Reinforcement Learning for Next-Generation Networks: *The Road to Trustworthiness*

Ahmad Nagib*, Hatem Abou-Zeid†, Hossam Hassanein*
*School of Computing, Queen's University
†Department of Electrical and Software Engineering, University of Calgary

IEEE ICMLCN, Barcelona, Spain
May 26, 2025

Practical Gap: AI for Wireless Networks

Research	Industry
SLA constraints not enforced	SLA constraints must be met
Diverse deployment scenarios not reflected	Solutions must generalize to deployment
Worst-case performance not guaranteed	Worst-case performance critical under uncertainties
AI models are black boxes	AI decisions require transparency

Gap between research and industry

Trustworthy Artificial Intelligence (AI)

- Trustworthy AI refers to artificial intelligence systems that are **explainable, fair, interpretable, robust, transparent, safe and secure**. These qualities create trust and confidence in AI systems among stakeholders and end users[1].

---

1 IBM, "What is trustworthy AI?," Accessed: 2025-05-01. (2024), [Online]. Available: https://www.ibm.com/think/topics/trustworthy-ai

Trustworthy Artificial Intelligence (AI)

- Trustworthy AI refers to artificial intelligence systems that are **explainable, fair, interpretable, robust, transparent, safe and secure**. These qualities create trust and confidence in AI systems among stakeholders and end users[1].
- Our primary focus in this tutorial is on **algorithmic dimensions of trustworthiness** in **Reinforcement Learning**, rather than explicitly covering broader cybersecurity principles.

1 IBM, "What is trustworthy AI?," Accessed: 2025-05-01. (2024), [Online]. Available: `https://www.ibm.com/think/topics/trustworthy-ai`

Outline

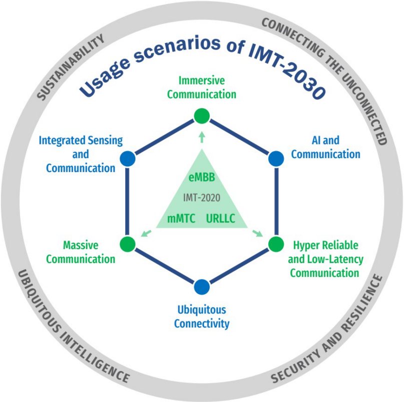
Tutorial Goals & Approach

- There are a lot of interesting and emerging advances on trustworthy RL, primarily driven by robotics applications. We will provide a "lay of the land" overview of these and discuss a select subset of them.
- We will avoid math but will occasionally introduce notation and some definitions in some slides. These however are not crucial to understand the tutorial.
- We will often introduce code snippets of implementations. This is to help turn "abstract" concepts to code and to show that the barrier to entry is not very high given the advances in Deep RL libraries.

Our main objective is to introduce concepts and familiarity of the advances in this emerging research area.

Introduction

The ITU's IMT-2030 Vision Framework



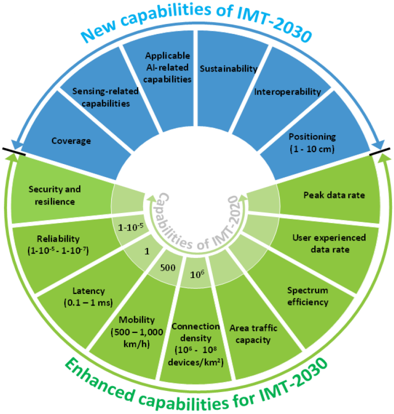


Figure: Usage scenarios and capabilities of IMT-2030[2].

[2] "Framework and overall objectives of the future development of imt for 2030 and beyond," *International Telecommunication Union (ITU) Recommendation (ITU-R)*, 2023

The Need for AI-Native Next-Gen Wireless Networks[3]

- **Escalating Network Complexity**:
  - Heterogeneous technologies and dynamic environments.
  - Manual optimization is no longer feasible.
  - **Solution**: ML can manage complexity and deliver competitive performance.
- **Model Deficiency**:
  - Traditional models rely on simplifying assumptions.
  - Unable to capture unknown dynamics and nonlinearities.
  - **Advantage**: ML captures complex patterns in NGWNs.
- **Algorithm Limitations**:
  - Optimal algorithms are often impractical due to high complexity.
  - Reliance on heuristics leads to suboptimal performance.
  - **Benefit**: ML balances performance and computational complexity.

---

[3] A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Accelerating reinforcement learning via predictive policy transfer in 6g ran slicing," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1170–1183, 2023. DOI: 10.1109/TNSM.2023.3258692

Background on RL for Next-Gen Wireless Networks (NGWNs)

Basic Reinforcement Learning (RL) Interactions

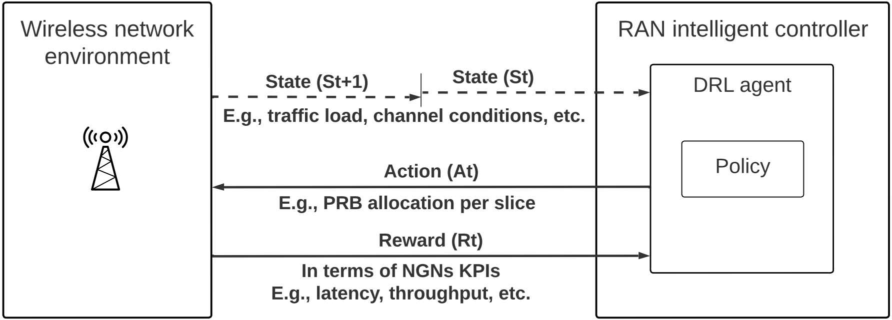


Figure: Basic interactions between a DRL agent and the network environment[4].

[4] A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Safe and accelerated deep reinforcement learning-based o-ran slicing: A hybrid transfer learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 310–325, 2024. DOI: 10.1109/JSAC.2023.3336191

RL Code Example

Traditional RL

```
# Initialize environment and agent
env = NetworkSlicingEnv()
agent = DQNAgent()

# Training loop
for episode in range(num_episodes):
    state = env.reset()
    done = False
    while not done:
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)
        agent.learn(state, action, reward, next_state)
        state = next_state

```

Listing: Traditional RL

RL Fundamentals

- **Formal Definition of RL**[5]:
  - RL is formulated as a Markov Decision Process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:
    - $\mathcal{S}$: set of states
    - $\mathcal{A}$: set of actions
    - $P(s'|s, a)$: transition probability
    - $R(s, a)$: reward function
    - $\gamma \in [0, 1)$: discount factor
  - The objective is to find a policy $\pi(a|s)$ that maximizes the expected cumulative reward:

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{1}$$

5 R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 2018, vol. 2

Why Reinforcement Learning for Network Optimization?

- **Seamless Integration with Network Control**:
  - RL naturally fits the feedback loop of network operations.
  - Adapts to operator goals and policies.

Why Reinforcement Learning for Network Optimization?

- **Seamless Integration with Network Control**:
  - RL naturally fits the feedback loop of network operations.
  - Adapts to operator goals and policies.
- **Towards Autonomous Networks**:
  - Capable of real-time decision-making in complex environments.
  - Does not require full knowledge of the network system.

Why Reinforcement Learning for Network Optimization?

- **Seamless Integration with Network Control**:
  - RL naturally fits the feedback loop of network operations.
  - Adapts to operator goals and policies.
- **Towards Autonomous Networks**:
  - Capable of real-time decision-making in complex environments.
  - Does not require full knowledge of the network system.
- **Industry Momentum**:
  - Standard bodies and vendors are promoting RL[6,7].
  - Growing recognition of RL's potential in NGWNs.

6 M. Tsampazi, S. D'Oro, M. Polese, *et al.*, "A comparative analysis of deep reinforcement learning-based xapps in o-ran," in *IEEE Global Communications Conference (GLOBECOM)*, 2023, pp. 1638–1643. DOI: 10.1109/GLOBECOM54140.2023.10437367

7 T. E. Blog, *Bringing reinforcement learning solutions to action in telecom networks*, https://www.ericsson.com/en/blog/2022/3/reinforcement-learning-solutions, [Accessed 22-01-2024], 2022

RL Applications in Next-Gen Wireless Networks

Examples of RL Applications[8]:

- **Power Control**
  - RL techniques manage transmission power levels to enhance network performance and energy efficiency.
- **Beamforming and Beam Management**
  - RL algorithms dynamically adjust beam directions and widths to improve signal quality and coverage.
- **Handover Management**
  - RL models predict optimal moments and targets for user equipment handovers to maintain seamless connectivity.
- **Network Slicing**
  - RL assists in the allocation and management of network slices to cater to diverse service requirements efficiently.

[8] A. Feriani and E. Hossain, "Single and multi-agent deep reinforcement learning for ai-enabled wireless networks: A tutorial," *IEEE Communications Surveys Tutorials*, vol. 23, no. 2, pp. 1226–1252, 2021. DOI: 10.1109/COMST.2021.3063822

Network Slicing: Use Cases

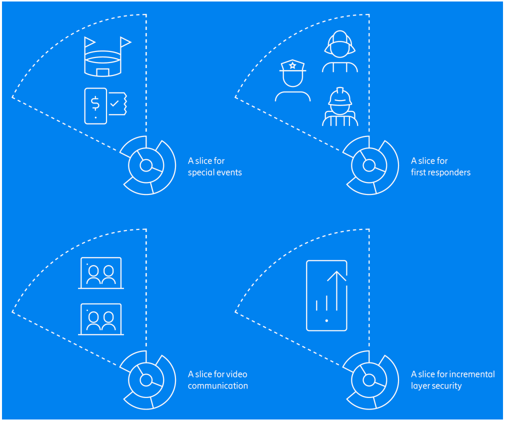


Figure: T-Mobile four network slice use-case realizations[9]

9 Ericsson, *Ericsson mobility report*, https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/november-2024, 2024

RAN Slicing: Inter-Slice Resource Allocation

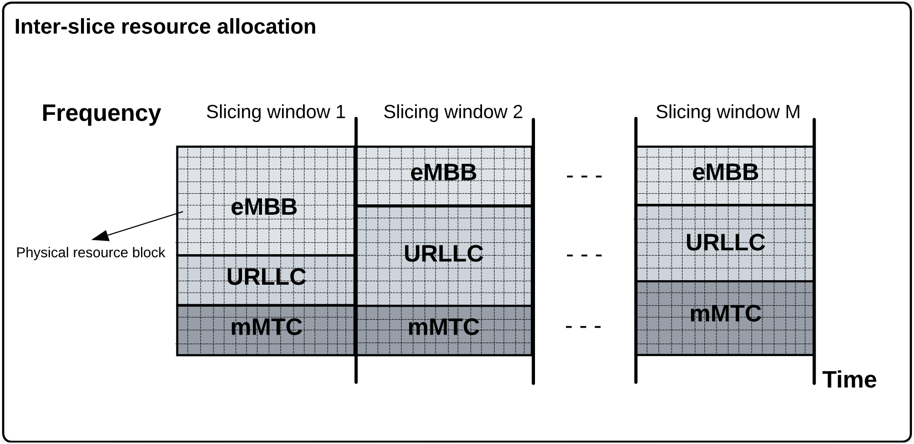


Figure: Scope and temporal resolution of inter-slice resource allocation[10]

10 A. M. Nagib, "A trustworthy deep reinforcement learning framework for slicing in next-generation open radio access networks," Ph.D. dissertation, School of Computing, Queen's University, 2024

Mapping Inter-Slice Resource Allocation to RL

- **State Representation**
  - Observed by slicing xApp in near-RT RIC
  - Represents slices' traffic contribution in previous window, Ω_{t-1}
  - Vector form:

$$\kappa = (\kappa_1, \ldots, \kappa_S)$$

  - Alternatives: Number of active users or packets per slicing window

- **Action Space**
  - Action taken at each slicing window start
  - Select PRB allocation per slice as bandwidth percentage
  - Constraint:

$$a = (b_1, \ldots, b_S), \quad \text{subject to} \sum_{s=1}^{S} b_s \leq B$$

Mapping Inter-Slice Resource Allocation to RL

- **Reward Function Design**
  - Receives network KPI feedback post action:

$$R = \left[w_u \ * \ 1 - \frac{\sum_{\mathrm{s}=1}^{\|\mathrm{S}\|} \mathrm{b_s}}{\mathrm{B}} \right] + \left[w_l \ * \ \sum_{s=1}^{\|S\|} w_s \ * \ \frac{1}{1 + e^{\ c1_s \ * \ (\ l_s \ - \ c2_s \ )}} \right]$$

  - Parameters:
    - The weights, w_u and $w_l \in [0, 1]$, reflect the importance of the goals
    - A $[w_u = 0.5, \ w_l = 0.5]$ setting means that both goals are equally important
    - w_s: Priority weight for slice s
    - l_s: Average latency for slice s

Mapping Inter-Slice Resource Allocation to RL

- **Reward Function Design**
  - Receives network KPI feedback post action:

$$R = \left[w_u \; * \; 1 - \frac{\sum_{s=1}^{\|S\|} b_s}{B} \right] + \left[w_l \; * \; \sum_{s=1}^{\|S\|} w_s \; * \; \frac{1}{1 + e^{\, c1_s \; * \; (\; l_s \; - \; c2_s \;)}} \right]$$

  - Parameters:
    - The weights, w_u and $w_l \in [0, 1]$, reflect the importance of the goals
    - A $[w_u = 0.5,\ w_l = 0.5]$ setting means that both goals are equally important
    - w_s: Priority weight for slice s
    - l_s: Average latency for slice s

However, deploying RL in real-world networks comes with significant challenges...

