance δ_s of the s-th slice and its given by

$$\delta_s = \frac{e_s - \psi_s}{B^{max}}, \tag{10}$$

where B^{max} is the UE's maximum throughput buffer size . The collection of performances for all S slices is defined as

$$\delta = [\delta_1, \delta_2, ..., \delta_S], \tag{11}$$

For the first use case, the agent's reward function can be formally described as

$$r = \sum_{s}^{S} \mathsf{min}\left(0, \frac{\delta_s}{\psi_s}\right). \tag{12}$$

Eq. 12 operates by evaluating, for each network slice, whether the observed performance metric satisfies the corresponding requirement, with the comparison expressed as a ratio normalized by the buffer and requirements values. The B^{max} factor is applied to normalize the reward scale and ensure numerical stability during the algorithm's policy updates. When the observed metric exceeds the requirement, this normalized ratio becomes positive, and the subsequent minimum operation between this ratio and zero yields a null contribution to the reward. In contrast, when the normalized ratio is negative, it indicates that the observed performance falls below the required threshold, and this negative ratio is directly incorporated into the reward. This design choice ensures that only violations of requirements are penalized, while performance above the requirement does not generate additional gains. Moreover, this formulation prevents negative contributions associated with underperforming slices from being offset by positive contributions from slices that exceed their requirements.

3.2.2. Use case 2 - Prioritize a slice

This use case aims to meet the requirements of all slices, except for one slice, which will have its metric optimized. Every time all the slices are fulfilling their requirements, and there are still available resources, the slice with higher priority should receive additional resources to improve its application's performance. The agent's reward function can be formally described as

$$r = \sum_{s=1}^{S} \xi_s, \tag{13}$$

where ξ_s is given by

$$\xi_s = \begin{cases} -3\delta_s \tanh \delta_s, & \delta_s < 0 \text{ and } s \neq k \\ 0, & 0 \leq \delta \leq 2 \text{ and } s \neq k \\ -\delta_s \tanh \delta_s, & \delta > 2 \text{ and } s \neq k \\ |\delta_s| \tanh \delta_s, & s = k \end{cases}, \tag{14}$$

where k is the slice to be optimized. In this equation, for all slices other than k, the agent is penalized by the absolute value of the deviation from the requirement. This incentivizes the scheduler to maintain the performance of these slices as close as possible to the requested threshold, avoiding both under-allocation and the waste of resources that could be redirected.

In contrast, the term associated with slice k does not utilize the minimum operation, allowing observed performance values above the requirement to generate a direct positive contribution to the total reward. Thus, the agent is penalized if the prioritized slice falls below the requirement and is incrementally rewarded as the performance of k exceeds it, provided that the non-prioritized slices maintain their values close to the established requirement range.

4. Results

This work implemented the proposed RL agent as an xApp using the official O-RAN software community (SC) *ricxappframe* library. The agent is deployed in the OSC Near-RT RIC and utilizes the PPO [Schulman et al. 2017] algorithm through the Ray RLlib library. Simultaneously, the LLM operates as an external agent that communicates directly with the RL xApp. The LLM agent is powered by the GPT-5.2 model, accessed through the OpenAI API. To facilitate a realistic closed-loop evaluation, the simulation uses the New Open RAN Interface (NORI) [Oliveira et al. 2025] module to integrate the NS-3 network simulator with the Near-RT RIC. NORI incorporates an E2 termination component in the simulated gNBs, enabling the exchange of telemetry and control messages through the *e2sim-lib* library. This modular architecture supports the E2SM-KPM service model, allowing the network to report real-time performance indicators, such as slice throughput and user-level metrics, without requiring intrusive modifications to the NS-3 core. Consequently, the RL-based xApp continuously receives standard-compliant telemetry to apply resource allocation decisions guided by the reward functions generated by the LLM.

