From Simulation to Practice: Generalizable Deep Reinforcement Learning for Cellular Schedulers

Petteri Kela
Nokia, Espoo, Finland
petteri.kela@nokia.com

Bryan Liu
Nokia Bell-Labs, Massy, France
bryan.liu@nokia-bell-labs.com

Alvaro Valcarce
Nokia Bell-Labs, Massy, France
alvaro.valcarce_rial@nokia-bell-labs.com

Abstract

Efficient radio packet scheduling remains one of the most challenging tasks in cellular networks, and while heuristic methods exist, practical deep learning–based schedulers that are 3GPP-compliant and capable of real-time operation in 5G and beyond are still missing. To address this, we first take a critical look at previous deep scheduler efforts. Secondly, we enhance State-of-the-Art (SoTA) deep Reinforcement Learning (RL) algorithms and adapt them to train our deep scheduler. In particular, we propose a novel combination of training techniques for Proximal Policy Optimization (PPO) and a new Distributional Soft Actor-Critic Discrete (DSACD) algorithm, which outperformed other variants tested. These improvements were achieved while maintaining minimal actor network complexity, making them suitable for real-time computing environments. Furthermore, entropy learning in SACD was fine-tuned to accommodate resource allocation action spaces of varying sizes. Our proposed deep schedulers exhibited strong generalization across different bandwidths, number of Multi-User MIMO (MU-MIMO) layers, and traffic models. Ultimately, we show that our pre-trained deep schedulers outperform their heuristic rivals in realistic and standard-compliant 5G system-level simulations.

1 Introduction

Wireless systems have evolved and are capable of serving vast amounts of data to many users over large areas. This progress stems from splitting wireless channels into resources across multiple dimensions: code, frequency, time, antennas, beams, and spatial layers. In 4G, 5G, and, likely, 6G, the radio packet scheduler dynamically allocates these resources to users, optimizing network performance, fairness, and spectral efficiency. We refer to these functions as Time Domain Scheduling (TDS), Frequency Domain Scheduling (FDS), and Spatial Domain Scheduling (SDS). They are NP-hard combinatorial problems, often solved with heuristics (see [16]) that lack optimality guarantees and scale poorly. As we transition to 6G, the number of radio resources at our disposal will continue to grow (e.g., larger Multiple Input Multiple Output (MIMO) arrays, larger bandwidths, diverse user devices, etc.). This increasing complexity strains traditional scheduling methods, creating a compelling need for new approaches. Machine Learning (ML) offers a promising path forward for several reasons:

- **Scalability**: ML shifts the computational cost away from inference. Trained models can often make complex scheduling decisions cheaper.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Workshop on AI and ML for Next-Generation Wireless Communications and Networking.

- **Flexibility**: ML models naturally consider multimodal factors, such as Channel State Information (CSI), User Equipment (UE) priorities, Quality of Service (QoS) profiles, etc.
- **Adaptability and customization**: ML models learn to adapt to changing channel conditions, traffic patterns, and user demands, a critical capability as networks become more dynamic.

The ability of packet schedulers to react quickly to load changes while considering all other inputs is hard to balance and build into heuristic algorithms. For instance, some radio schedulers may select the best beam based on beam-specific CSI reports. Grounding this decision also on the traffic load per beam (even if the beam has worse RF conditions) can sometimes yield net capacity gains [15]. Such load changes can happen very fast (especially in the higher FR2 frequencies at 24.25 GHz to 52.6 GHz) when UEs move across beams very quickly. Detecting such rapid changes requires additional subroutines built into heuristic schedulers, which increase their computational complexity and make debugging even harder. Instead, ML-based deep schedulers trained on real scenarios excel at detecting such hidden patterns in the data, often before the event occurs. While beam selection is a crucial aspect of scheduling in Grid of Beams (GoB) systems, our current paper addresses the broader challenge of resource allocation in a dynamic Zero Forcing (ZF)-based MU-MIMO beamforming context, particularly well suited for Frequency Range 1 (FR1) (below 6 GHz) and the emerging 6G mid-bands (7-15 GHz). In the remainder of this paper, we will refer to ML-based radio packet schedulers as *Deep Schedulers*. We provide an overview of current SoTA heuristic schedulers and previous work on deep schedulers. We then detail our own RL-based deep scheduler designs, focusing on schedulers based on Soft Actor-Critic Discrete (SACD) and PPO, which achieve competitive MU-MIMO gains while maintaining manageable computational complexity. Finally, we compare various approaches and discuss the simulation results obtained with a 5G New Radio (5G NR)-compliant System-Level Simulator (SLS).

2 Background

To contextualize our work, we first review existing scheduling approaches (see Table 2, Appendix A, for a summary). Our focus is on developing deep schedulers suitable for practical deployment in 3rd Generation Partnership Project (3GPP)-compliant cellular networks (Fifth Generation (5G) and beyond), as non-standard approaches have limited real-world applicability. The following review highlights key limitations of the prior art concerning complexity, scalability, standard adherence, and performance, motivating the need for the advancements presented in this paper.

2.1 Previous Deep Scheduler Efforts

The field of ML-based radio resource allocation has a rich history, with numerous designs proposed over the years [23]. However, the complexity of wireless scheduling makes optimal solutions computationally challenging, hindering the acquisition of reliable labeled data for supervised learning models. The time-sequential nature of scheduling decisions lends itself to hidden Markov chains, leading most deep schedulers to employ RL.

In [4], a Downlink (DL) Frequency Domain (FD) deep scheduler based on Double Deep Q-Network (DDQN) was proposed, trained to maximize a composite reward for bitrate and fairness via non-contiguous allocation of 5G-compliant Resource Block Groups (RBGs). However, it ignored subband-specific Channel Quality Indicators (CQIs), missing frequency selectivity opportunities. Its Q network had two Fully Connected (FC) hidden layers of size 128 with ReLU activations, performing comparably to a Proportional Fair (PF) baseline. The paper did not discuss inference time or computational complexity, which is crucial for 5G schedulers making decisions every slot with sub-ms latency.

The deep scheduler proposed in [6] addresses SDS in Uplink (UL) MU-MIMO settings using Soft Actor-Critic (SAC) and K-Nearest Neighbors (KNN), with the aim of maximizing Spectral Efficiency (SE) and fairness among users. It incorporates UE grouping based on channel correlations and is evaluated with simulated and real-world data. However, the proposed method deviates from the 5G NR uplink allocation requirements, which mandate contiguous or almost contiguous Resource Block (RB) allocations with RBG granularity [2]. This discrepancy in the uplink allocation strategy limits the direct applicability of the approach within standard-compliant 5G NR systems.

Other efforts to reduce the action space size of deep schedulers in MU-MIMO settings include Deep Q-Networks (DQNs) with Action Branching (AB) [13]. However, this method does not scale effectively to practical massive MIMO systems, as it requires a large action space even with just two MIMO layers, involving N_a actions and 3.6 million trainable parameters. Additionally, RL-based methods incorporating Monte Carlo Tree Search (MCTS) have also been explored [20].

More recently, a hybrid FD deep scheduler was proposed [19], where the UEs are selected heuristically according to a weighted priority metric. A PPO agent is trained to learn the heuristic's priority weights to minimize delay and packet loss. The PPO agent used five hidden layers of size 64. By leveraging the priority-based heuristic, the priority weights can change slowly and the inference pass does not need to be executed in each slot. Such an approach overcomes most implementation challenges, but sacrifices some dynamicity and may lead to suboptimal decisions due to the inherent limitations of the priority-based heuristic.

Recognizing the large combinatorial action space that radio schedulers confront, the authors in [7] have also opted for a policy gradient method such as DDPG (DDPG), which uses an Actor-Critic (AC) architecture. The aim is to train a ML model for aiding the scheduler to estimate an adequate allocation size N_{RBs} and Modulation and Coding Scheme (MCS) for each UE under virtualized Radio Access Network (vRAN) computational constraints. This design considers the UE-specific uplink Signal-to-noise Ratio (SNR) and the congestion of the cloud platform's CPUs, and it rewards the model for maximizing successful data decoding, thus promoting high bitrate. The authors claim significant gains in spectral efficiency over a Round Robin (RR) baseline.

Despite these efforts, none are practical enough for deployment in commercial cellular networks. We argue that a practical Deep Scheduler should possess the following traits:

- **Sub-ms latency**: Light inference passes.
- **Diverse traffic performance**: Effective handling of bursty and Full Buffer (FB) traffic.
- **SDS**: Support for MU-MIMO in massive MIMO systems.
- **Rank-aware scheduling**: Maintains performance across channels with varying diversity.
- **Frequency-selective scheduling**: In a standards-compliant manner.

Our current work focuses on FDS and SDS challenges. Extending similar ML-based principles to TDS functions could enable a fully integrated Deep Scheduler, including:

- Logical channel prioritization.
- Physical Downlink Control Channel (PDCCH) & Physical Downlink Shared Channel (PDSCH) co-scheduling.
- QoS-driven scheduling: Effective handling of Guaranteed Bit Rate (GBR) and non-GBR traffic, supporting standard QoS profiles.

3 Practical Deep Scheduler Implementation and Analysis

We train our ML-based schedulers with on-policy and off-policy RL algorithms combining several recent advances in RL research. We have evaluated two different designs for RL-based scheduling. Either looping through all RBGs and MU-MIMO user layers or increasing our input and output layers to accommodate all available RBGs and looping only through MU-MIMO layers to reduce computational complexity. These Single Loop Deep Scheduler (1LDS) and Double Loop Deep Scheduler (2LDS) designs are described in Appendix B. We describe our modified algorithms based on PPO and SACD in Appendices C and D, respectively. Performance comparisons are made against the baseline and the PF greedy scheduler based on the exhaustive search defined in Appendix E.