Practical Challenges of Reinforcement Learning

Challenges of Deploying DRL in NGWNs: Risky Exploration

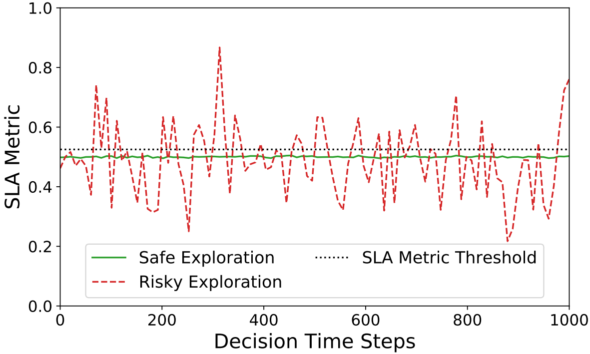


Figure: The Challenge of Risky Exploration in Wireless Networks

- **Exploration**, though limited, can occur in deployment environments.
- **Consequence**: Actions during exploration can lead to Service Level Agreements violations.

Challenges of Deploying DRL in NGWNs: Ungeneralizable Algorithms

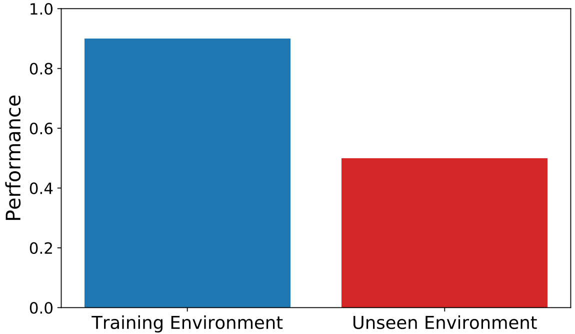


Figure: Challenges in Generalizing from Simulation to Real-World Environments

- Simulation environments often simplify real-world dynamics.
- DRL models may fail to adapt to unforeseen deployment conditions.

Challenges of Deploying DRL in NGWNs: Slow Convergence

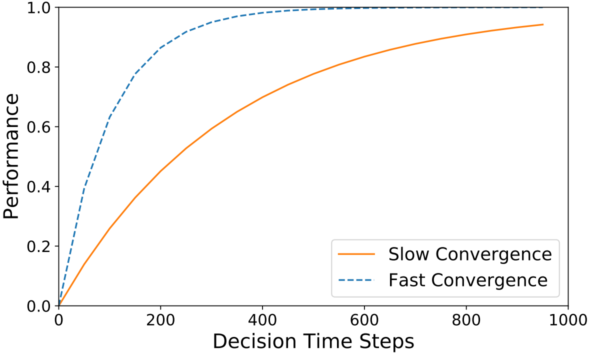


Figure: Challenges in Generalizing from Simulation to Real-World Environments

- DRL models may fail to adapt to unforeseen deployment conditions **quickly**.

Challenges of Deploying DRL in NGWNs: Non-Robust RL Algorithms

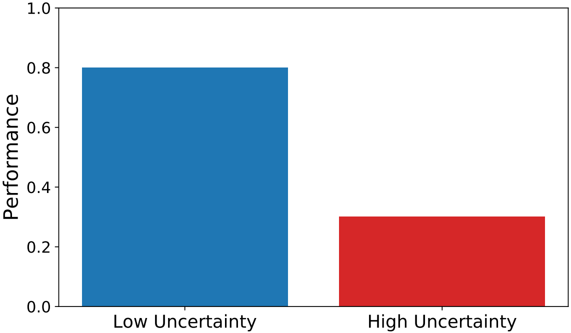


Figure: The Challenge of Non-Robust RL Algorithms in Wireless Networks

- Environment discrepancies and network stochasticity lead to uncertainties.
- **Need**: Enhance worst-case performance under uncertain network conditions.

Challenges of Deploying DRL in NGWNs: Lack of Explainability

- **Data and Model Transparency**:
  - Difficulty in interpreting DRL policies and decisions.
  - Deep neural networks act as "black boxes."
  - **Implication**: Challenges in trust, accountability, and adoption.

Trustworthiness in Reinforcement Learning

Overview of RL Trustworthiness Dimensions

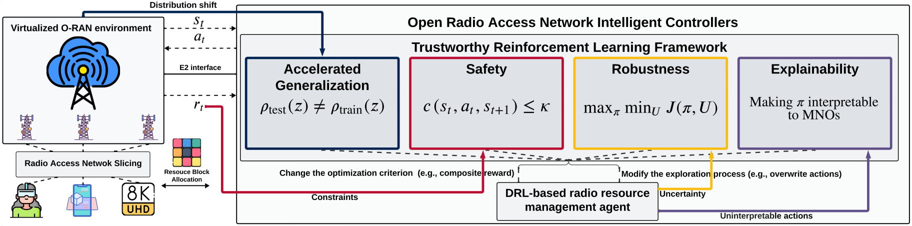


Figure: A trustworthy DRL framework for RRM in O-RANs[11]

11 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Developing trustworthy reinforcement learning applications for next-generation open radio access networks," in *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2024, pp. 137–138. DOI: 10.1109/CCECE59415.2024.10667311

Overview of RL Trustworthiness Dimensions (1/2)

- **Safety**
  - Ensuring agents avoid harmful or unsafe actions.
  - Satisfy safety constraints:

$$c\left(s_t, a_t, s_{t+1}\right) \leq \kappa \tag{2}$$

11 M. Xu, Z. Liu, P. Huang, *et al.*, "Trustworthy reinforcement learning against intrinsic vulnerabilities: Robustness, safety, and generalizability," *arXiv preprint arXiv:2209.08025*, 2022

Overview of RL Trustworthiness Dimensions (1/2)

- **Safety**
  - Ensuring agents avoid harmful or unsafe actions.
  - Satisfy safety constraints:

$$c\left(s_t, a_t, s_{t+1}\right) \leq \kappa \tag{2}$$

- **Generalizability**
  - Ability to perform well in unseen or varying environments.
  - Ensure that the policy π generalizes when:

$$\rho_{\text{deployment}}(z) \neq \rho_{\text{train}}(z) \tag{3}$$

[11] M. Xu, Z. Liu, P. Huang, *et al.*, "Trustworthy reinforcement learning against intrinsic vulnerabilities: Robustness, safety, and generalizability," *arXiv preprint arXiv:2209.08025*, 2022

Overview of RL Trustworthiness Dimensions (2/2)

- **Robustness**
  - Resilience to adversarial conditions and uncertainties.
  - Enhance the worst-case performance under uncertain variable U:

$$\max_{\pi} \min_{U} J(\pi, U) \tag{4}$$

11 M. Xu, Z. Liu, P. Huang, *et al.*, "Trustworthy reinforcement learning against intrinsic vulnerabilities: Robustness, safety, and generalizability," *arXiv preprint arXiv:2209.08025*, 2022

Overview of RL Trustworthiness Dimensions (2/2)

- **Robustness**
  - Resilience to adversarial conditions and uncertainties.
  - Enhance the worst-case performance under uncertain variable U:

$$\max_{\pi} \min_{U} J(\pi, U) \tag{4}$$

- **Explainability**
  - Making agent decisions understandable to humans.
  - Provide interpretable representations $\hat{\pi}(a|s)$ approximating $\pi(a|s)$.

11 M. Xu, Z. Liu, P. Huang, *et al.*, "Trustworthy reinforcement learning against intrinsic vulnerabilities: Robustness, safety, and generalizability," *arXiv preprint arXiv:2209.08025*, 2022

Safe Reinforcement Learning

Risky Exploration in Trustworthy DRL

- **What is Risky Exploration?**
  - The tendency of DRL algorithms to explore unsafe or suboptimal actions during learning.
  - Likely to happen when newly deployed to a network, or when significant condition changes occur
- **The Risky Exploration Problem**:
  - Service in NGWNs may have strict requirements such as latency and reliability.
  - Exploration may lead to unsafe states, e.g., severe QoS violations or service outages.
  - **Cause**:

$$\max_{\pi} \mathbb{E}[R(\pi)] \tag{5}$$

A trajectory may be maximizing the cumulative reward but can lead to unsafe interim states during exploration.

Trustworthiness Dimension: Safety

- **Definition**: Safety in RL involves ensuring agents operate without causing unintended harm or violating constraints[12].

- **Mathematical Formulation**: Depends on the constraint type, can be broadly categorized in RL into:
  - **Cumulative Constraints**
  - **Instantaneous Constraints**

12 J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015

Cumulative Constraints (Trajectory-Wise)

Definition: Cumulative constraints require that the sum or average of a certain metric (e.g., throughput, energy consumption) from the start to the current time step remains within a specified limit. These constraints are typically modelled based on the expectation of a cumulative cost signal.

Mathematical Formulation[13]**:**

$$J_{C_i}^{\pi_\theta} = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t C_i\left(s_t, a_t, s_{t+1}\right)\right] \leq \epsilon_i \tag{6}$$

- $\tau = (s_0, a_0, s_1, a_1, \ldots)$: Trajectory sampled from policy π_θ.
- γ: Discount factor.
- $C_i\left(s_t, a_t, s_{t+1}\right)$: Cost signal at time step t.
- ϵ_i: Threshold for the i^{th} cumulative constraint.

13 M. Xu, Z. Liu, P. Huang, *et al.*, "Trustworthy reinforcement learning against intrinsic vulnerabilities: Robustness, safety, and generalizability," *arXiv preprint arXiv:2209.08025*, 2022

Cumulative Constraints (Trajectory-Wise)

Constrained Markov Decision Process (CMDP) Formulation[14]:

$$\begin{aligned} \max_\theta\ J_R^{\pi_\theta} \\ \text{s.t.}\ J_{C_i}^{\pi_\theta} \leq \epsilon_i, \quad \forall i \end{aligned} \tag{7}$$

- Objective: Maximize the discounted cumulative reward $J_R^{\pi_\theta}$.
- Constraints: Ensure that each cumulative cost $J_{C_i}^{\pi_\theta}$ does not exceed its threshold ϵ_i.

Application Example:

- *Cumulative Throughput Constraint* for a service such as HD video streaming.
- Ensures that the average throughput over time abides by a specified threshold to maintain quality of service.

14 Y. Liu, A. Halev, and X. Liu, "Policy learning with constraints in model-free reinforcement learning: A survey," in *The 30th international joint conference on artificial intelligence (ijcai)*, 2021

Instantaneous (Step-Wise) Constraints

Definition: Instantaneous constraints require that specific conditions are met at each individual time step. Unlike cumulative constraints, these must hold true for every action taken by the policy.

Mathematical Representation[15]**:**

$$\max_\theta J_R^{\pi_\theta} \\ \text{s.t. } C_i(s_t, a_t, s_{t+1}) \leq \omega_i, \quad \forall t, \forall i \tag{8}$$

- ω_i: Threshold for the i^{th} instantaneous constraint.

15 Y. Liu, A. Halev, and X. Liu, "Policy learning with constraints in model-free reinforcement learning: A survey," in *The 30th international joint conference on artificial intelligence (ijcai)*, 2021

Instantaneous (Step-Wise) Constraints

Definition:

- Ensures that each action at every time step adheres to the predefined constraints.
- Thresholds are typically predefined constants based on service requirements, configurable by Mobile Network Operators (MNOs) in O-RAN scenarios.

Application Example:

- **Explicit Constraints**: Have closed-form expressions allowing direct numerical evaluation (e.g., availability of radio resources).
- **Implicit Constraints**: Lack precise closed-form representations due to system complexity (e.g., network latency).

Comparison of Cumulative vs. Instantaneous Constraints

- **Cumulative Constraints**:
  - Suitable for scenarios where long-term performance is critical.
  - Allows for occasional violations as long as the aggregate remains within limits.
- **Instantaneous Constraints**:
  - Essential for applications requiring real-time guarantees.
  - Demands strict compliance at every decision step, limiting policy flexibility.