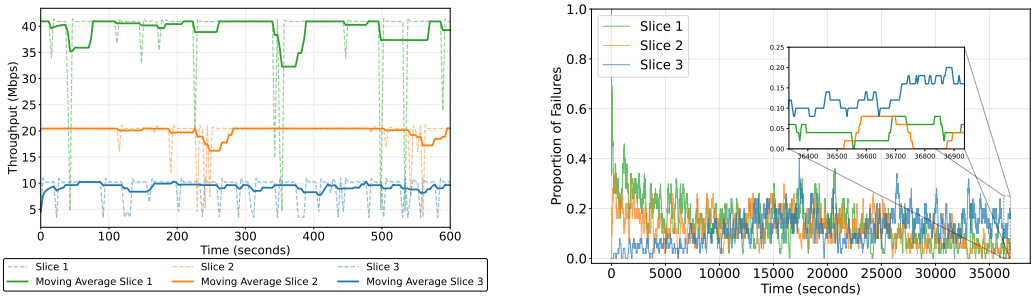
The simulation implements a mobile network scenario with a single 5G base station using the UMa 3GPP channel model [Zhu et al. 2019] with a single isotropic antenna model. The base station utilizes a 50 dBm transmission power, a 100 MHz bandwidth, a central frequency of 3.5 GHz, and the numerology 1. There are 11 UEs in the system, where each UE utilizes a single isotropic antenna model with a 23 dBm transmission power. We consider three different RAN slices. The first slice contains 3 UEs with a 30 Mbps throughput requirement, the second slice contains 4 UEs with a 15 Mbps throughput requirement, and the third slice contains 4 UEs with a 5 Mbps throughput requirement. The proposed method is evaluated in two different scenarios where a network operator requests different intents. In the first scenario, the intent focuses on defining the throughput requirements for each slice, while in the second scenario, besides the intents for fulfilling the slice requirements, there is also an indication to prioritize one of the slices in case there are available resources.

For the first evaluation scenario, the network operator provides the following intent in natural language "*Consider a situation where you have three network slices, each with its own performance requirements. The requirements for each slice are: 30, 15, and 5 Mbps, respectively. It is desired to comply with the requirements, and there is no interest in saving resources*". The LLM agent is responsible for interpreting the given intent, clas-

sifying the use-case, and generating a reward function to be utilized in the RL agent in the xApp. Fig. 3 shows the LLM agent output, which defines the RL agent reward function in the xApp. The LLM agent correctly identified the operator intent requirements and generated a reward function that aims to fulfill the specified slice requirements obtained from the text provided by the network operator.

```python
import numpy as np
def reward_function(throughput, requirements, buffer):
    delta = (throughput - requirements) / buffer
    reward = 0
    for s in range(delta):
        reward += (np.minimum(0, delta[s] / requirements[s])
    return reward
```

Figure 3. LLM-generated code for the first use case.



(a) Throughput evaluation.

(b) Proportion of failures.

Figure 4. Evaluation of the slice throughput results and proportion of failures for use case 1.

Once the LLM output was generated, it automatically deploys the RL agent xApp with the designed reward function. Fig. 4(a) shows the RL agent xApp performance when controlling the inter-slice scheduler. The obtained moving average throughput for each slice shows that the proposed RL agent, utilizing the LLM-generated reward function, was able to keep the slice's throughput over their specified requirements. When analyzing the instantaneous throughput, sometimes the requirements were not fulfilled due to the ordinary network variations. Still, the proposed method was able to adjust its scheduler decisions in order to fulfill the requirements in most of the analyzed simulation period, indicating that the scheduler is capable of meeting the defined service-level constraints for the considered scenario. Fig. 4(b) shows the proportion of failures (violations) to fulfill slice requirements over a window of 50 samples. With the training evolution, the number of violations for all the slices is reduced due to the improvement in the RL agent policy over time. Slice 1 has the highest throughput requirement, and then it is the slice that is more susceptible to violations. Still, when analysing the average throughput from Fig. 4(a) and Fig. 4(b), the instantaneous requirement violations detected on the proportion of failures is not sufficient to put the average throughput below the intent requirement, attesting to the effectiveness of our proposed method at fulfilling the operator network intents.