All deep schedulers were trained and evaluated using a proprietary enterprise-grade 3GPP-compliant 5G NR system level simulator. To assess deployment readiness, we further validated the trained models on commercial Base Transceiver Station (BTS) hardware under real-time conditions. These tests confirmed that the proposed approach meets practical latency requirements, reinforcing its applicability to operational networks. Detailed results are omitted due to confidentiality.

3.1 Evaluation

The key simulation parameters are collected in Table 4, Appendix H, where the focus is on downlink non-contiguous (Type 1) allocation. This enables unrestricted frequency-selective scheduling with RBG granularity. We assume the availability of sub-band CQI and Precoder Matrix Indicator (PMI) reports at the gNB and limit the maximum user-reported Rank Indicator (RI) to 2. To assess the generalization capabilities of our deep schedulers, we evaluated performance via separate simulations for each of the two traffic models (FB and FTP3 bursty), each employing 210 users to match the per traffic type scale used during training.

3.2 Training

Our on-policy and off-policy training methods employed a centralized approach, where a single RL agent interacts with the multi-cell system-level simulator. At each Transmission Time Interval (TTI), the agent collects state-action-reward-next state tuples from all simulated gNodeBs (gNBs) and updates a shared model. This yields a generalized model that can be used independently by all BTSs. This design choice was motivated by its faster convergence and superior simulation performance compared to distributed training, while ensuring that the resulting actor models are directly deployable on real 5G BTS hardware without additional fine-tuning.

To expedite training, models are trained on a reduced system bandwidth with fewer MU-MIMO layers, as detailed in Table 4, Appendix H. This simplification was necessary to be able to train 2LDS models for 18 RBGs. Then, to promote generalization, we trained the models under mixed traffic conditions, utilizing a total of 420 users, where 210 users had FB traffic, and 210 users had FTP Model 3 traffic (see Table 4, Appendix H). Carefully tuned hyperparameters for both methods are shown in Tables 5 and 6 of Appendix H. Note that the replay buffer size for PPO refers to the number of state-action pairs, in contrast to the traditional experience replay buffers.

For on-policy training, we adopt PPO enhanced with expert guidance. Specifically, we incorporate a Jensen-Shannon Divergence (JSD) loss term to align the agent's policy with a PF expert during early training, accelerating convergence and improving stability. This hybrid approach combines the robustness of PPO with imitation learning benefits, enabling the agent to learn effective scheduling strategies under limited sample diversity.

On the off-policy RL front, and as noted by [9], distributional RL does not universally outperform non-distributional variants. In our experiments, this effect was architecture-dependent: the 1LDS design, with its large multi-branch output layer and higher variance in action-value estimates, benefited from distributional critics, whereas the 2LDS design (making finer-grained decisions per RBG) showed no measurable gain from distributional modeling. Consequently, 1LDS was trained with DSACD, while 2LDS used SACD.

Fig. 1 illustrates smoothed learning curves for the best-performing models. It should be mentioned that the target entropy values β differed significantly between the 1LDS and 2LDS models. Although 2LDS models performed best with a low β of 0.4, emphasizing the exploitation of learned strategies, 1LDS models required a high β of 0.999, potentially due to their limited capacity to benefit from increased exploration. Additionally, 1LDS models, with their larger multi-branched output layers, needed more training samples to match the baseline scheduler's performance. This convergence occurred around the 400th TTI, at which point roughly 450,000 training $sars'$ tuples had been collected and used for training. Training samples (state-action-reward-state tuples) are collected starting only after the 100th TTI to allow the simulation to move past the initial transient phase while gNB transmit buffers reach a representative state. Specifically, considering the training parameters (Max UE layers $|L| = 4$ and $N_{RBG} = 18$, see Table 4, Appendix H) and the 21-cell deployment, one sample is generated for each scheduling decision per layer, per RBG, and per gNB, resulting in $4 \times 18 \times 21 = 1512$ samples collected across the network in each subsequent TTI. This collection rate aligns with the accumulation of approximately $450,000$ tuples by the 400th TTI.

The training performance of all models has been compared using geometric mean throughput. This is the Key Performance Indicator (KPI) that best captures proportional fairness, aligning with our primary goal for FDS/SDS scheduling phases. DSACD was the only method to surpass the baseline scheduler in the 1LDS architecture, making it the preferred choice for this configuration. For 2LDS architectures, SACD outperformed all others.

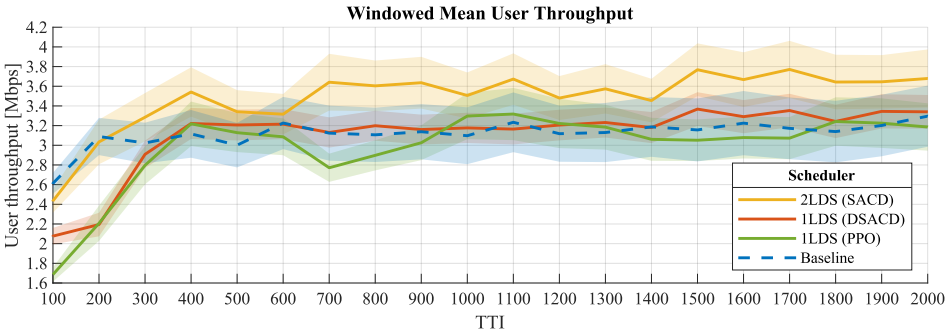

Figure 1: Convergence of windowed mean user throughputs during the training of best performing 1LDS and 2LDS options with 95% confidence interval. Due to higher entropy required by training 2LDS than 1LDS lower throughputs can be observed during the training.

3.3 Assessment of Numerical Performance

The final deep scheduler evaluations assessed the models under conditions significantly different from the training setup. Specifically, the evaluations used the full 100 MHz bandwidth, twice the maximum number of MU-MIMO user layers ($|L| = 8$ vs $|L| = 4$), and homogeneous traffic (100% FB or 100% FTP3), contrasting sharply with the resource-constrained (6.6 MHz), lower order MU-MIMO, mixed traffic training environment (see Table 4 in Appendix H for full details). Therefore, the strong performance and gains over baseline schedulers observed in these evaluations (detailed below, see Figures 2-3, Table 1) serve as direct quantitative evidence substantiating the claimed generalization capabilities across these different operating conditions. The results are collected from 10 different random simulation realizations. To ensure comparable geometric mean values, zero-valued UE throughputs are replaced with ones when calculating geometric means, as some random UE drops result in a few UEs being unable to receive data.

Figure 2 shows that all deep scheduler options perform well with full buffer traffic. Notably, the 2LDS configuration provides better throughput overall. In general, deep schedulers efficiently utilize spatial degrees of freedom by selecting UEs for each RBG and MU-MIMO user layer that can be co-scheduled with others while maintaining fairness. Figure 2 also highlights a key advantage over heuristics: that deep schedulers achieve higher MU-MIMO co-scheduling efficiency. This demonstrates the RL agents' ability to better navigate the complex combinatorial user pairing problem compared to the greedy, iterative nature of the baseline heuristics. In contrast the 1LDS (PPO) approach may sometimes over-schedule a few UEs for a single RBG, leading to minor degradation in user throughputs compared to SAC-based options.

Given the discontinuous nature of FTP traffic, we use User Perceived Throughput (UPT) to evaluate network performance with bursty traffic models. UPT, defined as the throughput measured only when a UE has downlink data to receive, is illustrated in Figure 3. The figure shows clear improvements in the fairness of the UPT distribution achieved by deep schedulers. A similar conclusion can be drawn from the classic user throughput CDF shown in Figure 3. The results reveal a particular strength under bursty traffic, where Table 1 and Figure 3 show dramatic improvements for cell-edge users (5th percentile) and fairness (geometric mean) compared to PF-based heuristics. This suggests that the learned policies overcome limitations in heuristic approaches in balancing efficiency and fairness for disadvantaged users under dynamic load. Table 1 summarizes the numerical throughput gains for separately simulated full buffer and bursty FTP model 3 traffic. All proposed deep schedulers achieve substantial gains, particularly for users in the worst radio conditions. Notably, the least complex variant, 1LDS DSACD, performs almost as well as the more complex 2LDS variant.

Our computational complexity measurements (Appendix F, Table 3) indicate that 1LDS meets strict per-TTI latency budgets, whereas 2LDS can struggle due to the $N_{RBG} \times L$ forward-pass requirement.

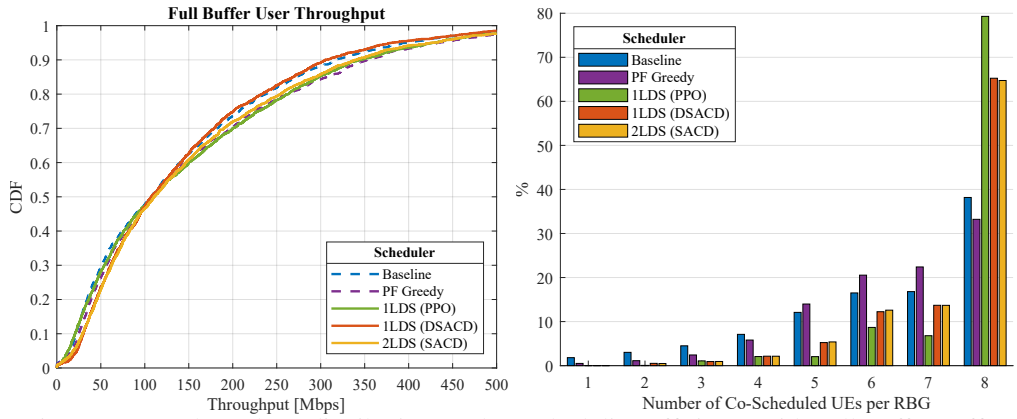

Figure 2: User throughput distributions and co-scheduling efficiency with Full Buffer traffic.