Aspect	**Cumulative Constraints**	**Instantaneous Constraints**
Definition	Constraints on aggregated metrics over time	Constraints on metrics at each time step
Formulation	$J_{C_i}^{\pi_\theta} = \mathbb{E}\left[\sum_t \gamma^t C_i(s_t, a_t, s_{t+1})\right] \leq \epsilon_i$	$C_i(s_t, a_t, s_{t+1}) \leq \omega_i, \forall t$
Evaluation	Expectation over trajectories	Per action and state
Flexibility	Handles long-term trade-offs	Requires strict adherence each step
Complexity	Easier with CMDP techniques	More challenging due to per-step constraints
Use Cases	Average throughput, cumulative energy	Real-time resource allocation, latency requirements

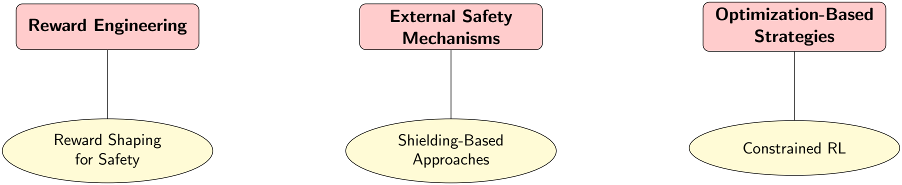
Table: Cumulative vs. Instantaneous Constraints

Changes in RL Environment to Accommodate Constraints

Adding Cost Signal to Gym Environment

```
class MyCustomEnv(gym.Env):
    def __init__(self):
        super(MyCustomEnv, self).__init__()
        # Initialization code for the environment

    def step(self, action):
        # Standard step logic
        next_state, reward, done, info = ...  # Transition
    calculations

        # Add a cost signal
        cost = self._calculate_cost(next_state, action)
        info['constraint'] = cost

        return next_state, reward, done, info

    def _calculate_cost(self, state, action):
        # Custom cost calculation logic
        cost = 1 if state violates some_condition else 0
        return cost
```

Strategies to Enhance DRL Safety



Reward Engineering

Reward Shaping

Reward Shaping

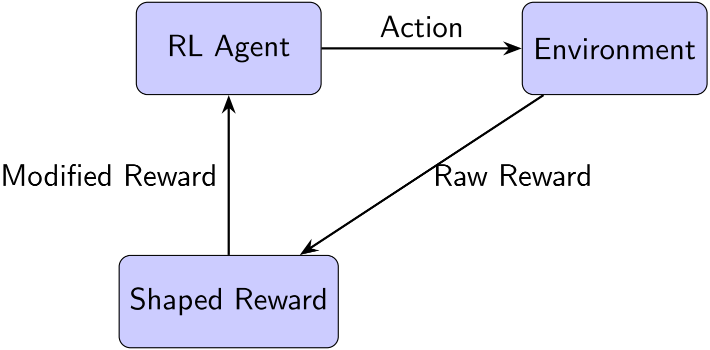


Figure: Reward shaping enhances raw rewards with safety-aware modifications.

Reward Shaping for Safety

- **Definition**: Modifies the reward function to embed safety guidance.
- **Key Idea**: Guides the agent towards safer behaviours by incentivizing safe actions and discouraging unsafe ones.

$$R'(s, a, s') = R(s, a, s') + F(s, a, s')$$

where $F(s, a, s')$ represents additional rewards or penalties for safety.

15 Z. Zhu, K. Lin, A. K. Jain, *et al.*, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 13 344–13 362, 2023. DOI: 10.1109/TPAMI.2023.3292075

Reward Shaping: Code Example

RL With Reward Shaping

```
raw_reward = env.step(action)
safety_penalty = compute_penalty(state, action)
reward = raw_reward - safety_penalty
agent.learn(state, action, reward, next_state)

```

Listing: RL with reward shaping adding penalties

Reward Shaping Example in Network Slicing

Objectives:

- Minimize resource consumption
- Fulfill SLA constraints

Approach:

- Shape rewards to guide resource allocation actions to avoid SLA violations

Example Implementation:

- Add extra penalties for violating QoS constraints such as latency.

15 A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Safe and accelerated deep reinforcement learning-based o-ran slicing: A hybrid transfer learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 310–325, 2024. DOI: 10.1109/JSAC.2023.3336191

Risk-Aware Multi-Objective Reward Function

Reward Function:

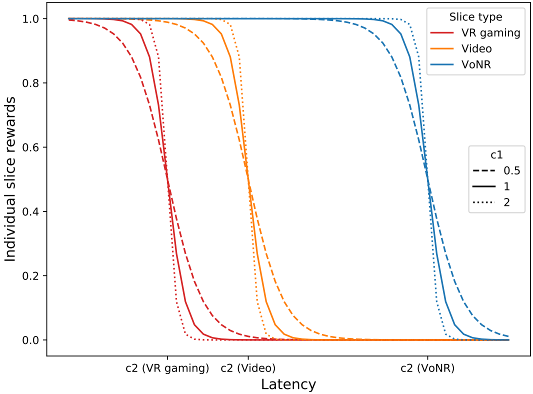
$$R = \left[w_u \left(1 - \frac{\sum_{s=1}^{|S|} b_s}{B} \right) \right] + \left[w_l \sum_{s=1}^{|S|} w_s \frac{1}{1 + e^{c1_s (l_s - c2_s)}} \right] \quad (9)$$

- $w_u, w_l \in [0, 1]$: Importance weights
- b_s: Bandwidth utilization for slice s
- B: Total available bandwidth
- w_s: Priority weight for slice s
- l_s: Average latency for slice s
- $c1_s$: Slope of the sigmoid function for slice s
- $c2_s$: Inflection point (acceptable latency for slice s)

Sigmoid-Based Reward Function

Behavior:

1. **Far from Threshold**: High reward
2. **Near Threshold**: Rapid decrease in reward
3. **Beyond Threshold**: Significant penalty



Reward Function Design Highlights

Why Sigmoid?

- Non-linear penalization near latency thresholds
- Differentiates behavior based on distance from thresholds

Weight Configuration:

- $w_u = 0.5$, $w_l = 0.5$: Equal importance
- Adjust weights based on priority

Parameter Tuning:

- $c1_s$: Determines when to start penalizing
- $c2_s$: Sets the pre-defined SLA threshold per slice

Advantages:

- Encourages safe actions near latency limits
- Attempt to balance resource utilization with SLA compliance

External Safety Mechanisms

Safety Shields

Shielding-Based Approaches Visualization

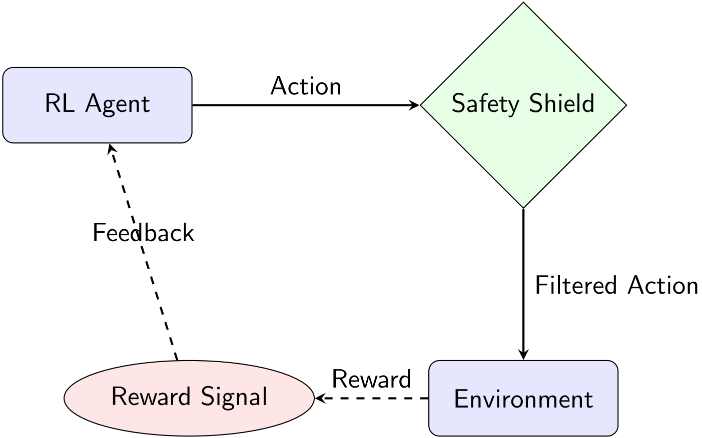


Figure: A shield intercepts and blocks unsafe actions. Optional feedback from the environment can guide the RL agent.

Safety Shields

- **Concept**:
  - Utilizes external shields to monitor actions proposed by the agent.
  - Overrides or modifies actions that may lead to unsafe states.
- **Advantages**:
  - Ensures safety without severely restricting learning.
  - Can be applied during both training and deployment.
- **Formal Definition**:

$$a_t = \begin{cases} a_t & \text{if } a_t \in \mathcal{S}(s_t) \\ a_{\text{safe}} & \text{otherwise} \end{cases}$$

where $\mathcal{S}(s_t)$ is the set of safe actions.

Shielding: Code Example

With Shield

```
action = agent.act(state)
if not is_safe(state, action):
    action = fallback_action(state)
next_state, reward, done, _ = env.step(action)
agent.learn(state, action, reward, next_state)
```

Listing: RL With Shield

Shielding Advantages and Challenges

Advantages of Shielding:

- **Enhanced Safety**: Guarantees safety during both training and deployment phases.
- **Learning Stability**: Prevents the agent from entering dangerous or highly negative reward states, aiding in stable convergence.
- **Transferability**: Shields can be reused across agents or environments if the safety rules are generalizable.

Challenges of Shielding:

- **Shield Design Complexity**: Requires domain expertise and can be challenging in environments with complex dynamics.
- **Potential Suboptimality**: Restricting risky actions can prevent optimal exploration, possibly resulting in a suboptimal policy.

Alternative Approach Example for Network Slicing

- **Study Overview**[16]: Ensure safe bandwidth allocation without SLA violations in network slicing.
- **Idea**:
  - RL agent allocates bandwidth for slices.
  - Overwrite actions expected to lead to the violation of the defined SLA thresholds.
- **Approach**:
  - A supervised learning model is used to predict the cost of actions.
  - A feasible set is created based on such predictions.

16 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Safeslice: Enabling sla-compliant o-ran slicing via safe deep reinforcement learning," in *IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, 2025

Alternative Approach Example for Network Slicing

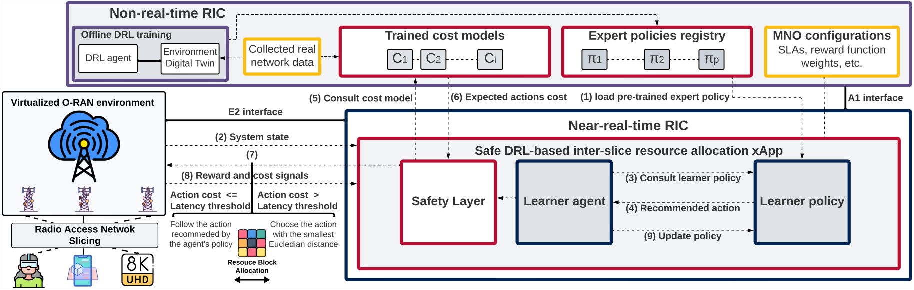


Figure: SafeSlice: A Safe DRL-Based O-RAN Slicing System[17]

17 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Safeslice: Enabling sla-compliant o-ran slicing via safe deep reinforcement learning," in *IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, 2025

Alternative Approach Example for Network Slicing

Algorithm Safe DRL-based Inter-slice RA

for each time step $t = 1$ to T **do**
  Compute action from policy: $a_t = \pi(s_t)$
  Predict cost for each slice: $C_i(s_t, a_t) = \mathbb{C}(s_t, a_t)$
  if $C_i(s_t, a_t) > \omega_i$ for any slice i **then**
    Define feasible actions: $A_f = \{a' \in \mathcal{A} \mid C_i(s_t, a') \leq \omega_i\}$
    Select closest action: $a'_t = \arg\min_{a' \in A_f} \|a' - a_t\|$
  else
    Set $a'_t = a_t$
  end if
  Execute action a'_t
  Observe reward R_t and next state s_{t+1}
  Update the agent's policy π
end for

Supervised Learning for Cost Prediction

Purpose: Predict cost function C_i using a supervised regression model $\mathbb{C}$.

- **Inputs**: State-action pairs (s_t, a_t).
- **Output**: Predicted cost $C_i(s_t, a_t)$.
- **Cost Signal**:

$$C_{\Omega_t,s} = \frac{1}{U_s}\sum_{u=1}^{U_s} L_{\Omega_t,s,u} \qquad (10)$$

$L_{\Omega_t,s,u}$: Latency experienced by user u in slice s during window Ω_t.

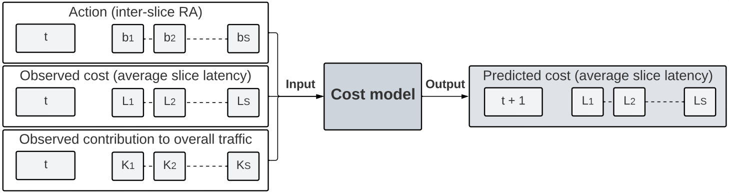


Figure: Inputs and outputs of the proposed cost model.

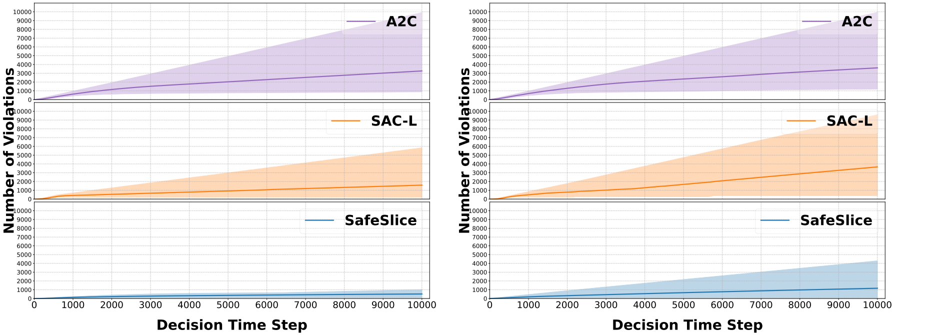
Alternative Approach Example for Network Slicing

Goal: Project RL agent's action onto a feasible space to satisfy latency constraints.

$$\begin{aligned} &\min_{a'_t} \frac{1}{2} \left\| a'_t - a_t \right\|^2 \\ &\text{s.t. } C_i\left(s_t, a'_t\right) \le \omega_i \end{aligned} \tag{11}$$

- a'_t: Feasible action closest to original action a_t.
- C_i: Cost function (latency) for slice i.
- ω_i: Latency threshold for slice i.