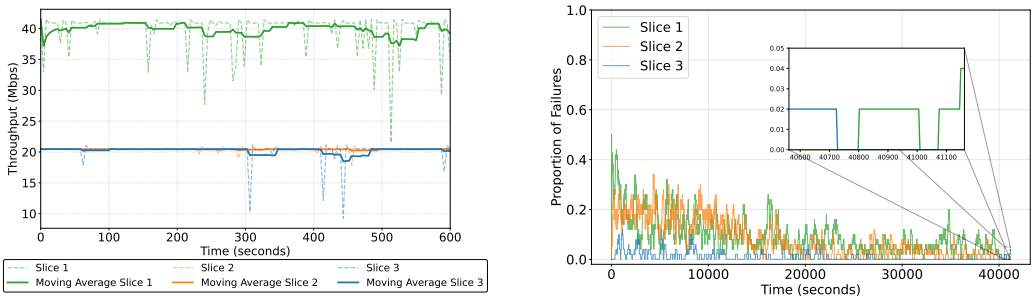
For the second evaluation scenario, the network operator provides the following intent in natural language: *"Consider a situation where you have three network slices, each with its own performance requirements. The requirements for each slice are: 30, 15 and 5 Mbps, respectively. You want to meet the requirements of the first and second slice, without exceeding or falling below them, to maximize performance in the third slice, in which the performance is desired to be as great as possible"*. Similar to the first evaluation scenario, the LLM agent generates a reward function for the RL agent xApp to guide the proposed method learning process. In this evaluated scenario, besides the slice requirements fulfillment, the slice 3 has a higher priority to receive additional resources in relation to other slices. Fig. 5 shows the generated reward function where the LLM agent correctly identified the slice to be prioritized and provided a greater reward when it receives additional resources (considering all the other slice requirements are fulfilled).

```python
import numpy as np
def reward_function(throughput, requirements, buffer, k):
    delta = (throughput - requirements) / buffer
    reward = 0
    for s in range(len(delta)):
        if s == k:
            reward += np.abs(delta[s]) * np.tanh(delta[s])
        else:
            if delta[s] < 0:
                reward -= 3 * delta[s] * np.tanh(delta[s])
            elif 0 <= delta[s] <= 2:
                reward += 0
            else:
                reward -=  delta[s] * np.tanh(delta[s])
    return reward
```

Figure 5. LLM-generated code for the second use case.

Fig. 6(a) shows the RL agent xApp performance when controlling the inter-slice scheduler for the second evaluation scenario. The obtained moving average throughput for each slice shows that, similarly to the first evaluation scenario, the proposed method was able to keep the slice's throughput over its specified requirements. Moreover, the slice with higher priority (slice 3) obtained additional resources in relation to the previous evaluation scenario due to the network operator's intent goals expressed in the natural language input. Fig. 6(b) shows the proportion of failures (violations) over the training of the model, which shows a reduction of the number of violations over time. When analysing Fig. 6(a) and Fig. 6(b), the proposed method was able to keep the average throughput over the requirements even with the perceived instantaneous violations.

When analysing use-cases 1 and 2, the proposed method was able to correctly identify and interpret the network operator intents and translate them to reward functions able to fulfill their specified requirements. This behavior provides a flexible method to deal with different network scenarios and intents that could automatically generate scheduler policies based solely on network operator intents without the need for additional human interventions.



(a) Throughput evaluation. (b) Proportion of failures.

Figure 6. Evaluation of the slice throughput results and proportion of failures for use case 2.

5. Conclusion

This work explored the potential of leveraging LLM to assist in the design of reward functions for RL agents responsible for the inter-slice radio resource scheduling in Open RAN environments. By introducing a mechanism in which the reward formulation can be dynamically generated according to the characteristics of the network scenario and slice requirements, the proposed approach aims to address a key limitation of many RL-based schedulers: their dependence on manually designed rewards that are often tailored to a specific environment. The results and discussion suggest that integrating LLM-driven reward generation with RL scheduling policies is a promising direction for improving the adaptability and flexibility of resource allocation mechanisms in complex and heterogeneous RAN scenarios.