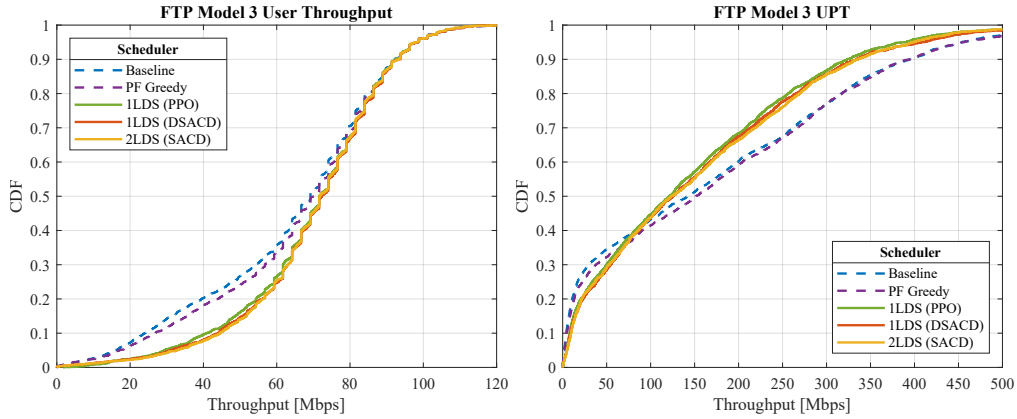

Figure 3: User throughput and UPT distributions with FTP Model 3 traffic. Most improvements are obtained among users in worst radio conditions.

Table 1: Summary of worst 5th %-ile, median, and geometric mean user throughput gains over the baseline with full buffer and FTP Model 3 traffic.

Full Buffer	**5th %-ile**	**Median**	**Geomean**
PF Greedy	11.1 %	1.3 %	9.5 %
1LDS (PPO)	-17.6 %	2.2 %	6.4 %
1LDS (DSACD)	46.7 %	0.2 %	13.7 %
2LDS (SACD)	20.6 %	3.1 %	15.6 %
FTP Model 3	**5th %-ile**	**Median**	**Geomean**
PF Greedy	7.9 %	1.4 %	3.5 %
1LDS (PPO)	85.5 %	3.5 %	18.5 %
1LDS (DSACD)	97.5 %	4.5 %	18.3 %
2LDS (SACD)	106.3 %	3.8 %	18.6 %

4 Conclusion

We presented practical RL-based deep schedulers for 5G NR, introducing architectural and algorithmic innovations that enable frequency- and spatial-domain scheduling under strict latency requirements. We did not rely solely on SoTA RL; instead, we introduced specific modifications to enhance its suitability for deep scheduler training. For PPO, incorporating expert guidance via JSD loss represents a key enhancement over standard implementations. Furthermore, we derived DSACD, a novel SAC variant, combining advances from state-of-the-art algorithms by integrating distributional critics for richer value representation and adaptive, state-aware entropy learning critical for stability with

dynamic action spaces, along with tailored prioritized replay. By employing a multi-branch output layer to schedule each MU-MIMO layer separately, DSACD provided the best trade-off between PF throughput and computational complexity for frequency-selective massive MIMO scheduling.

Our analysis also revealed that while deep schedulers based on 2LDS architectures may struggle to meet 5G TTI time limits, 1LDS-based designs offer a good balance between throughput performance and computational efficiency. This makes the 1LDS architectures a viable option for real-time applications, ensuring scalability to massive MIMO without compromising performance.

These findings underscore the potential of modified RL algorithms in efficiently scheduling massive radio resources, paving the way for more sophisticated and resource-laden next-generation cellular networks.

Acknowledgments and Disclosure of Funding

The authors would like to thank their colleagues Ian Garcia, Hua Xu, Igor Filipovich, Simon Teferi, Morgan Cormier, Ngoc Duy Nguyen, Simon Goedecke, Andre Gille, Marian Beniac, Simo Aaltonen, Petri Ervasti, Silvio Mandelli, and Pavan K. Srinath for their feedback during numerous exchanges. The European Union funds the work of B. Liu and A. Valcarce through the CENTRIC project (G.A no. 101096379).

References

[1] 3GPP. Technical report 36.889: Study on licensed-assisted access to unlicensed spectrum, . URL `https://www.3gpp.org/DynaReport/36889.htm`.

[2] 3GPP. Technical specification 38.101: User equipment (UE) radio transmission and reception, . URL `https://www.3gpp.org/DynaReport/38101.htm`.

[3] 3GPP. Technical report 38.913: Study on scenarios and requirements for next generation access technologies, . URL `https://www.3gpp.org/DynaReport/38913.htm`.

[4] Faroq Al-Tam, Noélia Correia, and Jonathan Rodriguez. Learn to Schedule (LEASCH): A Deep Reinforcement Learning Approach for Radio Resource Scheduling in the 5G MAC Layer. *IEEE Access*, 8: 108088–108101, 2020.

[5] Madyan Alsenwi, Nguyen H. Tran, Mehdi Bennis, Shashi Raj Pandey, Anupam Kumar Bairagi, and Choong Seon Hong. Intelligent Resource Slicing for eMBB and URLLC Coexistence in 5G and Beyond: A Deep Reinforcement Learning Based Approach. *IEEE Transactions on Wireless Communications*, 20(7): 4585–4600, 2021.

[6] Qing An, Santiago Segarra, Chris Dick, Ashutosh Sabharwal, and Rahman Doost-Mohammady. A deep reinforcement learning-based resource scheduler for massive mimo networks. *IEEE Transactions on Machine Learning in Communications and Networking*, 1:242–257, 2023.

[7] Nikolaos Apostolakis, Marco Gramaglia, Livia Elena Chatzieleftheriou, Tejas Subramanya, Albert Banchs, and Henning Sanneck. ATHENA: Machine Learning and Reasoning for Radio Resources Scheduling in vRAN Systems. *IEEE Journal on Selected Areas in Communications*, 42(2):263–279, 2024.

[8] Marc G. Bellemare, Will Dabney, and Mark Rowland. *Distributional Reinforcement Learning*. MIT Press, 2023.

[9] Johan Samir Obando Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *Proceedings of the 38th International Conference on Machine Learning*, pages 1373–1383, 18–24 Jul 2021.

[10] Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv: 1910.07207*, 2019.

[11] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, page 2892–2901, 2018.

[12] Jiancun Fan, Geoffrey Ye Li, and Xiaolong Zhu. Multiuser MIMO scheduling for LTE-A downlink cellular networks. In *IEEE 79th Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2014.

[13] A. Giovanidis, M. Leconte, Sabrine Aroua, Tor Kvernvik, and D. Sandberg. Online frequency scheduling by learning parallel actions. *arXiv: 2406.05041*, 2024.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018.

[15] Ramin Hashemi, Vismika Ranasinghe, Teemu Veijalainen, Petteri Kela, and Risto Wichman. User throughput optimization via deep reinforcement learning for beam switching in mmwave radio access networks. In *2024 IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops), to be published*, 2024.

[16] Petteri Kela, Jani Puttonen, Niko Kolehmainen, Tapani Ristaniemi, Tero Henttonen, and Martti Moisio. Dynamic packet scheduling performance in UTRA long term evolution downlink. In *International Symposium on Wireless Pervasive Computing*, pages 308–313, 2008.

[17] D. Kingma and J Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations (ICLR)*, 2015.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv: 1312.5602*, 2013.

[19] Moustafa Roshdi, Ethan Swistak, Reinhard German, and Mehdi Harounabadi. QoS-DRAMA: Quality of Service aware DRL-based Adaptive Mid-level resource Allocation scheme. In *The 2nd Workshop on Next-generation Open and Programmable Radio Access Networks (NG-OPERA) at IEEE INFOCOM*, 2024.

[20] David Sandberg, Tor Kvernvik, and Francesco Davide Calabrese. Learning robust scheduling with search and attention. In *IEEE International Conference on Communications*, pages 1549–1555, 2022.

[21] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference for Learning Representations (ICLR)*, 2016.

[22] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference for Learning Representations (ICLR)*, 2016.

[23] Michael Seguin, Anjali Omer, Mohammad Koosha, Filippo Malandra, and Nicholas Mastronarde. Deep Reinforcement Learning for Downlink Scheduling in 5G and Beyond Networks: A Review. In *IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6, 2023.

[24] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, José Luis Palmero Simón, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-start reinforcement learning. In *International Conference on Machine Learning*, 2022.

A Recent Deep Scheduler Efforts

Table 2: Deep Scheduler recent history

Ref.	UL	DL	ML focus	Opt. target	Actor model				
					Algorithm	Input	Hidden	Output	Act. fun.
2020 [4]		✓	FDS	bitrate fairness	DDQN	$2 \cdot \lvert U \rvert$	2×128	$\lvert U \rvert$	ReLU
2021 [5]		✓	QoS slicing	reliability	AC, PG	$2 \cdot \lvert U \rvert + 1$	N.K.	N_{res}	N.K.
2023 [6]	✓		FDS, SDS	bitrate fairness	SAC, KNN	$3 \cdot \lvert U \rvert$	2×64	1	ReLU
2024 [19]		✓	FDS	packet loss rate	PPO	$3 \cdot 6 = 18$	5×64	5	N.K.
2024 [7]	✓		MCS, N_{RBs}	bitrate	DDPG	$2 \cdot \lvert U \rvert$	128, 3×256, 128	2	ReLU, $\sigma(x)$
2024 [13]		✓	FDS, SDS	bitrate fairness	DQN, AB	$\lvert U \rvert \cdot N_{\text{RBG}} \cdot D \cdot N_{tx}$	~3.6M params	$N_{\text{RBG}} \cdot N_a$	ReLU

B Framework for RL-based Radio Resource Schedulers

To develop a practical deep radio resource scheduler that meets real-time BTS constraints while maintaining or exceeding legacy scheduler KPIs, we devised two ML-based scheduler architectures. The first, a 1LDS, optimizes execution time by providing scheduling decisions for all RBGs per MU-MIMO user layer in a single forward pass, as illustrated in Fig. 4. In contrast, the second 2LDS approach decides the user allocation of one single RBG per inference pass, requiring loops over both RBGs and MU-MIMO user layers. In both schemes, looping over the user layers is necessary.