Instantaneous Violations Performance

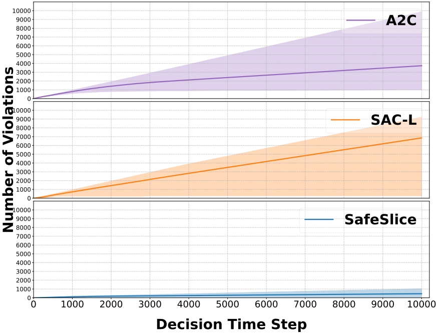


(a) Same traffic and latency threshold. (b) Same traffic, different latency threshold.

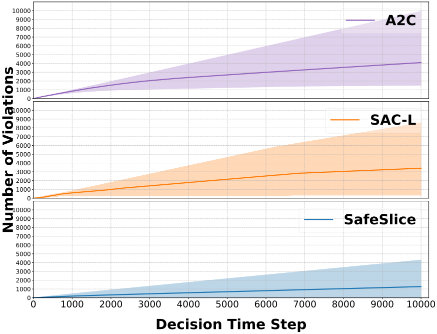
Figure: Number of instantaneous violations accumulated over decision time steps.

17 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Safeslice: Enabling sla-compliant o-ran slicing via safe deep reinforcement learning," in *IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, 2025

Instantaneous Violations Performance



(a) Different traffic, same latency threshold.

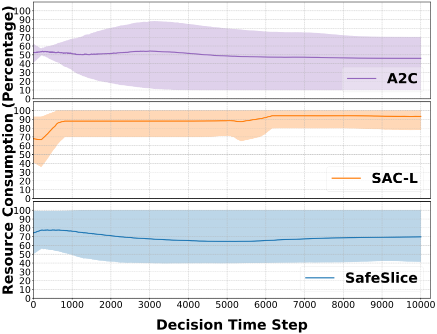


(b) Different traffic and latency threshold.

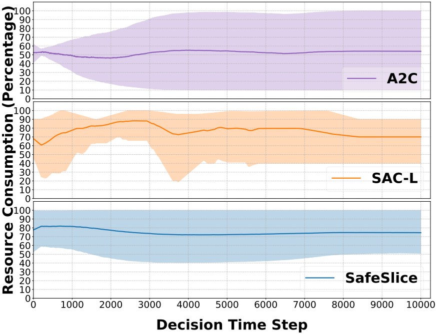
Figure: Number of instantaneous violations accumulated over decision time steps.

17 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Safeslice: Enabling sla-compliant o-ran slicing via safe deep reinforcement learning," in *IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, 2025

Resource Consumption Performance



(a) Same traffic and latency threshold.

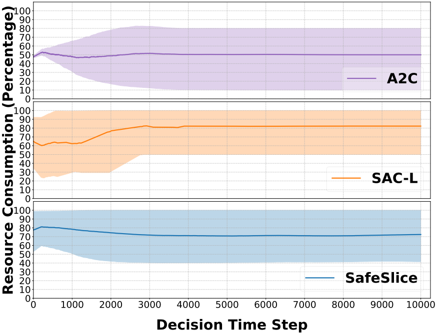


(b) Same traffic, different latency threshold.

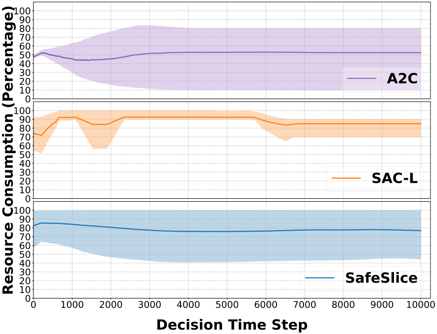
Figure: Resource consumption under the first two traffic test categories.

17 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Safeslice: Enabling sla-compliant o-ran slicing via safe deep reinforcement learning," in *IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, 2025

Resource Consumption Performance



(a) Different traffic, same latency threshold.



(b) Different traffic and latency threshold.

Figure: Resource consumption under the last two traffic test categories.

17 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Safeslice: Enabling sla-compliant o-ran slicing via safe deep reinforcement learning," in *IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, 2025

Alternative Approach Example: Digital Twin Shielding

- **Scenario**: Safe Adaptive Data Rate (SADR) system.
- **Approach**: evaluate traffic requests from the User Equipments (UEs) to identify and prevent risky actions and states that can lead to outages, improving the performance in the real network.
- **Drawbacks**: Can be infeasible for time-critical network functionalities.

[17] C. P. Robinson, A. Lacava, P. Johari, *et al.*, "Twinet: Connecting real world networks to their digital twins through a live bidirectional link," in *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*, 2024, pp. 5277–5282. DOI: 10.1109/GLOBECOM52923.2024.10901203

Alternative Approach Example: Digital Twin Shielding

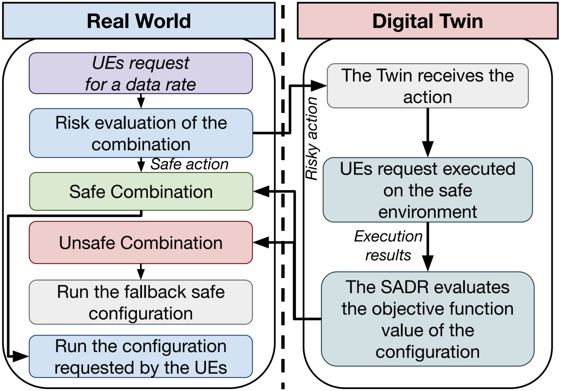


Figure: Digital twin as a safety shield[18].

18 C. P. Robinson, A. Lacava, P. Johari, *et al.*, "Twinet: Connecting real world networks to their digital twins through a live bidirectional link," in *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*, 2024, pp. 5277–5282. DOI: 10.1109/GLOBECOM52923.2024.10901203

Alternative Approach Example: Time Series Forecasting

- **Objective**: Enhance DRL performance under traffic demand uncertainties in network slicing.
- **Challenge**: DRL agents may not quickly adapt to sudden changes in traffic demand.
- **Solution**: Incorporate a *forecasting module* to guide DRL agents.

[18] A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "How does forecasting affect the convergence of drl techniques in o-ran slicing?" In *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023, pp. 2644–2649. DOI: 10.1109/GLOBECOM54140.2023.10437780

Alternative Approach Example: Time Series Forecasting

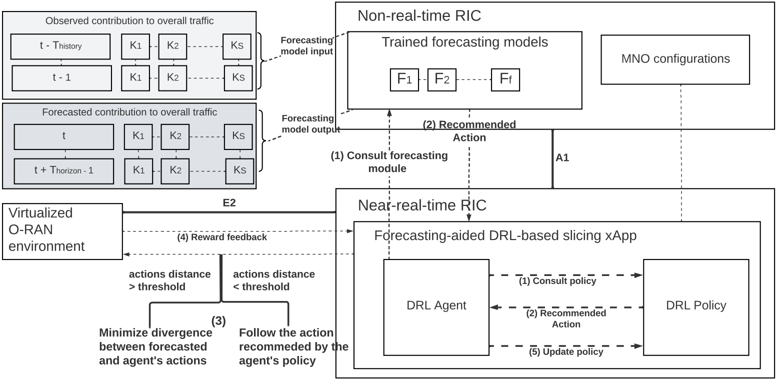


Figure: Forecasting-aided DRL-based O-RAN slicing: Interaction steps[19].

19 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "How does forecasting affect the convergence of drl techniques in o-ran slicing?" In *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023, pp. 2644–2649. DOI: 10.1109/GLOBECOM54140.2023.10437780

Alternative Approach Example: Time Series Forecasting

Algorithm Forecasting-Aided DRL Approach

while $t < T$ **do**
  Forecast future demand $\hat{\boldsymbol{\kappa}}$ using model $\mathbb{F}$
  Generate action a_{forecast} based on $\hat{\boldsymbol{\kappa}}$
  Obtain DRL agent's action $a_\pi = \pi(\boldsymbol{\kappa})$
  if $\gamma(a_\pi, a_{\text{forecast}}) > \gamma_{\text{threshold}}$ **then**
    Compute midpoint $a_{\text{distilled}} = \frac{1}{2}(a_\pi + a_{\text{forecast}})$
    Execute action $a_{\text{distilled}}$
  else
    Execute action a_π
  end if
  Receive reward R and update policy π
end while

Forecasting Module Guiding DRL Agent

- Forecast future traffic demand $\hat{\boldsymbol{\kappa}}$ using model $\mathbb{F}$.
- Generate action a_{forecast} based on $\hat{\boldsymbol{\kappa}}$.
- DRL agent's action: $a_\pi = \pi(\boldsymbol{\kappa})$.
- Measure difference between actions:

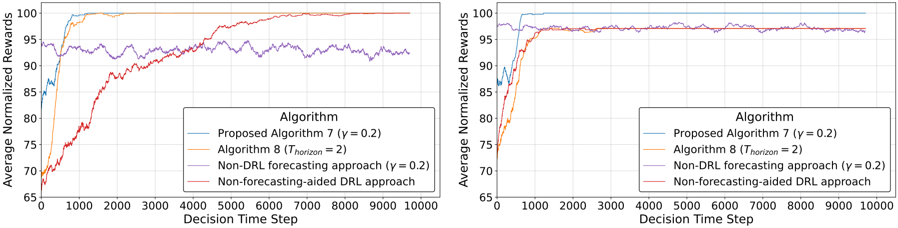
$$\gamma(a_\pi, a_{\text{forecast}}) = \sqrt{\sum_{s=1}^{S} (a_{\pi,s} - a_{\text{forecast},s})^2} \tag{12}$$

- If $\gamma(a_\pi, a_{\text{forecast}}) > \gamma_{\text{threshold}}$, adjust action.

TSF with Action Adjustment: Code Example

```python
# Initialize environment, agent, and forecast model
env = NetworkSlicingEnv()
forecast_model = LSTMForecastModel() # Time Series Forecasting Model
agent = DQNAgent()
distance_threshold = 0.5 # Threshold for action adjustment

# Training loop with forecasting and action adjustment
for episode in range(num_episodes):
    state = env.reset()
    done = False
    while not done:
        # Get action from RL policy
        action_rl = agent.act(state)

        # Forecast future traffic
        future_traffic = forecast_model.predict(env.current_traffic())

        # Determine forecast-based action
        action_forecast = determine_action_based_on_forecast(future_traffic)

        # Compare actions and adjust if needed
        if abs(action_rl - action_forecast) > distance_threshold:
            action = action_forecast
        else:
            action = action_rl

        # Take action in the environment
        next_state, reward, done, _ = env.step(action)

        # Update RL agent with the experience
        agent.learn(state, action, reward, next_state)
```

Forecasting-Aided DRL-based Slicing



(a) Traffic pattern 1

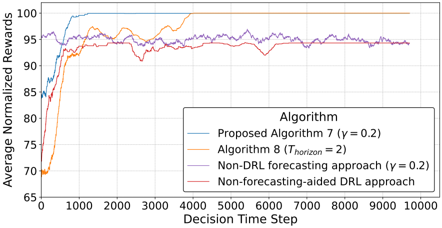
(b) Traffic pattern 2

Figure: Convergence performance of the proposed forecasting-aided approach under 2 different traffic patterns.

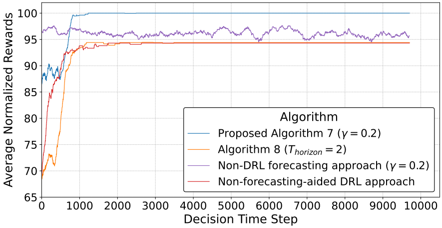
---

19 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "How does forecasting affect the convergence of drl techniques in o-ran slicing?" In *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023, pp. 2644–2649. DOI: 10.1109/GLOBECOM54140.2023.10437780

Forecasting-Aided DRL-based Slicing



(a) Traffic pattern 3



(b) Traffic pattern 4

Figure: Convergence performance of the proposed forecasting-aided approach under 2 different traffic patterns.

19 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "How does forecasting affect the convergence of drl techniques in o-ran slicing?" In *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023, pp. 2644–2649. DOI: 10.1109/GLOBECOM54140.2023.10437780

Forecasting-Aided DRL-based Slicing

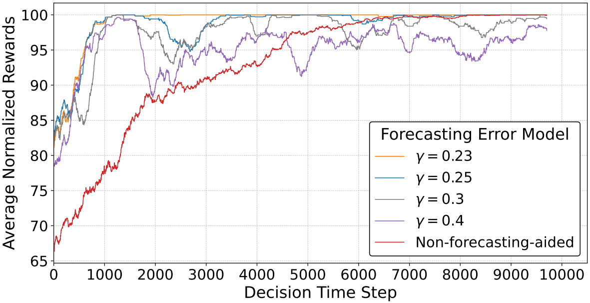


Figure: Convergence performance of the proposed approach under different forecasting error models (traffic pattern 1).