Nevertheless, further investigation is required to fully validate the effectiveness and robustness of this approach. Future work should incorporate well-established baselines to enable a rigorous comparison between the proposed method and existing approaches in the literature, including current state-of-the-art resource allocation strategies, and also evaluate the proposed framework across a broader set of use cases and network conditions, including different traffic models, radio configurations, and simulation parameters. In particular, it is important to assess the ability of the LLM to correctly identify and encode the relevant objectives for a wide range of scenarios, ensuring that the generated reward functions remain aligned with the system requirements. Additionally, more challenging environments should be considered, such as scenarios with a significantly larger number of user equipments and network slices, in order to better understand the scalability limits of the technique and the capability of the RL agents to maintain efficient scheduling performance under highly dynamic and congested network conditions.

Acknowledgment

This work was partially financed by the Innovation Center, Ericsson Telecomunicações S.A., Brazil; Brasil 6G project (RNP/MCTI grant 01245.010604/2020-14); OpenRAN Brazil - Phase 2 project (MCTI grant Nº A01245.014203/2021-14); Universal (CNPq grant 405111/2021-5); Project Smart 5G Core And MUltiRAn Integration (SAMURAI) (MCTIC/ CGI.br/FAPESP under Grant 2020/05127-2); INCT STREAM (CNPq grant Nº 409179/2024-8).

References

3GPP (2024). Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3. Technical specification (TS) 28.541, 3rd Generation Partnership Project (3GPP). Version 17.15.0.

Afolabi, I. et al. (2018). Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453.

Calabrese, F. D. et al. (2018). Learning radio resource management in RANs: Framework, opportunities, and challenges. *IEEE Communications Magazine*, 56(9):138–145.

Nahum, C. V., D'Oro, S., Batista, P., Both, C. B., Cardoso, K. V., Klautau, A., and Melodia, T. (2026). Intent-based radio scheduler for ran slicing: Learning to deal with different network scenarios. *IEEE Transactions on Mobile Computing*, 25(3):3229–3246.

Nahum, C. V., Lopes, V. H., Dreifuerst, R. M., Batista, P., Correa, I., Cardoso, K. V., Klautau, A., and Heath, R. W. (2023). Intent-aware radio resource scheduling in a RAN slicing scenario using reinforcement learning. *IEEE Transactions on Wireless Communications*, pages 1–1.

Oliveira, A. A. M. d., Albuquerque, J. P., Nahum, C. V., Campos, D., Cardoso, K. V., Klautau, A., and Rezende, J. F. d. (2025). Enabling NS-3 simulations integrated with latest versions of open RAN near-RT RICs. In *Anais do XLIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*. Sociedade Brasileira de Telecomunicações.

Polese, M. et al. (2022). ColO-RAN: Developing machine learning-based xApps for open RAN closed-loop control on programmable experimental platforms. *IEEE Transactions on Mobile Computing*.

Quy, V. K. et al. (2023). Innovative trends in the 6G era: A comprehensive survey of architecture, applications, technologies, and challenges. *IEEE Access*, 11:39824–39844.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raftopoulos, R., D'Oro, S., Melodia, T., and Schembra, G. (2024). DRL-based latency-aware network slicing in O-RAN with time-varying SLAs. *arXiv preprint arXiv:2401.05042*.

Schulman, J. et al. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Yu, W., Gileadi, N., Fu, C., Kirmani, S., Lee, K.-H., Arenas, M. G., Chiang, H.-T. L., Erez, T., Hasenclever, L., Humplik, J., et al. (2023). Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*.

Zangooei, M. et al. (2023). Flexible ran slicing in open ran with constrained multi-agent reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 42(2):280–294.

Zhu, Q. et al. (2019). 3GPP TR 38.901 Channel Model. *Wiley 5G Ref: The Essential 5G Reference Online*, pages 1–35.