Despite using larger input and final neural layers, the 1LDS design achieves a significant speedup due to fewer forward passes while managing to still keep ML model dimensions rather small. Concretely, 1LDS reduces the number of forward passes from $N_{\text{RBG}} \times |L|$ down to $|L|$, where $|L|$ denotes the maximum number of spatially co-scheduled users per RBG, and N_{RBG} is the total number of downlink RBGs to be allocated. This reduction is key in cellular systems with large bandwidths. However, performing a single forward pass for all RBGs and MU-MIMO layers would require a model that is too large for practical massive MIMO implementation.

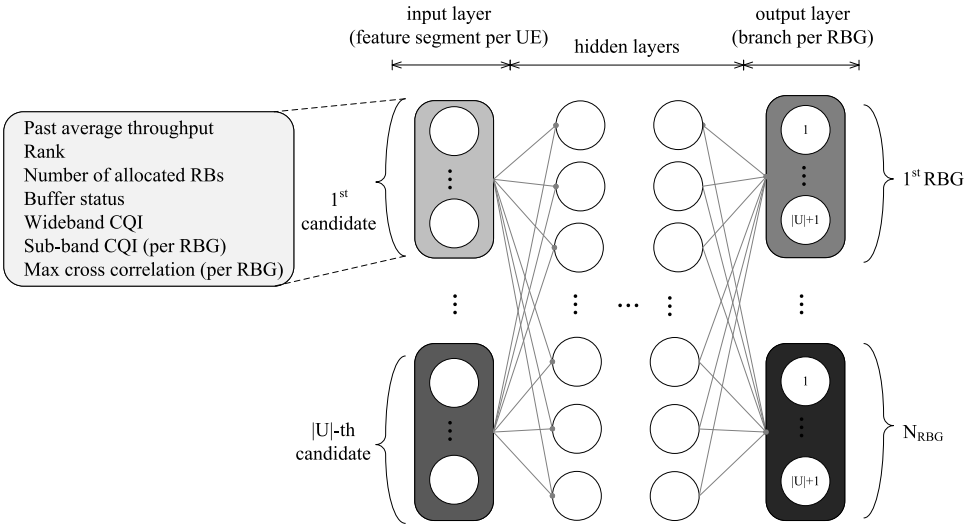

Figure 4: 1LDS baseline neural network architecture, with feature segment blanking for variable UE support. $|U|$ indicates maximum schedulable UEs per MU-MIMO user layer.

B.1 Action Space

For each decision stage, the deep scheduler selects a user from the set of candidate UEs provided by the Time Domain (TD) scheduler or chooses not to allocate any user. The action space is thus

defined as $\mathcal{A} = \{1, \ldots, |U|+1\}$, where $|U|$ is the number of candidate UEs, and the additional action $|U| + 1$ represents no allocation. The implementation of a decision stage, however, differs between the 1LDS and 2LDS architectures. In the 1LDS approach, a single decision stage corresponds to a complete user-layer. The scheduler performs one forward pass of the neural network to determine the action (user allocation or no allocation) for all RBGs within that specific user-layer simultaneously. Conversely, in the 2LDS architecture, a decision stage is implicitly equivalent to the allocation of an RBG within a user-layer. The scheduler iterates through each RBG and each user-layer, making an independent allocation decision at each step. Action masking is employed in both architectures to prevent invalid allocations, such as assigning already scheduled UEs to the same RBG or considering empty candidate inputs. In this framework, the deep scheduler makes decisions at each TTI, assigning a user (or no user) to each RBG per MU-MIMO layer. This process implicitly defines a time-frequency resource allocation pattern over consecutive TTIs.

B.2 State Space

The deep scheduler inputs are grouped into so-called state vectors, whose features have been chosen for their sufficiency in scheduling decision-making and low computational demands. Moreover, only values that the 5G standard supports, are already available in BTS, and do not require additional signaling overhead are used as input features. Since features are collected per UE, they effectively construct what we call a *UE feature segment*, and the state vector is therefore built out of $|U|$ UE feature segments. Each segment includes normalized values of UE's past average throughput, rank, number of allocated RBs, wideband CQI, sub-band CQI (per RBG) and max cross correlation (per RBG).

Next we describe the input features in detail, although for clarity, and omit the TTI index as follows: A prime denotes the variable at the next state, e.g., x', and a double prime denotes the variable at the previous state, e.g., x''.

- **Normalized Past Averaged Throughput ($\hat{R}_u$):** The past averaged throughput for user u is calculated using exponential smoothing:

  $$R_u = (1 - \epsilon)p_u + \epsilon R''_u,$$

  where p_u is the instantaneous throughput, R''_u is the previous averaged throughput, and ϵ is the forgetting factor. The normalized past averaged throughput is then:

  $$\hat{R}_u = \frac{R_u}{R_{\text{max}}},$$

  with R_{max} being the maximum observed past averaged throughput.

- **Normalized Rank of UE ($\hat{h}_u$):** The rank of user u, denoted by h_u, represents the UE's recommendation for the number of transmission layers. We normalize it as:

  $$\hat{h}_u = \frac{h_u}{2},$$

  The 2 in the denominator stems from the maximum UE rank (see Table 4).

- **Normalized Number of Already Allocated RBGs ($\hat{d}_u$):** d_u denotes the number of RBGs allocated to user u during the previous iteration of the scheduling loop across the MU-MIMO layers. This is then normalized as:

  $$\hat{d}_u = \frac{d_u}{d_{\text{max}}},$$

  where d_{max} is the bandwidth-dependent total number of RBGs across the full bandwidth.

- **Normalized Downlink Buffer Status ($\hat{b}_u$):** The downlink buffer status for user u, denoted by b_u, is normalized as:

  $$\hat{b}_u = \frac{b_u}{b_{\text{max}}},$$

  where b_{max} is a predefined maximum possible buffer size.

- **Normalized Wideband CQI ($\hat{o}_u$):** Min-max scaling is used to scale the wideband CQI for user u to the $[0, 1]$ range as follows:

$$\hat{o}_u = \frac{o_u - o_{\text{min}}}{o_{\text{max}} - o_{\text{min}}},$$

where o_u is the current wideband CQI, and o_{min} and o_{max} are the minimum and maximum possible CQI values, respectively.

Per-RBG User-Specific Input Features:

- **Normalized Sub-band CQI ($\hat{g}_{m,u}$):** The sub-band CQI for user u on RBG m, denoted by $g_{m,u}$, is normalized as:

$$\hat{g}_{m,u} = \frac{g_{m,u}}{\bar{g}_{m,u}},$$

where $\bar{g}_{m,u}$ is a predefined normalization scalar.

- **Max Cross-Correlation in User Pairing ($\rho_{m,u}$):** The maximum user pair cross-correlation between already scheduled users and the new candidate u on that RBG m is computed. This pre-calculated value is based on the precoder matrices $P_{m,u}$ and $P_{m,c}$ (for users u and c co-scheduled on RBG m) and is given by:

$$\rho_{m,u} = \max\{\kappa_{m,u,1}, \kappa_{m,u,2}, \ldots, \kappa_{m,u,N_{m,l}}\},$$

where $\kappa_{m,u,c} = \max\{\sum_i |[P_{m,u}^H P_{m,c}]_{i,j}|$, for $j \in \{1, 2, \ldots, h_c\}\}$, h_c is the rank of user c, $N_{m,l}$ is the number of co-scheduled users on RBG m at layer l, and $[\cdot]_{i,j}$ denotes the element at the i-th row and j-th column of a given matrix.

All inputs come from standard CSI and buffer reports. No extra uplink signaling is introduced. Having defined the input features, we can now specify the state vectors for the different architectural variants. For the fast 1LDS architecture (Fig. 4), the state vector includes five features ($\hat{R}_u, \hat{h}_u, \hat{d}_u, \hat{b}_u, \hat{o}_u$) and two RBG-specific features ($\hat{g}_{m,u}, \rho_{m,u}$) for each candidate UE. The size of this state vector is $|U| \times (5 + 2N_{\text{RBG}})$. On the other hand, the 2LDS scheduler, designed for throughput performance, executes one forward pass per RBG and receives RBG-specific inputs. These inputs include the seven previously defined features plus an eighth feature for the mean precoder cross-correlation, as well as the number of co-scheduled users so far. The input vector size for the 2LDS scheduler is thus $|U| \times 8 + 1$.

The defined state features enable adaptation to key network dynamics. Channel variations, including fading and multi-cell interference, are implicitly captured via the comprehensive CSI inputs (CQIs, rank, correlation). The local traffic load is reflected in the buffer status and past throughput features. Although the current design handles FD/Spatial Domain (SD) scheduling across different traffic models, explicitly incorporating fine-grained QoS parameters remains a future work. This is because traditionally QoS optimization is seen as a job of scheduling candidate selection, which is outside the scope of this study. Considering all these neural architectures and input feature vectors, there exist multiple ways to train them. In the following sections, we describe different training algorithms and their performance.

B.3 Optimization Goal

All of the deep schedulers shown here share one common objective: Maximizing the long-term geometric mean of user throughput. This is a commonly used KPI of radio schedulers, which captures both network capacity and fairness of resource allocation. It is defined as follows:

$$G = \left(\prod_{u=1}^{|U|} p_u \right)^{1/|U|}. \tag{1}$$

C On-Policy Deep Scheduler Training

Although pre-trained Deep Reinforcement Learning (DRL) schedulers may be sufficient in most cellular scenarios, niche applications like military deployments require rapid adaptation to unpredictable radio environments. In these safety-critical situations, the stability and cautious exploration of on-policy RL methods are advantageous.