19 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "How does forecasting affect the convergence of drl techniques in o-ran slicing?" In *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023, pp. 2644–2649. DOI: 10.1109/GLOBECOM54140.2023.10437780

Forecasting-Aided DRL-based Slicing

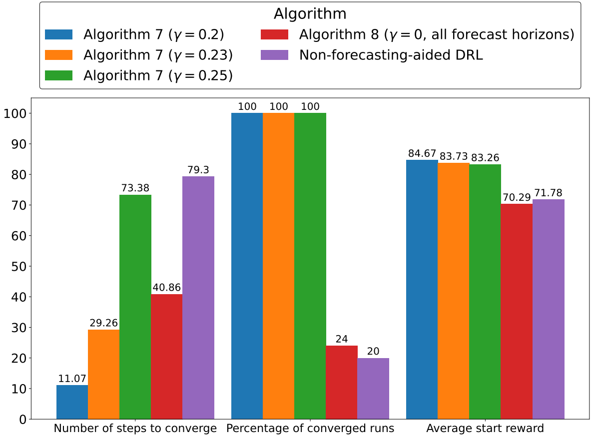


Figure: Convergence performance averaged over multiple runs (the higher the better except for the number of steps to converge).

Optimization-Based Strategies

Constrained RL

Constrained RL

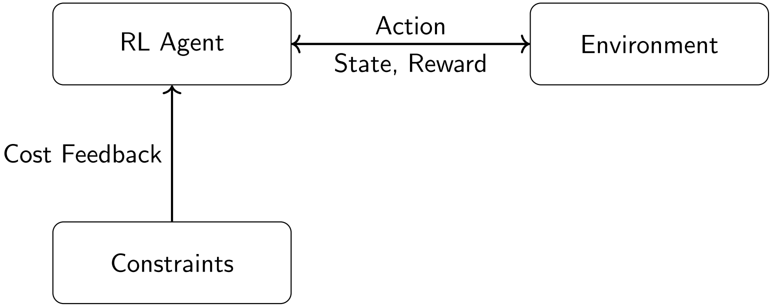


Figure: Constrained RL with explicit feedback on constraints.

Maximize expected return, subject to the expected cost not exceeding a given threshold

Constrained RL

- **Concept**:
  - Incorporating safety constraints directly into the learning process.
- **Mathematical Formulation**:

$$\max_{\pi} \quad \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{13}$$

$$\text{s.t.} \quad \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t)\right] \leq C_{\max} \tag{14}$$

- **Solution Method Example**[20]:
  - **Lagrangian Relaxation**:
    - Convert constrained optimization into an unconstrained one using Lagrange multipliers.
    - Iteratively update policy π and multipliers λ.

[20] Y. Liu, A. Halev, and X. Liu, "Policy learning with constraints in model-free reinforcement learning: A survey," in *The 30th international joint conference on artificial intelligence (ijcai)*, 2021

Lagrangian-Based Safe Reinforcement Learning

Formulation:

- Optimize a policy π to maximize task reward $\mathbb{E}_\pi[\sum_t \gamma^t r(s_t, a_t)]$.
- Satisfy constraints $\mathbb{E}_\pi[c(s_t, a_t)] \leq \epsilon$.

Lagrangian Objective:

$$\mathcal{L}(\pi, \lambda) = \mathbb{E}_\pi \left[\sum_t \gamma^t r(s_t, a_t) \right] - \lambda \cdot \left(\mathbb{E}_\pi[c(s_t, a_t)] - \epsilon \right),$$

where:

- λ is the Lagrange multiplier (penalty term).
- ϵ is the constraint threshold.

Dynamic Penalty:

- λ is updated dynamically based on constraint violations.
- Allows adaptive trade-off between task reward and constraint satisfaction.

Integrating Constraints in the Reward Function

Modified Reward:

$$r'(s,a) = r(s,a) - \alpha \cdot c(s,a),$$

where:

- $\alpha > 0$ is a fixed weight.
- $c(s,a) \geq 0$ measures the constraint violation.

Characteristics:

- Simplifies the optimization problem by embedding the constraint directly into the reward signal.
- Requires careful tuning of α to balance task performance and constraint satisfaction.

Limitation:

- Fixed α does not adapt to varying degrees of constraint violation.
- May lead to suboptimal solutions if α is not chosen carefully.

Key Differences Between the Two Approaches

Aspect	**Reward-Based**	**Lagrangian-Based**
Penalty Weight	Fixed (α)	Dynamic (λ)
Adaptability	Non-adaptive	Adaptive to violations
Flexibility	Simple to implement	Handles multiple constraints
Trade-Off Tuning	Manual tuning required	Automatically balances
Complexity	Low	Higher (requires λ-update)

Key Differences between Shielding and Constrained RL

- **Constrained Reinforcement Learning (CRL)** incorporates constraints directly into the optimization problem, ensuring safety as part of the policy learning process.
- **Shielding**, on the other hand, works as an external mechanism that intervenes during action selection.
- Shielding can operate on top of existing RL algorithms, ensuring safety without modifying the underlying learning objective.
- In contrast, CRL mathematically formulates the policy optimization problem to include constraints on expected costs.

Constrained RL: Code Example

Constrained RL

```
# CRL Training
for ep in range(num_eps):
    state = env.reset()
    done = False
    while not done:
        action = agent.act(state)
        next_state, reward, done, info = env.step(action)
        cost = info['constraint']
        agent.learn(state, action, reward, next_state, cost)
        state = next_state
```

Listing: Constrained RL

Overview of RL Trustworthiness Dimensions

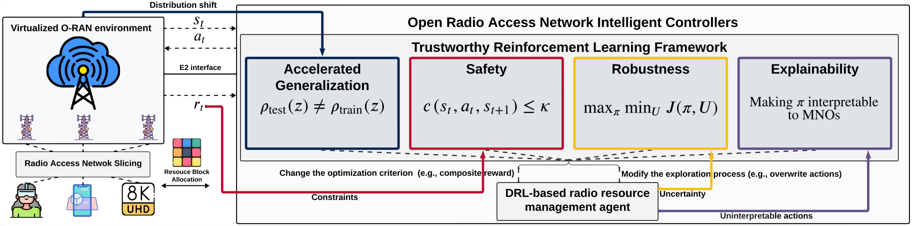


Figure: A trustworthy DRL framework for RRM in O-RANs[21]

21 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, “Developing trustworthy reinforcement learning applications for next-generation open radio access networks,” in *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2024, pp. 137–138. DOI: 10.1109/CCECE59415.2024.10667311

Generalizable Reinforcement Learning

Strategies to Enhance DRL Generalization

- **Learning-Aided Approaches**
  - Transfer Learning
  - Multi-Task Learning
  - Meta-Learning
  - Ensemble Methods
- **Data Augmentation and Manipulation Approaches**
  - Domain Randomization
  - Synthetic Data Generation
- **RL Algorithmic and Design Enhancements**
  - Initialization Strategies
  - Inherent DRL Algorithm Properties
  - Mode of Training / Deployment
- **Domain Knowledge Integration**
  - Problem Structure Optimization
  - Hybrid Solutions with Conventional Methods

Learning-Based Approaches

Policy Transfer

What Does Transferring a Policy Mean?

Policy Transfer:

- A policy encodes knowledge about how to act in an environment.
- Policy transfer refers to the process of taking a policy learned in one environment or task (the source task) and using it in another environment or task (the target task) to improve learning efficiency, performance, or adaptability.

---

21 Z. Zhu, K. Lin, A. K. Jain, *et al.*, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 13 344–13 362, 2023. DOI: 10.1109/TPAMI.2023.3292075

How do we Transfer a Policy?

Representation of Policies:

- In Deep RL, policies are typically represented by neural networks:

$$\pi(a \mid s; \theta), \text{ where } \theta \text{ are the network parameters.}$$

- Policy transfer means transferring these learned parameters (or a portion of them) to the target task, potentially with modifications.

- Policy transfer can also be performed by using the output/actions of expert policies to guide the agent in learning a new policy.

Why Policy Transfer?

Knowledge Reuse:

- A policy learned for one task (**source policy**) can provide valuable insights for solving a different but related task (**target task**).
- Example: If an agent has learned to navigate a **simple maze**, that policy can be reused when navigating a **more complex maze**.

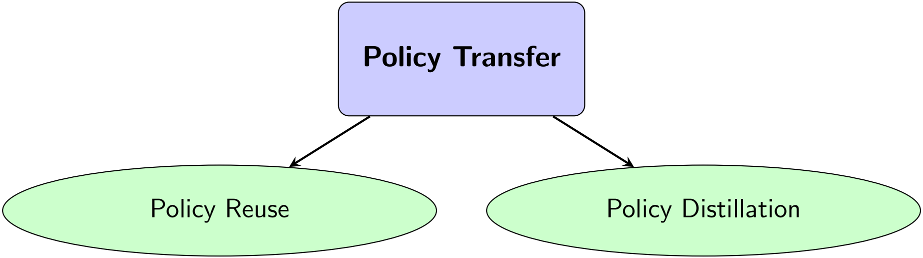
Accelerating Learning:

- Reusing a policy can reduce the exploration effort needed in new environments.
- Guides the agent toward **promising regions** of the state-action space.

Improving Sample Efficiency:

- Learning from scratch requires many interactions with the environment.
- Policy transfer builds on prior experience, reducing sample requirements.

Policy Transfer Strategies



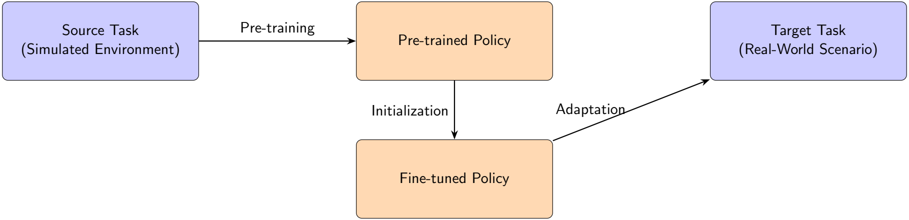
Policy Reuse: Deployment Examples

1. **Initialization with Expert Policy[22]:**

$$\pi_{\text{learner}}(t = 0) = \pi_{\text{expert}}(t = N)$$

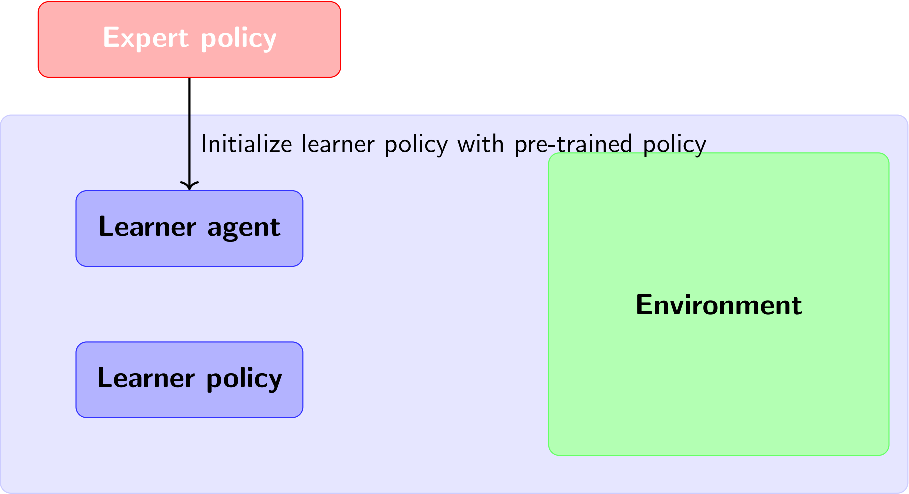
22 A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Accelerating reinforcement learning via predictive policy transfer in 6g ran slicing," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1170–1183, 2023. DOI: 10.1109/TNSM.2023.3258692

Basic Policy Reuse: Initialization with Pre-trained Policies + Fine tuning

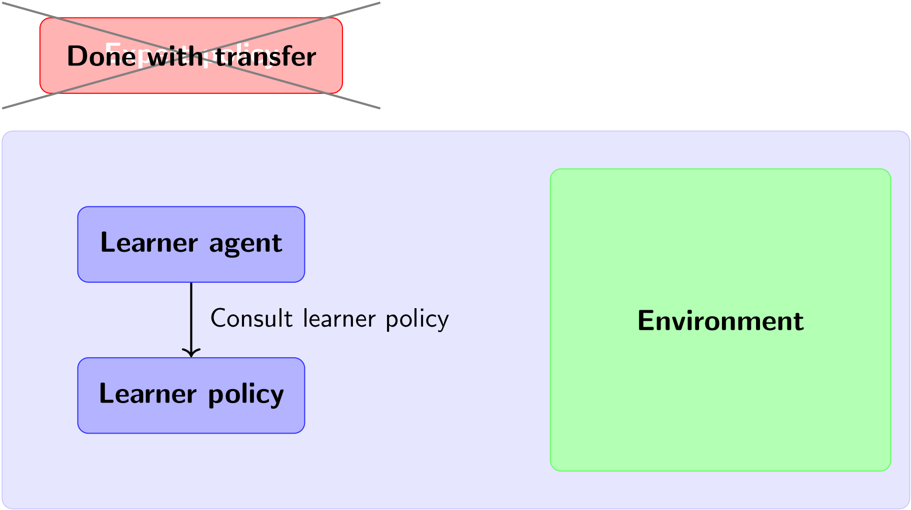


The idea is to initialize your policy for a new target task with a pre-trained policy and then fine-tune it with interactions with the target task/environment.

Policy Reuse: Initialization with Expert Policy



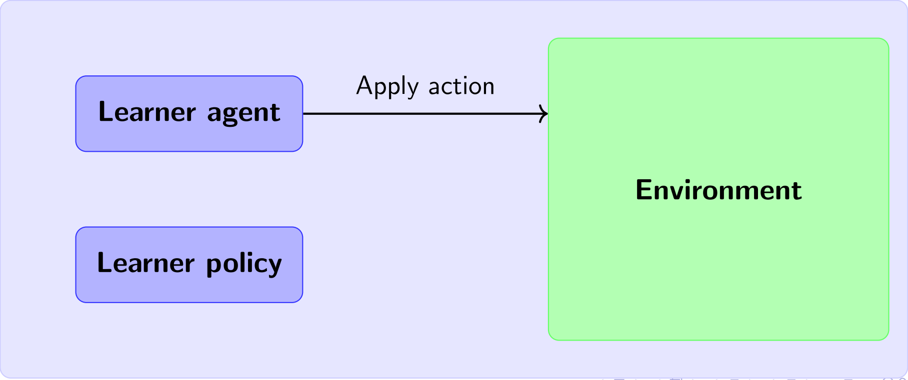
Policy Reuse



Policy Reuse: Update the Policy with Environment Interaction

~~**Done with transfer**~~

Continue with the normal learning process



Basic Policy Transfer + Fine-tuning

Implementation

```
# Load pre-trained model
pretrained_model = load_model('source_task_model.pth')
agent = DQNAgent()
# Initialize with learned parameters (weights and biases)
agent.load_state_dict(pretrained_model.state_dict())

# Fine-tuning loop on target task
for episode in range(num_finetune_episodes):
    state = env.reset()
    done = False

    while not done:
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)
        agent.learn(state, action, reward, next_state)
        state = next_state

```

Policy Reuse: Deployment Examples

1. **Initialization with Expert Policy[23]:**

$$\pi_{\text{learner}}(t = 0) = \pi_{\text{expert}}(t = N)$$

2. **Consulting Expert Policy During Learning[24]:**

$$\pi = (1 - \theta)\pi_{\text{learner}} + \theta\pi_{\text{expert}}$$

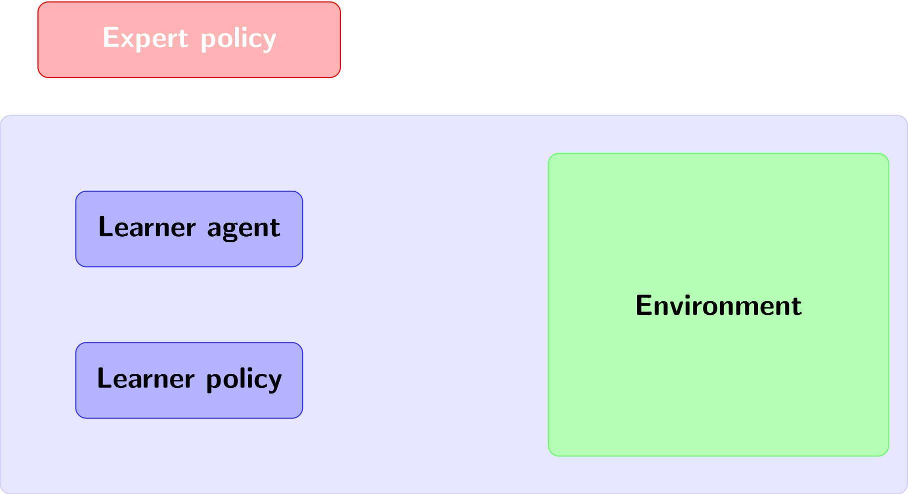
- θ: Transfer rate
- θ decays over time to favor learner policy
- Allows the new policy to occasionally learn on a 'clean slate' versus always following the expert policy.

---

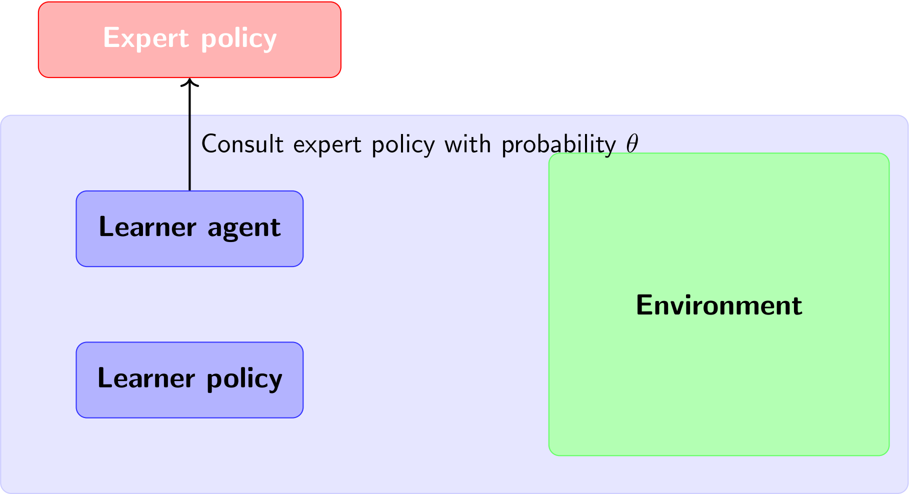
23 A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Accelerating reinforcement learning via predictive policy transfer in 6g ran slicing," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1170–1183, 2023. DOI: 10.1109/TNSM.2023.3258692

24 A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Safe and accelerated deep reinforcement learning-based o-ran slicing: A hybrid transfer learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 310–325, 2024. DOI: 10.1109/JSAC.2023.3336191

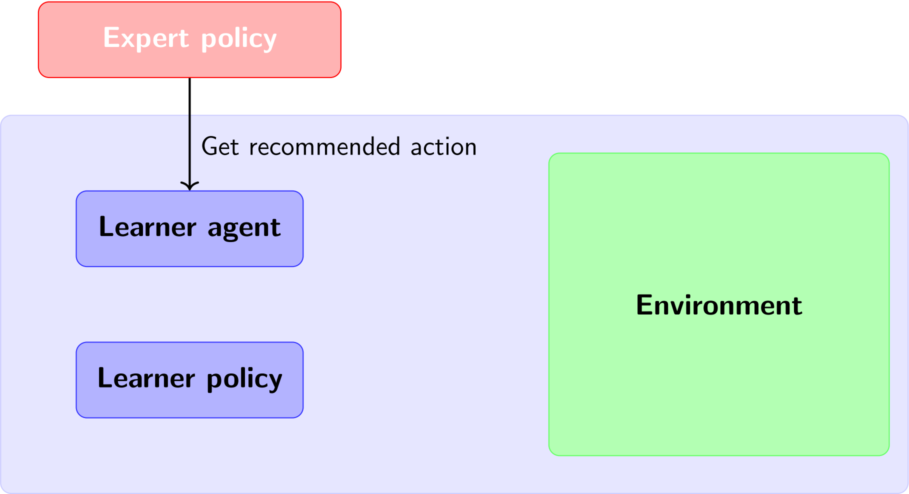
Policy Reuse: Consulting Expert Policy During Learning