C.1 On-policy Framework with Expert Guidance

On-policy RL algorithms like PPO are favored for their ease of implementation and stability. However, their reliance on the current policy for environment interaction can lead to slower convergence due to limited sample diversity. Addressing the pertinent challenge of enhancing sample efficiency and accelerating learning in complex RL tasks like scheduling, our proposed 1LDS-based framework integrates an *expert policy* to guide the PPO agent during training (similar to the guide policy described in [24]). This expert policy, a PF scheduler in our case, acts as a source of additional training data, and is illustrated in Fig. 5.

The training phase involves four key stages:

- **State and reward observation**: Following traditional reinforcement learning, the input state for the actor and critic networks is updated at each new TTI. Additionally, the reward from the previous TTI is collected and saved into an experience buffer for the PPO agent to update its policy.
- **Expert label generation**: Based on system requirements and computational resources, an expert policy generates labels to guide the PPO agent.
- **Policy update**: Once the experience buffer reaches its capacity, the PPO agent updates its policy using the collected data and expert labels. This is achieved through back-propagation and optimization of a pre-defined loss function.
- **Resource allocation**: The deep scheduler returns a vector of UE candidate indices for each RBG. After that, the candidates vector is sampled and the resource allocator assigns the RBG to the chosen UE. Modulation and coding schemes are selected after all RBGs have been allocated.

This approach leverages the strengths of both on-policy RL and expert knowledge, leading to faster convergence and improved scheduling performance in a practical setting. The other fundamental components of training a 1LDS-PPO are detailed next. The complete training procedure is formally outlined in Algorithm 1, Appendix G.

C.2 Fundamental Components

C.2.1 Action Space

The final layer of the 1LDS-PPO actor network outputs a vector $\mathbf{z} \in \mathbb{R}^{(|U|+1)N_{\text{RBG}}}$, which is then reshaped into $\hat{\mathbf{z}} \in \mathbb{R}^{N_{\text{RBG}} \times (|U|+1)}$. Let $\text{softmax}(X, d)$ be a softmax function to be applied to the d-th dimension of an input matrix X. The softmax function is applied along the second dimension (representing UE candidates) to obtain a probability distribution $\tilde{\mathbf{z}} \in \mathbb{R}^{N_{\text{RBG}} \times (|U|+1)}$ over the UEs for each RBG: $\tilde{\mathbf{z}} = \text{softmax}(\hat{\mathbf{z}}, 2)$, where $\sum_{u=1}^{|U|+1} \tilde{z}_{m,u} = 1$ for $m \in \{1, 2, ..., N_{\text{RBG}}\}$.

From this distribution, the action vector $\mathbf{a} = [a_1, a_2, ..., a_{N_{\text{RBG}}}]^{\text{T}}$ is sampled, with each element $a_m \in \{1, 2, ..., |U| + 1\}$ representing the index of the chosen UE for RBG m. If the selected index does not correspond to a valid user, the RBG remains unassigned.

In summary, the action determination process involves:

- Obtaining the output vector $\mathbf{z}$ from the actor network.
- Reshaping $\mathbf{z}$ into $\hat{\mathbf{z}}$.
- Applying the softmax function to obtain the probability distribution $\tilde{\mathbf{z}}$.
- Sampling the action vector $\mathbf{a}$ from $\tilde{\mathbf{z}}$.

C.2.2 Reward Function

The reward function is designed to balance two key objectives: maximizing the geometric mean of user throughput (for fairness) and maximizing the performance gain from MU-MIMO. This approach encourages the scheduler to learn a policy that efficiently allocates resources while exploiting the benefits of MU-MIMO. In a 1LDS architecture, the deep scheduler loops over the MU-MIMO user

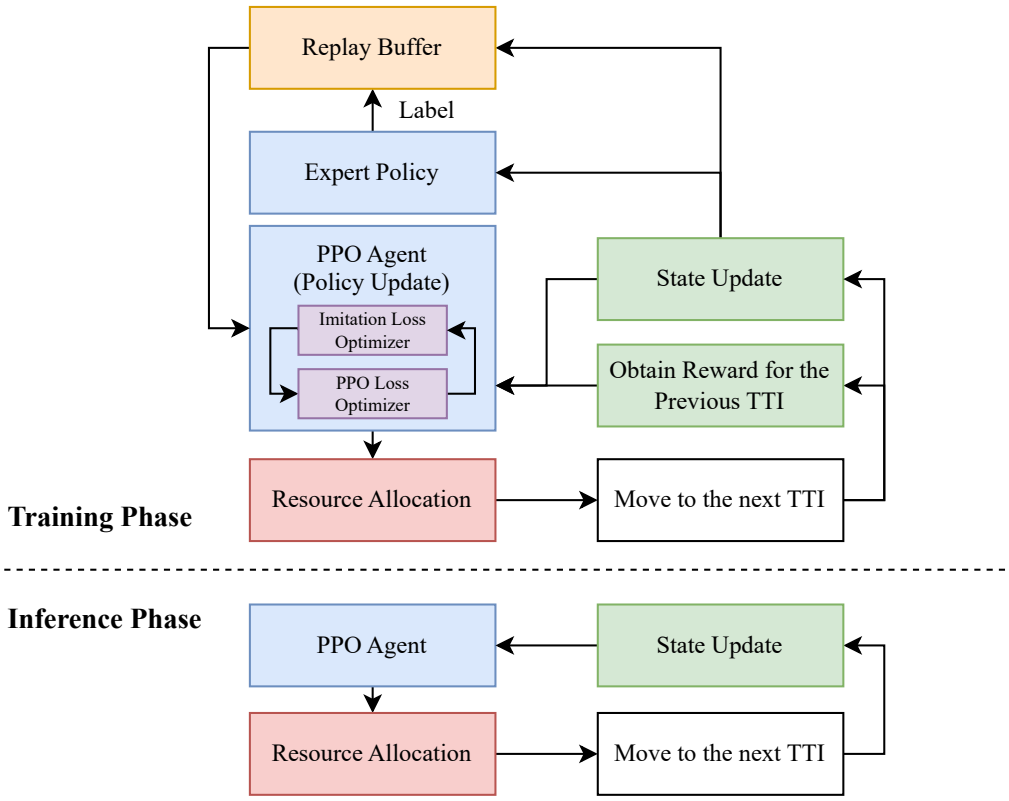

Figure 5: Overall Framework of 1LDS - PPO

layers, and the reward for the l-th user layer and m-th RBG, denoted by $r_{m,l}$, is defined as

$$r_{m,l} = \begin{cases} P \cdot v_m & \text{for } l = 1 \\ k \cdot v_m & \text{otherwise} \end{cases}, \tag{2}$$

where

$$P = \frac{1}{G_{\text{max}}} \cdot G \tag{3}$$

refers to the geometric mean of user throughput at the current TTI, G_{max} is a training-only normalizer, without dependencies at inference. We mitigate scale-shift by training under mixed traffic and test under stronger settings (100 MHz, $L = 8$). In addition,

$$v_m = \begin{cases} -1 & \text{if a better allocation choice exists} \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

denotes the result of a greedy search to determine whether another resource allocation exists, whose PF metric exceeds the one selected by the deep scheduler. k is a factor that adjusts the effect of greedy search during user pairing.

We consider the greedy search indicator v_m to determine whether an alternative UE allocation could enhance the total PF metrics. This indicator captures the trade-off between overall throughput and individual user fairness. By incorporating throughput fairness and PF metrics during user pairing into the reward function, the scheduler is guided to learn a policy that efficiently allocates resources while leveraging the spatial multiplexing capabilities of MU-MIMO. Following conventional reinforcement learning training architecture, the reward is computed after each iteration of the user layer.

Furthermore, consistent with standard RL timing (see Fig. 5), the reward $r_{m,l}$ corresponding to actions taken during TTI t is computed based on the instantaneous throughput p_u realized in that

interval (which determines G used in P), but this reward value becomes available for agent updates at the start of TTI $t+1$. This inherent delay ensures the required throughput outcome from TTI t is known before the corresponding reward is computed, resolving any apparent conflict with the timing of final MCS selection which occurs after allocations for TTI t are complete.

C.2.3 Training Data Augmentation

To enhance generalization and accelerate training, we shuffle the UE feature segments within the input state (see Fig. 4). This generates additional experience samples for the scheduler model to learn from while preserving the reward and swapping the UE candidate indices according to the permutation order. The number of permutations, N_{Π}, is a pre-set hyperparameter. Each new experience tuple saved in the buffer undergoes N_{Π} permutations, effectively multiplying the available training samples.

An experience replay buffer $\mathcal{D}_{\text{expert}}$ with ample memory is also employed to store input state-action pairs from the expert policy. This buffer provides the learning agent with diverse expert demonstrations to learn from, further improving its performance. Unlike traditional off-policy replay buffers, which store rewards, next states, and termination flags, this buffer focuses solely on current states and corresponding expert actions.

C.2.4 Loss Function with Expert Guidance

To accelerate convergence and benefit from expert knowledge, we introduce an additional loss term to the standard PPO loss function, encouraging the on-policy DRL agent to mimic the expert policy. The expert policy generates the labels for the scheduling decisions that the PPO agent can learn from. This term is based on the Jensen-Shannon Divergence (JSD), which measures the similarity between probability distributions. JSD is symmetric, robust to zero probabilities (possible in radio scheduling), and bounded between 0 and 1, offering a normalized dissimilarity metric.

Specifically, we calculate the JSD loss, $\mathcal{L}_{\text{JSD}}$, between the actor network's output and the expert policy's label:

$$\begin{aligned}\mathcal{L}_{\text{JSD}} = \frac{1}{2} \sum_{a_u}^{\{1,2,\ldots,|U|+1\}} \pi_{\theta_{\text{PPO}}}(a_u|s) \log \frac{\pi_{\theta_{\text{PPO}}}(a_u|s)}{\pi_{\text{expert}}(a_u|s)} \\ + \frac{1}{2} \sum_{a_u}^{\{1,2,\ldots,|U|+1\}} \pi_{\text{expert}}(a_u|s) \log \frac{\pi_{\text{expert}}(a_u|s)}{\pi_{\theta_{\text{PPO}}}(a_u|s)}\end{aligned} \tag{5}$$

where:

- $\pi_{\theta_{\text{PPO}}}(a_u|s)$ is the probability of the PPO agent taking action a_u given state s.
- $\pi_{\theta_{\text{expert}}}(a_u|s)$ is the probability of an expert policy taking action a_u given state s.