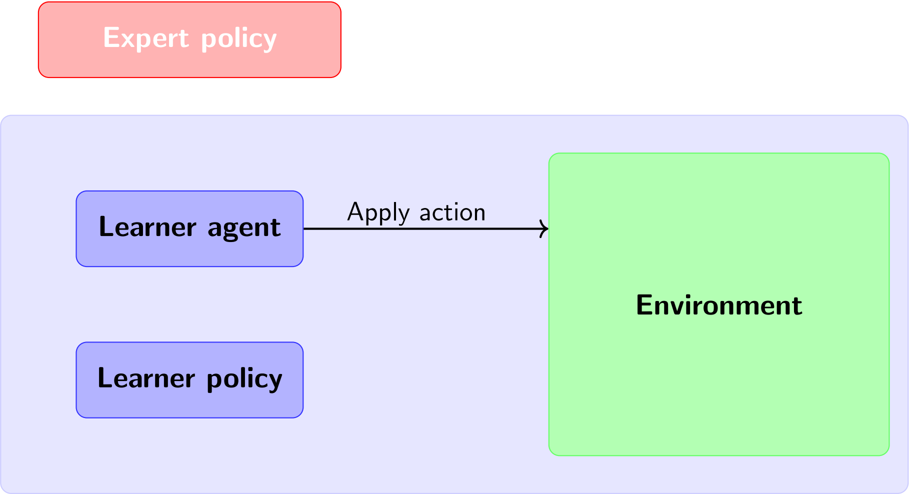
Policy Reuse: Step 1



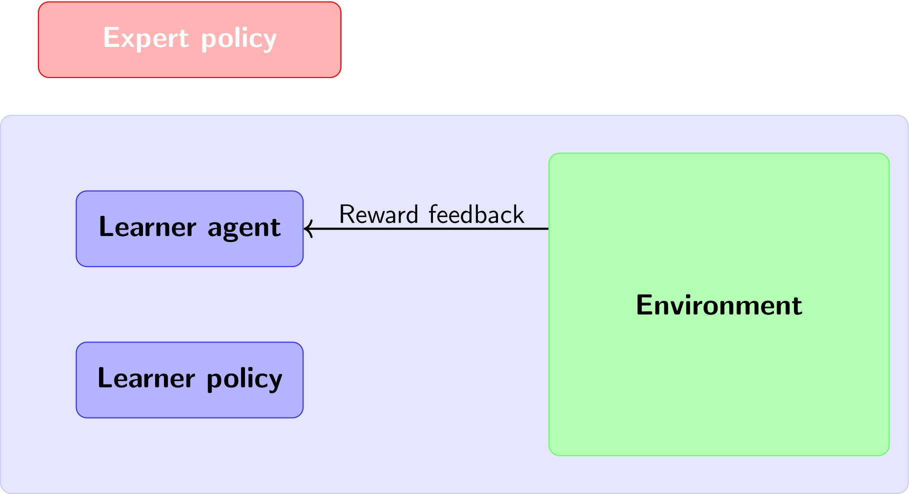
Policy Reuse: Step 2



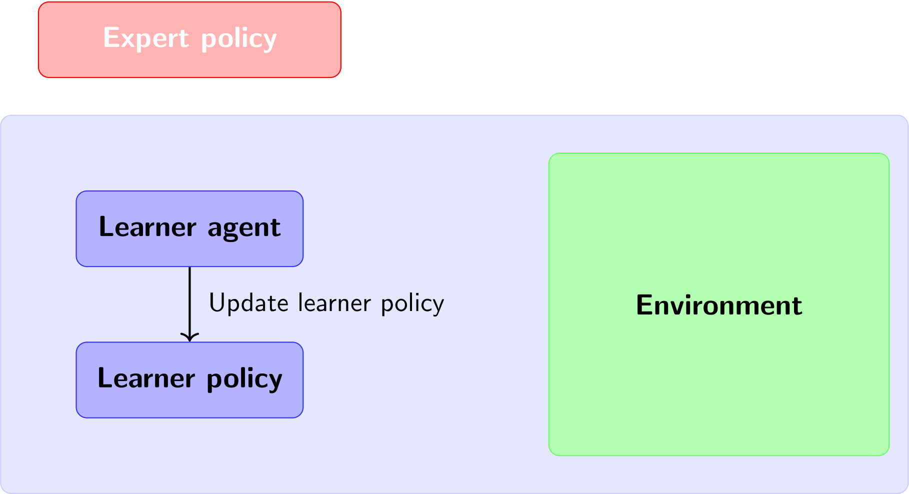
Policy Reuse: Step 3



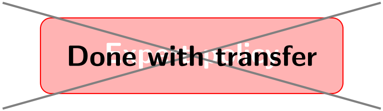
Policy Reuse: Step 4



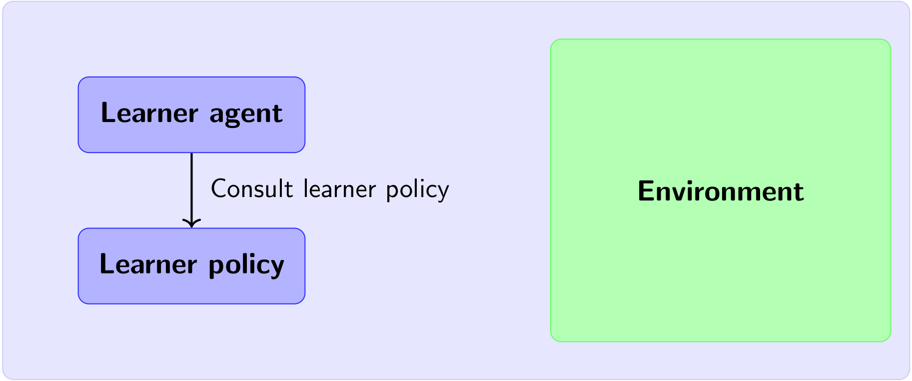
Policy Reuse: Step 5



Policy Reuse: Step 6



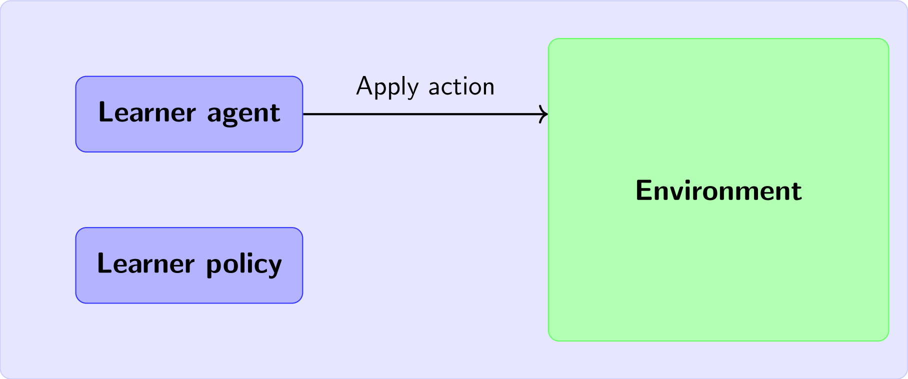
After t time-steps



Policy Reuse: Final Step

~~**Done with transfer**~~

Continue with the normal learning process



Policy Reuse Algorithm

Policy Reuse in Python

```python
import random

def policy_reuse(expert_policy, learner_policy, theta, T, beta, nu, total_steps):
    for t in range(total_steps):
        if t < T:
            x = random.uniform(0, 1)
            if x <= theta:
                action = expert_policy.choose_action(state)
            else:
                action = learner_policy.choose_action(state)
        else:
            action = learner_policy.choose_action(state)

        execute_action(action)
        reward = calculate_reward()
        learner_policy.update(reward, action)

        theta *= nu  # Decay transfer rate

```

Listing: Policy Reuse in Python

Policy Reuse Example in Network Slicing

- **Study Overview**: Utilized policy reuse as one of the baselines to adapt a DRL agent to real-world network slicing scenarios [25].
- **Approach**: Pre-trained agents in source network slicing environments and employed policy reuse to help newly deployed RL agents adapt to target environments with various traffic demand profiles.
- **Results**: Improved DRL generalization in some situations.

25 A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Safe and accelerated deep reinforcement learning-based o-ran slicing: A hybrid transfer learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 310–325, 2024. DOI: 10.1109/JSAC.2023.3336191

Policy Reuse Example in Network Slicing

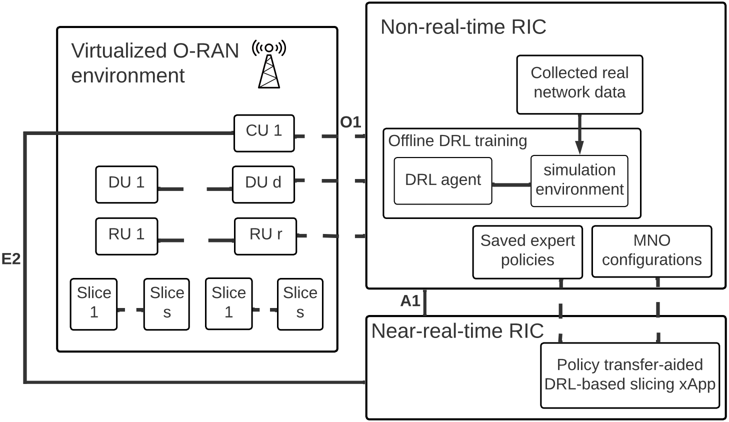


Figure: The policy transfer-aided O-RAN system architecture.

Example Study: Simulation Parameters Settings

Parameter	Video	VoNR	VR Gaming
Scheduling Algorithm	Round-robin per 1 ms slot		
Slicing Window Size	PRB allocation among slices every 100 scheduling time slots		
Packet Interarrival Time	Truncated Pareto (mean = 6 ms, max = 12.5 ms)	Uniform (min = 0 ms, max = 160 ms)	Real VR gaming dataset [19]
Packet Size	Truncated Pareto (mean = 100 B, max = 250 B)	Constant (40 B)	Real VR gaming dataset [19]
Number of Users	Poisson (max = 43, mean = 20)	Poisson (max = 104, mean = 70)	Poisson (max = 7, mean = 1)

Results: Similar Traffic

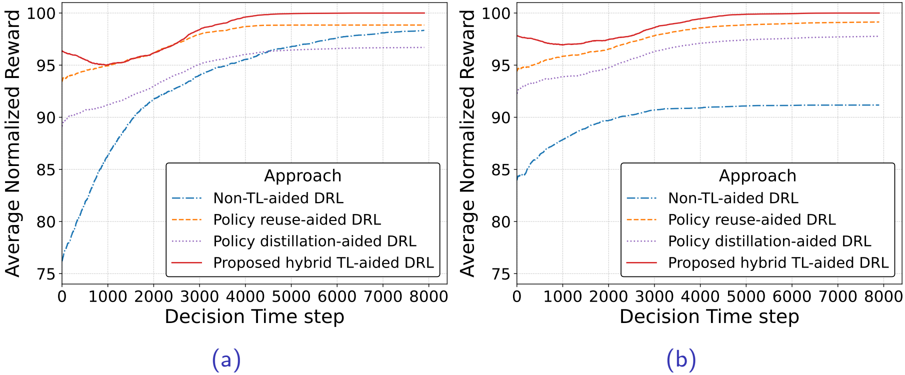
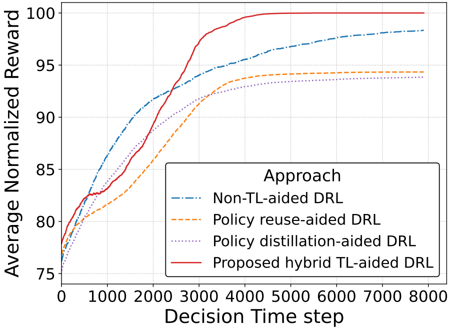
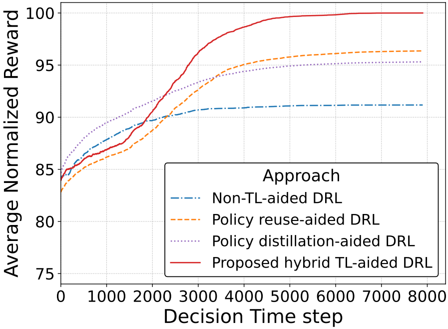


Figure: Reward convergence performance of the proposed policy transfer algorithms: a) and b) traffic patterns 1 and 2 guided by an expert policy trained using a similar traffic pattern (average of best 64 runs).

Results: Different Traffic (poor performance of policy reuse)



(a)



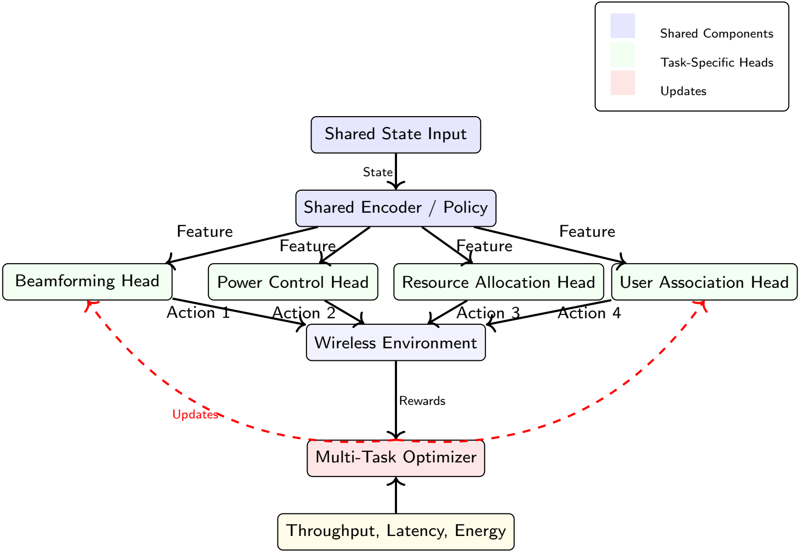
(b)

Figure: Reward convergence performance of the proposed policy transfer algorithms: a) and b) traffic patterns 1 and 2 guided by an expert policy trained using a different traffic pattern (average of best 64 runs).

Learning-Based Approaches

Multi-Task Reinforcement Learning

Multi-Task RL



Multi-Task Reinforcement Learning

- **Concept**:
  - Train an agent across multiple tasks to learn a generalized policy.
- **Method**:
  - Agent learns multiple tasks simultaneously, typically using a shared network architecture for parts of the policy/value function and task-specific components for others.
  - Learn via joint optimization over all tasks.
- **Benefit**:
  - Improves generalization by using domain information from related tasks or across network configurations (e.g. sizes, topologies, or traffic patterns).
  - Reduces costly per-task or per-scenario retraining

[25] N. Vithayathil Varghese and Q. H. Mahmoud, "A survey of multi-task deep reinforcement learning," *Electronics*, vol. 9, no. 9, 2020, ISSN: 2079-9292. DOI: 10.3390/electronics9091363. [Online]. Available: https://www.mdpi.com/2079-9292/9/9/1363

Example: Multi-Task DRL for Dynamic MAC Scheduling

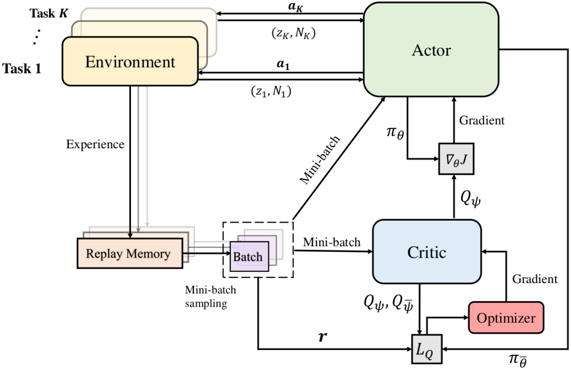


Figure: Illustration of Multi-Task Deep RL in dynamic MAC scheduling[26].

26 Z. Chen, X. Sun, Y. Jin, *et al.*, "Multi-task reinforcement learning-based multiple access for dynamic wireless networks," *IEEE Transactions on Mobile Computing*, pp. 1–15, 2025. DOI: 10.1109/TMC.2025.3559676

Multi-Task RL Code Snippet

Training Across Tasks with Shared Policy

```
tasks = [Env1(), Env2(), ..., EnvT]
agent = RLAgent(shared_policy=True)

for episode in range(num_episodes):
    # Randomly pick a task for this episode
    task_idx = sample_from(0, len(tasks)-1)
    env = tasks[task_idx]
    task_ctx = get_task_context(task_idx) # one-hot or learned
    embedding

    state = env.reset()
    while True:
        # Policy may use task_idx or context
        action = agent.select_action(state, task_ctx)
        next_state, reward, done, _ = env.step(action)
        agent.store_transition(task_ctx, state, action,
                               reward, next_state, done)
        if done: break
        state = next_state

```

DRL Generalization: Multi-Task Learning vs. Transfer Learning

Aspect	**Multi-Task Learning**	**Transfer Learning**
Objective	Learn shared representations across tasks	Transfer knowledge to a new task
What is Shared?	Parameters, features across tasks	Pre-trained weights from source task
Data Requirement	Multiple tasks at once	Source task data; target task optional

Data Augmentation-Based Approaches

Domain-Randomization

Domain Randomization

- Domain Randomization is a training technique in RL that exposes the agent to diverse simulated environments with randomized variations. The randomness ensures the agent does not overfit to specific conditions and instead learns robust policies.

- Goal: Enable the agent to learn a policy that generalizes well to new, unseen environments.

- Especially useful for:
  - Handling variability in environments (more so than tasks).
  - Sim-to-real transfer for production environments.
  - Requires careful thought and industry/applied domain expertise to know what needs randomization!

Example: Network Slicing with a Digital Twin

- **Concept**:
    - A digital twin is a live-synced virtual model of the real environment.
    - RL agent trains in the twin and deploys to the real network.
- **Method**:
    - Collect real-time data from the network.
    - Update the digital twin environment.
    - Use the twin to simulate actions and train the agent safely.
- **Benefit**:
    - Enables continuous learning and safe policy refinement.
    - Bridges sim-to-real generalization by minimizing domain gap.
- **Drawbacks**:
    - High fidelity twin modeling and real-time data syncing can be resource intensive.