To help the agent explore and converge to a policy better than the expert, the traditional PPO loss is employed and defined as:

$$\mathcal{L}_{\text{PPO}} = L_{\text{value}} - L_{\text{policy}} - \xi L_{\text{entropy}}, \tag{6}$$

$$L_{\text{value}} = \frac{1}{2}\mathbb{E}\big\{(V(s) - \hat{J})^2\big\}, \tag{7}$$

$$L_{\text{policy}} = \mathbb{E}\Big\{\min\big(\varrho(\theta)\hat{A}, \text{clip}(\varrho(\theta), 1-\epsilon, 1+\epsilon)\hat{A}\big)\Big\}, \tag{8}$$

$$L_{\text{entropy}} = H(\mathbf{z}|s; \pi_{\theta_{\text{PPO}}}) \tag{9}$$

where:

- ξ is a coefficient controlling the strength of entropy regularization, promoting exploration.
- $H(\cdot)$ is the statistical entropy of the current policy.
- $\hat{A}$ is the advantage value, estimated using a Generalized Advantage Estimator (GAE) [22].
- $\varrho(\cdot) = \frac{\pi_{\theta_{\text{PPO}}}}{\pi_{\theta'_{\text{PPO}}}}$ is the ratio of probabilities between the updated policy ($\pi_{\theta_{\text{PPO}}}$) and the old policy ($\pi_{\theta'_{\text{PPO}}}$). The overall policy is calculated as the product of individual action probabilities across all RBGs: $\pi_{\theta_{\text{PPO}}} = \prod_{i=1}^{N_{\text{RBG}}} \pi_{\theta_{\text{PPO}},i}$.

- ϵ is the clipping parameter to constrain policy updates and maintain stability.
- $V(\cdot)$ is the estimate of the state value function.
- $\hat{J}$ is the target return, computed as the cumulative sum of discounted rewards.

We train the agent using alternating optimization. In each training step, the standard PPO loss ($\mathcal{L}_{\text{PPO}}$) is first minimized using batches of standard agent experiences. Then, the $\mathcal{L}_{\text{JSD}}$ loss is minimized using expert state-action pairs sampled from a dedicated replay buffer, thereby incorporating expert guidance. This alternating process allows the agent to learn simultaneously from its own experience (via PPO) and from the expert's guidance (via JSD), aiming for faster convergence and potentially better performance than using PPO alone. Note also that two buffers are used: a small, temporary one for collecting the PPO update batch (size $M = 128$ as per Table 5) and a larger, persistent one for storing expert demonstrations for the JSD loss (size 4000 as per Table 5, Appendix H).

D Off-Policy Deep Scheduler Training

Off-policy RL methods offer a distinct advantage over their on-policy counterparts by allowing agents to learn from experiences generated by a different policy. This enables efficient utilization of past scheduling decisions stored in a replay buffer, facilitating faster and more robust learning. In radio resource scheduling, off-policy methods are particularly appealing when exploration and stability are critical.

This section investigates and combines several off-policy RL advancements suitable for our radio resource allocation problem. We first consider SAC [14], which is proven to be a rather effective and sample efficient algorithm due to its entropy regularization. However, as the original SAC algorithm is designed for continuous action spaces, we adopt a modified version, SACD [10], instead. This variant is better suited for our discrete action space, with its critic networks estimating Q-values for all actions simultaneously, allowing the actor to learn from the entire Q-value distribution for a given state. Furthermore, since nearby scheduling decisions are not necessarily correlated (especially with shuffled state feature segments and a correspondingly branched output layer, see Figure 4), the Gaussian policy of the original SAC is less suitable than the individual action outputs of SACD, similar to traditional DQN-based algorithms.

Finally, we explore distributional RL, by extending SACD, with distributional critic networks, resulting in DSACD. With both SAC variants, we learn a policy π_θ and two Q-functions $Q_{\phi 1}$ and $Q_{\phi 2}$, along with their soft-updated targets $\bar{Q}_{\bar{\phi} 1}$ and $\bar{Q}_{\bar{\phi} 2}$.

D.1 Distributional Soft Actor-Critic Discrete

A key problem in distributional RL [8] is that it increases the ML model size and computational complexity. This is not desired for models intended for real-time systems such as BTSs. To capture the benefits of distributional learning and SoTA maximum entropy RL for discrete action spaces, we modify the online and target critic networks to be distributional while keeping the actor (policy) network unchanged. This allows us to train a model that combines maximum entropy RL with distributional RL and yield an actor model (policy network) that is no more complex to execute than the original DQN [18].

To transform the critic networks into distributional ones, we apply two key modifications proposed in [11] to the SACD critic networks. First, we expand the critics' output layers by a factor of N, a hyperparameter that determines the number of quantiles used to approximate the distribution of Q-values. This results in an output layer size of $|A| \times N$, where $|A|$ is the size of the action space. This expansion allows the network to represent a distribution of Q-values for each action, providing a richer representation of the value function. Second, we replace the Mean Square Error (MSE) loss used in the original SACD critics with the quantile Huber loss. This loss is more robust to outliers than MSE, and is given by:

$$L(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| < k \\ k(|x| - \frac{1}{2}k) & \text{otherwise} \end{cases}, \tag{10}$$

where k is a hyperparameter we set to $k = 1$.

The quantile Huber loss, an asymmetric variant of the Huber loss is given by:

$$\rho_\tau^k(x) = |\tau - \delta_{\{x<0\}}|L(x), \tag{11}$$

where δ denotes a Dirac delta function and τ represents the quantile level, with N quantiles evenly spaced between 0 and 1 as $\tau_n = \frac{n}{N}$, for $n = 1, ..., N$. This loss function creates N quantiles per action in the output layers of the critic networks, allowing them to represent a distribution of Q-values instead of a single point estimate. This distributional representation captures uncertainty in the value estimation and can lead to improved performance in RL tasks.

For our DSACD algorithm, the error x_t for each quantile $n = 1, ...N$ is defined as:

$$x_{\phi,n} = q_{\phi,n}(s, a) - y_{\bar{\phi}}(r(s, a), s'), \tag{12}$$

where $q_{\phi,n}(s, a)$ is the n-th quantile output of the critic network for state s and action a, and the target value is calculated as:

$$\begin{aligned} y_{\bar{\phi}}(r(s, a), s') = r(s, a) + \\ \gamma\pi_\theta(s', a')\big(\bar{Q}_{\bar{\phi}}(s', a') - \alpha \log(\pi_\theta(s', a'))\big), \\ a' \sim \pi_\theta(\cdot|s'). \end{aligned} \tag{13}$$

Here, γ is the discount factor, $r(s, a)$ is the reward, α is the learned entropy regularization coefficient, and $\pi_\theta(s', a')$ is the probability of taking action a' in state s' according to the actor network. The term $\bar{Q}_{\bar{\phi}}(s', a')$ represents the expected value from the target critic network for state s' and action a'. Note that action a' for the next state s' is drawn from the probability distribution given by the actor-network. The target value $\bar{Q}_{\bar{\phi}}(s', a')$ is obtained by averaging over all N quantiles in the next state s' for the drawn action a', as follows:

$$Q_\phi(s, a) := \sum_{n=1}^{N} \frac{1}{N} q_{\phi,n}(s, a). \tag{14}$$

This distributional approach captures the uncertainty in future rewards and allows for a more robust and nuanced learning process.

The actor-network is trained following the principles of SACD [10]. During training, a softmax function is applied to the output layer (separately for each output branch for 1LDS approach) to convert the discrete outputs into a probability distribution over actions. However, to simplify inference and reduce complexity, the softmax operation can be omitted after training, allowing greedy action selection using argmax. The main difference from the original SACD [10] lies in handling the distributional Q-values from the critic networks. Since the critics output a distribution of Q-values for each action, we compute the mean Q-value across all N quantiles for each action. This transforms the quantile outputs into a format comparable to the actor network's probability distribution. To mitigate Q-value overestimation, SAC utilizes two critic networks. The policy objective function, which guides the actor's learning, is defined as:

$$J_\pi(\theta) = \mathbb{E}_{s\sim D}\big[\pi_\theta(s) \odot [\alpha \log(\pi_\theta(s)) - \min_{j=1,2} Q_{\phi j}(s)]\big], \tag{15}$$

where D is the replay memory and the array of mean Q-values $Q_{\phi j}(s)$ from critic network j are obtained as described previously. It should be noted that the min operation selects the smaller Q-value from the two critics for each action, promoting conservative estimates. This objective function guides the gradient descent process across all actions in the discrete action space. While the Q-networks are updated based on the actions taken, the policy is trained using all action probabilities and their corresponding Q-values, ensuring comprehensive learning from the state-action value distribution.

D.2 Learning SACD Entropy with Action Masking

To ensure the selection of valid scheduling candidates, we employ action masking to exclude candidates already scheduled for specific RBGs. Additionally, the number of valid actions may vary across TTIs due to the dynamic nature of TD scheduling.

In SACD, the entropy regularization parameter α is learned, and the entropy target depends on the size of the action space. Instead of using a static target value $\bar{H}$ as in [10], we adopt a state-specific target in the entropy objective:

$$J(\alpha) = \mathbb{E}_{s\sim D}[\pi_\theta(s)[\alpha(\log(\pi_\theta(s)) + \bar{H}(s))]], \tag{16}$$

where the state-specific target is derived from the number of valid actions $|A(s)|$ in the current state:

$$\bar{H}(s) = -\beta \cdot \log\left(\frac{1}{|A(s)|}\right). \tag{17}$$

Here, β is a hyperparameter controlling the target entropy. While the default value of $\beta = 0.98$ from [10] performs well, we further optimized our deep scheduler by tuning α based on the number of valid actions and searching for a more optimal β (see values in Table 6, Appendix H).

Masks are stored in the replay memory alongside states, enabling us to determine the number of valid actions during training. Since the actor produces probabilities for all actions, including invalid ones, the probability distribution is adjusted by setting invalid action probabilities to zero and renormalizing the remaining probabilities to sum to one. Finally, the entropy is updated using gradient descent, averaging over all actions:

$$\alpha \leftarrow \alpha - \frac{1}{|A(s)|}\sum_{a}^{A} \pi_\theta(s,a)[\log(\pi_\theta(s,a)) - \bar{H}(s)], \tag{18}$$

where the logarithm is applied element-wise to the probability distribution of the discrete action space.

D.3 Prioritized Experience Replay

Replay memory prioritization, based on the method proposed in [21], originally generated bias toward transitions with higher temporal difference (TD) errors in DQNs. However, in our DSACD algorithm, we have two Q-networks. Therefore, we adapt the prioritization strategy [21]. The priority δ_i assigned to a transition i is based on the average absolute TD error across the critic networks and quantiles, ensuring transitions with larger estimation errors are prioritized:

$$\delta_i = \frac{1}{2N}\sum_{j=1}^{2}\sum_{n=1}^{N} |x_{\phi j,n,i}| + \epsilon, \tag{19}$$

where $x_{\phi j,n}$ is the TD error for quantile n of critic j for transition i (computed as in Equation (12)), and ϵ is a small positive constant ensuring non-zero priority. Transitions are then sampled from the replay buffer $\mathcal{D}$ with probability $P(i)$ calculated as:

$$P(i) = \frac{\delta_i^\omega}{\sum_{k\in\mathcal{D}} \delta_k^\omega},$$

where ω is a hyperparameter (see Table 5 in Appendix H) controlling the degree of prioritization ($\omega = 0$ yields uniform sampling). To correct for the bias introduced by this non-uniform sampling, importance sampling weights $w_i = 1/(|\mathcal{D}| \cdot P(i)))^{\omega'}$ are applied during the loss calculations (where ω' typically anneals from an initial value to 1). It should be noted that we use the same method for SACD.

D.4 Rewarding Off-Policy Methods

Our training objective is to develop a deep scheduler that allocates scheduling candidates in both the frequency (RBG) and spatial (MU-MIMO) domains in a proportionally fair manner. One approach to achieve this is to maximize the long-term geometric mean of user throughput. However, due to the necessity of making multiple RBG decisions per MU-MIMO user layer at each TTI, we found it more effective to maximize the instantaneous increase in the PF metric resulting from each scheduling decision. Therefore, the reward for the u-th scheduling candidate on the m-th RBG and l-th MU-MIMO user layer is defined as:

$$r_{u,m,l} = \begin{cases} \dfrac{T_{u,m,l}}{R_u} - \dfrac{T_{u,m,l-1}}{R_u} & \text{for } l > 1 \\ \dfrac{T_{u,m,l}}{R_u} & \text{otherwise} \end{cases}, \tag{20}$$

where $T_{u,m,l}$ is the achievable data rate for that RBG and user layer, and R_u is the past average throughput of user u.

To prevent the model from learning to artificially minimize past average throughput to inflate rewards, we normalize the reward at each TTI:

$$r_{u,m,l} \leftarrow \begin{cases} \max\left\{\dfrac{r_{u,m,l}}{\max\limits_u r_{u,m,l}}, -1\right\} & \text{for } \max\limits_u r_{u,m,l} > 0 \\ 1 & \text{for } \max\limits_u r_{u,m,l} < 0 \text{ and } a = |A| \\ -1 & \text{otherwise} \end{cases}, \tag{21}$$

This normalization clips the maximum reward to 1 and the minimum to -1, ensuring a stable and balanced learning process. We set the discount factor $\gamma = 0$ intentionally to emphasize immediate PF increments per RBG decision. Long-term fairness is already enforced via the geomean KPI in evaluation. Finally, the detailed training update steps for the DSACD algorithm are presented in Algorithm 2, Appendix G.

E Reference Scheduler Architecture

Building on the established practice of splitting radio scheduling into TDS and FDS as described in [16], our reference scheduler architecture adds an SDS step to support MU-MIMO. The TDS step shortlists the candidate UEs for scheduling based on QoS requirements and, for example, the PF metric. The FDS step allocates frequency resources (optionally non-contiguous) to these candidates. With MU-MIMO, the SDS step pairs candidate UEs on additional DL MU-MIMO user layers for each RBG. We use two iterative heuristic SDS methods similar to those in [12], modifying them to first allocate Single-User MIMO (SU-MIMO) resources with FDS and then add spatial user layers until no suitable candidates remain or no additional MU-MIMO gain is expected with Regularized Zero Forcing (RZF)-based beamforming. We describe both SDS variants next:

E.1 Baseline Spatial Domain Scheduler

The baseline SDS algorithm operates on a pre-sorted list of UE candidates using the PF metric from the TDS phase. It iterates through candidates in PF-sorted order, selecting the first UE that increases the total throughput of the RBG when co-scheduled with UEs already assigned to previous MU-MIMO user layers. This approach can be computationally demanding for real-time environments, as it may need to iterate through all candidates to find one that improves sum throughput, posing a challenge for practical implementation.

E.2 Proportional Fair Exhaustive Search per MU-MIMO Layer Spatial Domain Scheduler

To explore further gains, we implemented a modified SDS scheduler, *PF Greedy* SDS. For each RBG and MU-MIMO layer, this scheduler exhaustively searches for the candidate UE that maximizes the PF sum metric and increases the MU-MIMO sum throughput estimate for each RBG and spatial layer. This approach increases computational complexity, requiring recalculating beamforming weights and throughput estimations for every scheduled UE and new candidate for each RBG and spatial layer. While an exhaustive search over all MU-MIMO user layers could theoretically find the optimal combination, it is computationally infeasible even within our system-level simulations, let alone in a real-time environment.

F Computational Complexity Analysis

Table 5 outlines the parameters affecting the computational complexity of both off- and on-policy methods. When comparing the 1LDS and 2LDS architectures, it is important to clarify the definition of *computational complexity* used herein. While the 2LDS variant employs a neural network with fewer parameters per forward pass, its overall computational complexity per TTI is significantly higher due to the operational requirement of performing a separate inference for each RBG and each MU-MIMO user layer. This results in $N_{RBG} \times |L|$ forward passes per TTI for 2LDS, compared to only $|L|$ passes for 1LDS, which processes all RBGs for a given layer simultaneously. Therefore, in

this context, *complexity* primarily refers to the total computational load and resulting inference latency per TTI, dominated by the number of required forward passes rather than solely the parameter count of the neural network used in a single pass. This distinction is critical for real-time implementation constraints, as elaborated below and quantified in Table 3.

All input parameters are designed to avoid unnecessary per-TTI calculations, as they can be pre-calculated based on CSI reports. Additionally, UE SU-MIMO precoder cross-correlations can be pre-calculated, with only the maximum value needed during the scheduling of additional MU-MIMO layers. Consequently, execution times are primarily dependent on neural network forward passes. Indicative forward pass times, measured using single-precision floating-point format neural networks on an Intel Xeon Gold 6330 CPU, are provided in Table 3. We cannot disclose results with real commercial BTS hardware, but these equivalent measurements obtained from simulations using a CPU-optimized in-house C++ implementation, suggest that deep schedulers with 2LDS architectures may struggle to meet 5G inference latency requirements and TTI limits. In contrast, the 1LDS design offers a good balance between throughput performance and computational complexity, with scalability to massive MIMO. We do not provide CPU times for the baselines because they depend on specific implementations and libraries, making direct comparisons unsuitable.

Table 3: CPU times of deep scheduler forward passes with 18 RBGs and 8 MU-MIMO UE layers by using ReLU activation.

Model architecture	**2LDS**	**1LDS**	
Neurons per Hidden Layer	**32**	**64**	**32**
CPU time per forward pass	2 us	5.6 us	3.4 us
Sum CPU time per TTI	288 us	44.8 us	27.2 us

The execution times shown do not include state feature vector filling or other simulator or product-specific procedures. However, due to the pre-calculated inputs, such preprocessing consumes a negligible amount of CPU time. Forward pass times can be further reduced when models are used with CPUs supporting native half-precision floating-point operations.

Unlike heuristic baseline schedulers that require extensive throughput estimations with updated RZF precoders and re-selected MCSs, deep schedulers calculate RZF precoders and select MCS only once, after obtaining scheduling decisions from the actor networks. This efficiency makes ML models particularly appealing for scheduling tasks.

G Pseudocode for Proposed RL Algorithms

For clarity and reproducibility, the following algorithms detail the implementation of the training procedures for the 1LDS-PPO (Algorithm 1) with expert guidance and DSACD (Algorithm 2) methods presented in Appendix C and Appendix D.

Algorithm 1 1LDS-PPO with Expert Guidance Training

Initialize actor network π_θ, critic network V_ϕ.
Initialize agent experience buffer $\mathcal{B}_{\text{agent}}$ (size $M = 128$, for on-policy PPO updates).
Initialize expert demonstration buffer $\mathcal{D}_{\text{expert}}$ (size 4000 from Table 5 for JSD loss).
Initialize expert policy π_{expert} (e.g., PF scheduler).
Hyperparameters: PPO clip ϵ, entropy coeff ξ, JSD weight λ_{JSD}, layers L.
for each environment TTI $t = 1, 2, ...$ **do**
    Initialize temporary storage for TTI trajectory $\mathcal{T}_{\text{TTI}}$.
    for layer $l = 1$ to L **do**
        Observe state $s_{t,l}$ (Section III-B).
        Shuffle candidate UEs within $s_{t,l}$ if needed. ▷ Perform Data Augmentation (optional, Appendix C.2.3)
        Generate action probabilities $\mathbf{z}_{t,l} = \pi_\theta(s_{t,l})$. Sample action $\mathbf{a}_{t,l}$ (Appendix C.2.1).
        Generate expert action $\mathbf{a}_{\text{expert},t,l} = \pi_{\text{expert}}(s_{t,l})$ (Appendix C).
        Store $(s_{t,l}, \mathbf{a}_{t,l}, \log\pi_\theta(\mathbf{a}_{t,l}|s_{t,l}), \mathbf{a}_{\text{expert},t,l})$ in $\mathcal{T}_{\text{TTI}}$. ▷ Store log prob for PPO
        Resource allocation based on $\mathbf{a}_{t,l}$ happens here
    end for
    ▷ Environment step complete for TTI t. MCS selected, throughput p_u realized.
    ▷ At TTI $t+1$, calculate rewards for TTI t:
    Calculate G using p_u from TTI t (Equation (1)). Calculate P (Equation (3)).
    for each $(s_{t,l}, \mathbf{a}_{t,l}, \log\pi_{\text{old}}, \mathbf{a}_{\text{expert},t,l})$ in $\mathcal{T}_{\text{TTI}}$ **do**
        Calculate reward $r_{t,l}$ using P and v_m (Equation (2)). ▷ v_m needs to be computed based on TTI t outcome
        Store $(s_{t,l}, \mathbf{a}_{t,l}, r_{t,l}, \log\pi_{\text{old}})$ in agent buffer $\mathcal{B}_{agent}$. ▷ Accumulates across TTIs
        Store $(s_{t,l}, \mathbf{a}_{\text{expert},t,l})$ in expert buffer $\mathcal{D}_{\text{expert}}$ (if buffer not full or using replacement).
    end for
    if $|\mathcal{B}_{\text{agent}}|$ reaches update size M **then** ▷ Perform Alternating Optimization Update
        ▷ PPO Update
        Compute target returns $\hat{J}_i$ and advantages $\hat{A}_i$ for transitions in $\mathcal{B}_{\text{agent}}$ using GAE [22].
        Compute PPO loss $\mathcal{L}_{\text{PPO}}$ (Equations (6) to (9)) using transitions in $\mathcal{B}_{agent}$.
        Update $\theta, \phi \leftarrow \theta, \phi - \lambda_{\text{PPO}}\nabla_{\theta,\phi}\mathcal{L}_{\text{PPO}}$.
        Clear agent buffer $\mathcal{B}_{\text{agent}}$.
        ▷ Expert Guidance Update
        Sample mini-batch $\mathcal{B}_{\text{expert}} = \{(s_j, \mathbf{a}_{\text{expert},j})\}_{j=1..b'}$ from $\mathcal{D}_{\text{expert}}$. ▷ b' might differ from PPO batch
        Compute JSD loss $\mathcal{L}_{\text{JSD}}$ (Equation (5)) using B_{expert} and current policy π_θ.
        Update policy $\theta \leftarrow \theta - \lambda_{\text{JSD}}\nabla_\theta\mathcal{L}_{\text{JSD}}$.
    end if
end for

Algorithm 2 DSACD Training Update

Initialize policy network π_θ, critic networks Q_{ϕ_1}, Q_{ϕ_2} (outputting N quantiles per action), target networks $\overline{Q}_{\overline{\phi}_1}, \overline{Q}_{\overline{\phi}_2}$ with $\overline{\phi}_j \leftarrow \phi_j$.
Initialize entropy coefficient α, replay buffer $\mathcal{D}$ (size from Table 5), quantile levels $\tau_n = n/N$.
Hyperparameters: Mini-batch size b, PER ω, target entropy β, learning rates $\lambda_Q, \lambda_\pi, \lambda_\alpha$, target update τ.
procedure UPDATENETWORKS
    if $|\mathcal{D}| < b$ **then** ▷ **Wait for enough samples**
        return
    end if
    Sample mini-batch $B = \{(s_i, a_i, r_i, s'_i, \text{mask}_i, \text{mask}'_i)\}_{i=1\ldots b}$ from $\mathcal{D}$ using sampling probabilities $P(i)$
▷ **Critic Update**
    Compute target quantile values $y_{\overline{\phi},n}(r_i, s'_i)$ for each $n = 1..N$ using Eq. 13 *(Requires target critics, policy π_θ, α)*.
    for $j = 1, 2$ **do**
        Compute TD errors $x_{\phi j,n,i} = q_{\phi_j,n}(s_i, a_i) - y_{\overline{\phi},n}(r_i, s'_i)$.
        Compute critic loss $\mathcal{L}(\phi_j) = \frac{1}{bN}\sum_{i=1}^{b}\sum_{n=1}^{N} w_i \rho^k_{\tau_n}(x_{\phi j,n,i})$ using quantile Huber loss $\rho^k_{\tau_n}$ (Eq. 11).
        Update critic $\phi_j \leftarrow \phi_j - \lambda_Q \nabla_{\phi_j}\mathcal{L}(\phi_j)$.
    end for
▷ **Policy and Entropy Update**
    Compute mean Q-values $Q_\phi(s_i, \cdot) \leftarrow \frac{1}{N}\sum_{n=1}^{N} \min_{j=1,2} q_{\phi_j,n}(s_i, \cdot)$ (Eq. 14).
    Compute policy loss $J_\pi(\theta) = \frac{1}{b}\sum_{i=1}^{b} w_i \mathbb{E}_{a\sim\pi_\theta(\cdot|s_i)}[\alpha \log(\pi_\theta(a|s_i)) - Q_\phi(s_i, a)]$ (Eq. 15). Use action mask mask_i.
    Update policy $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$.
    Compute state-specific entropy target $\overline{H}(s_i) = -\beta \log(1/|A(s_i)|)$ using Eq. 17 (where $|A(s_i)|$ from mask_i).
    Compute entropy loss $J(\alpha) = \frac{1}{b}\sum_{i=1}^{b} w_i \mathbb{E}_{a\sim\pi_\theta(\cdot|s_i)}[-\alpha(\log(\pi_\theta(a|s_i)) + \overline{H}(s_i))]$ (Eq. 16). Use action mask mask_i.
    Update entropy coefficient $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha J(\alpha)$.
▷ **Target Network Update**
    Update target networks: $\overline{\phi}_j \leftarrow \tau\phi_j + (1-\tau)\overline{\phi}_j$ for $j = 1, 2$.
▷ **Priority Update**
    Update sampling probabilities $p_i \propto (\frac{1}{2N}\sum_{j,n} |x_{\phi j,n,i}|)^\omega$ for used samples (Eq. 19).
    Update probabilities in $\mathcal{D}$ for sampled indices i.
end procedure
▷ Main loop: Interact with env, store in $\mathcal{D}$, call UpdateNetworks periodically

H Detailed Simulation Parameterizations

Table 4: System parameters

Evaluation Parameters	
Parameter	**Value**
Deployment	3GPP Dense Urban Macro (see Table 6.1.2-1 in [3])
Number of cells	21
Inter-site distance	200 m
gNB height	25 m
gNB Tx power	44 dBm
Number of UEs	210 uniformly distributed
Random drops	10
Bandwidth	100 MHz (273 RBs, $N_{\text{RBG}} = 18$)
Carrier frequency	4 GHz
Subcarrier spacing	30 kHz
MIMO scheme	MU-MIMO with regularized Zero Forcing
Modulation	QPSK to 256QAM
Link adaptation	10 % Block Error Rate (BLER) target
gNB antenna panel	12x8 and 2 polarizations
Max UE layers	$\lvert L \rvert = 8$
Max UE rank	2
Max MIMO layers	16 (i.e. $\lvert L \rvert \times$ max rank)
CSI	RI and sub-band CQI & PMI
Max scheduled UEs	$\lvert U \rvert = 10$
TD scheduler	PF
UE speed	3 km/h
UE height	1.5 m
UE receiver	Maximal-ratio combining (MRC)
UE antenna	2 dual-polarized Rx antennas
Traffic model	100% Full Buffer, or 100% FTP Model 3 [1] (FTP3 with 0.5 MB packets, 20 packets/s)
Parameter Modifications for Training	
Parameter	**Value**
Number of UEs	420 uniformly distributed
Random drops	1
Traffic model	50% of UEs Full Buffer and 50% of UEs FTP Model 3 [1] (FTP3 with 1.5 kB packets, 500 packets/s)
Bandwidth	6.6 MHz (18 RBs, 18 RBGs)
Max UE layers	$\lvert L \rvert = 4$

Table 5: Training hyperparameters

Parameter	**Off-Policy**		**PPO**
Discount factor	0		0.95
Epochs per sample	1		1
Mini batch size	32		128
Actor learning rate	0.0001		0.0001
Critic learning rate	0.0001		0.0002
Hidden layers	2		2
Hidden layer size	32		32
Activation function	ReLU		ReLU
Optimizer	Adam [17]		Adam [17]
Replay buffer size	$N_{\text{gNB}} \times 1000$		4000
Prioritized replay ω	0.5		N.A.
Target smoothing coef.	0.001		N.A.
Reward scaling factor k	N.A.		0.2
Clipping parameter ϵ in PPO	N.A.		0.2
	SACD	**DSACD**	**PPO**
Critic quantiles N	N.A.	16	N.A.

Table 6: Architecture hyperparameters

	2LDS	**1LDS**
Input size	$\lvert U\rvert \times 8 + 1$	$\lvert U\rvert \times (5 + 2N_{\text{RBG}})$
Output size	$\lvert U\rvert + 1$	$N_{\text{RBG}} \times (\lvert U\rvert + 1)$
Inferences per TTI	$\lvert L\rvert \times N_{\text{RBG}}$	$\lvert L\rvert$
SAC target entropy β	0.4	0.999