---

26 Z. Zhang, Y. Huang, C. Zhang, *et al.*, "Digital twin-enhanced deep reinforcement learning for resource management in networks slicing," *IEEE Transactions on Communications*, vol. 72, no. 10, pp. 6209–6224, 2024. DOI: 10.1109/TCOMM.2024.3395698

Digital Twin Enhanced RL

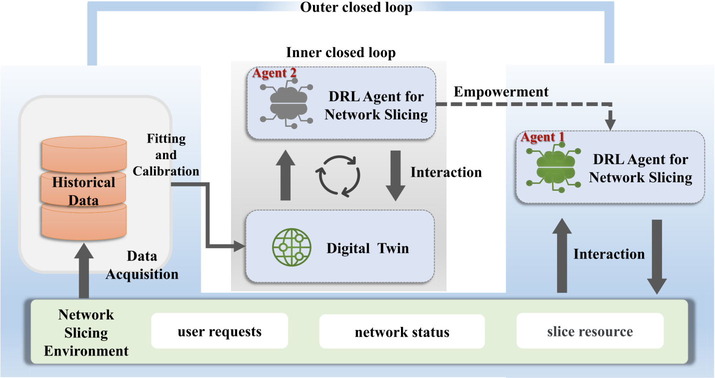


Figure: Digital twin provides a training ground synchronized with real network conditions[27].

27 Z. Zhang, Y. Huang, C. Zhang, et al., "Digital twin-enhanced deep reinforcement learning for resource management in networks slicing," *IEEE Transactions on Communications*, vol. 72, no. 10, pp. 6209–6224, 2024. DOI: 10.1109/TCOMM.2024.3395698

Digital Twin RL: Code Example

Training in Twin, Deploying in Real

```
for episode in range(num_episodes):
    twin_env.update_from_real(real_env.measurements())
    state = twin_env.reset()
    done = False
    while not done:
        action = agent.select_action(state)
        next_state, reward, done, _ = twin_env.step(action)
        agent.learn(state, action, reward, next_state)
        state = next_state

# Deployment
real_env.apply_policy(agent.policy)
```

Generative AI for RL

- This can also be done using GenAI[28].

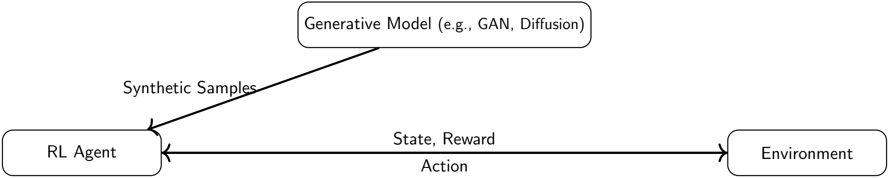


Figure: Generative models can help expand training data.

[28] A. T. Z. Kasgari, W. Saad, M. Mozaffari, *et al.*, "Experienced deep reinforcement learning with generative adversarial networks (gans) for model-free ultra reliable low latency communication," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 884–899, 2021. DOI: 10.1109/TCOMM.2020.3031930

Overview of RL Trustworthiness Dimensions

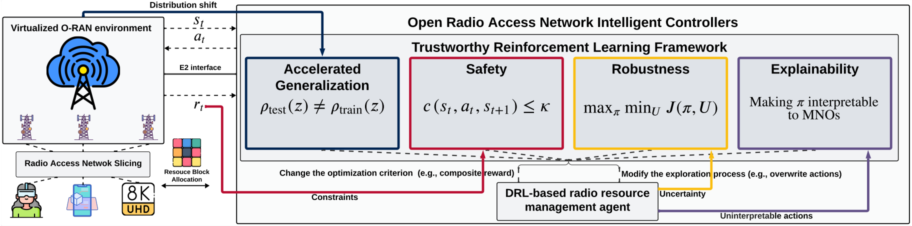
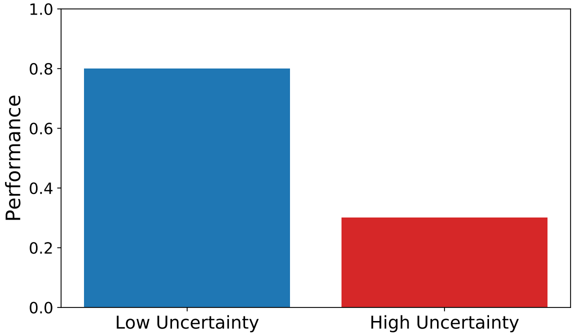


Figure: A trustworthy DRL framework for RRM in O-RANs[29]

29 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Developing trustworthy reinforcement learning applications for next-generation open radio access networks," in *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2024, pp. 137–138. DOI: 10.1109/CCECE59415.2024.10667311

Robust Reinforcement Learning

Non-Robust RL Algorithms



- Environment discrepancies and network stochasticity lead to uncertainties.
- **Need**: Enhance worst-case performance under uncertain network conditions.

Trustworthiness Dimension: Robustness

- **Definition**: Robustness is the ability of an RL agent to maintain performance under uncertainties and adversarial conditions[30].
- **Mathematical Formulation**:

$$\pi^* = \arg\max_{\pi} \min_{P \in \mathcal{P}} \mathbb{E}_{\pi,P}\left[\sum_t \gamma^t R(s_t, a_t)\right] \tag{15}$$

  where $\mathcal{P}$ is a set of plausible transition models.
- **Challenges**:
  - **Adversarial Attacks**: Deliberate perturbations.
  - **Model Uncertainties**: Inaccurate or incomplete $P(s'|s,a)$.
  - **Noise and Disturbances**: Random environmental fluctuations.
- **Importance**: Ensures consistent performance in non-ideal conditions.

30 M. Xu, Z. Liu, P. Huang, *et al.*, "Trustworthy reinforcement learning against intrinsic vulnerabilities: Robustness, safety, and generalizability," *arXiv preprint arXiv:2209.08025*, 2022

Approaches to Enhance Robustness: Adversarial Training

- **Concept**:
  - Training the agent to be resilient against adversarial inputs [31].
- **Method**:
  - Introduce adversarial perturbations during training:

$$s'_t = s_t + \delta_t \tag{16}$$

    where δ_t is crafted to maximize the agent's loss.
- **Benefit**:
  - Improves the agent's ability to handle unexpected disturbances.

[31] L. Pinto, J. Davidson, R. Sukthankar, *et al.*, "Robust adversarial reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2017, pp. 2817–2826

Example of Adversarial RL for Robust Beam-Tracking[32]

- **Core idea**: Treat differences between training and testing scenarios as disturbances introduced by an adversarial agent.
- By jointly training a protagonist (beam-tracking agent) and an adversarial agent, the protagonist experiences severe, realistic disturbances.
- **Result**: The protagonist becomes robust to various discrepancies between training and testing scenarios.

32 M. Shinzaki, Y. Koda, K. Yamamoto, *et al.*, "Zero-shot adaptation for mmwave beam-tracking on overhead messenger wires through robust adversarial reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 1, pp. 232–245, 2022. DOI: 10.1109/TCCN.2021.3116231

Example of Adversarial RL for Robust Beam-Tracking

- The adversarial agent applies changes, causing faster and more unpredictable beam misalignment.
- This emulates a challenging environment where beam directions are harder to correct.
- The protagonist learns a robust policy that effectively counters these amplified disturbances.
- Thus, after training with adversarial disturbances, the beam-tracking agent can adapt zero-shot to new conditions during testing.

RARL-Based Beam-Tracking Training Procedure

- Protagonist (beam-tracking agent):
  - Observes state (e.g., beam alignment, wind disturbances).
  - Selects actions to maximize received signal power.
  - Updates neural network from transitions.
- Adversary:
  - Observes same state.
  - Applies additional wind force to minimize received signal power.
  - Also updates its neural network.

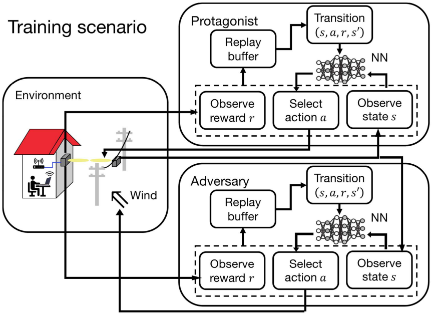


Figure: Training scenario of RARL-based beam-tracking.

Testing with Zero-Shot Adaptation

- In testing:
  - Only the protagonist is active.
  - Environmental parameters (e.g., wire mass, spring constant) differ from training values.
  - The protagonist applies its learned policy to counter beam misalignment in these new conditions without additional tuning.
- Goal: Demonstrate that training with adversarial disturbances yields a robust beam-tracking policy that generalizes to new scenarios.

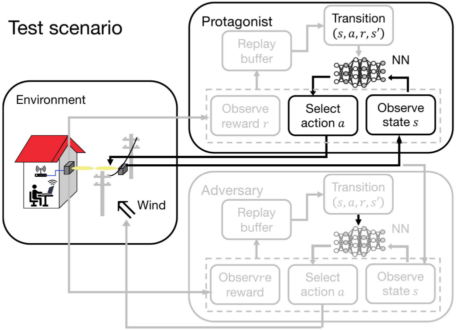


Figure: Testing scenario of zero-shot adaptation.

Overview of RL Trustworthiness Dimensions

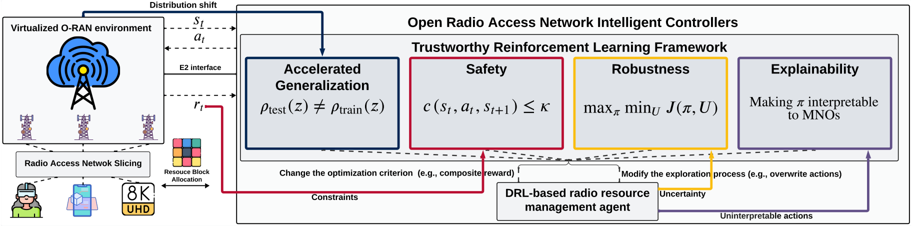


Figure: A trustworthy DRL framework for RRM in O-RANs[33]

33 A. M. Nagib, H. Abou-zeid, and H. S. Hassanein, "Developing trustworthy reinforcement learning applications for next-generation open radio access networks," in *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2024, pp. 137–138. DOI: 10.1109/CCECE59415.2024.10667311

Explainable Reinforcement Learning

Trustworthiness Dimension: Explainability

- **Transparent Decision-Making**: The rationale behind RL decisions should be explainable to stakeholders, ensuring trust in automated decisions.
- **Traceability**: It should be possible to trace decisions back to specific policies or learning processes.
- Deep neural networks act as "black boxes."
- **Implication**: Challenges in trust, accountability, and adoption.

33 A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez, "Explainability in deep reinforcement learning," *Knowledge-Based Systems*, vol. 214, p. 106 685, 2021

Approaches to Achieve Explainability: SHAP

What is SHAP?

- SHAP stands for **SHapley Additive exPlanations**.
- A popular framework for interpreting the output of machine learning models.
- Based on cooperative game theory concepts, specifically the **Shapley value**.

Key Idea:

- Assign a contribution value to each feature, indicating its influence on the model's prediction.
- Provides a consistent and theoretically sound explanation by fairly distributing the prediction among the input features.

33 S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 4765–4774

Approaches to Achieve Explainability: SHAP

Formal Definition:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[\pi(a|S \cup \{s_i\}) - \pi(a|S)\right], \tag{17}$$

where:

- ϕ_i: Contribution of state feature s_i to the decision to take action a.
- S: A subset of state features excluding s_i.
- $\pi(a|S)$: Policy output (probability of action a) with features in subset S.
- Shapley weighting ensures fair distribution of contributions across all subsets.

33 S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 4765–4774

Approaches to Achieve Explainability: SHAP

Explanation:

- SHAP quantifies how much each feature s_i influences the agent's decision.
- It considers the marginal impact of s_i across all subsets of features.

Context in XRL:

- Explains RL policies by identifying the most influential state features.
- Enhances interpretability, debugging, and trustworthiness of RL systems.

Approaches to Achieve Explainability: Example

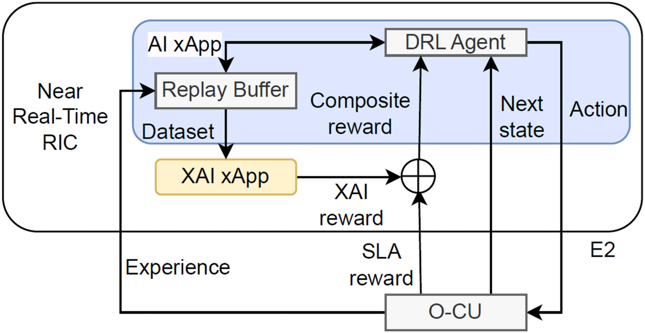


Figure: Deployment of explanation-guided deep reinforcement learning in O-RAN[34].

34 F. Rezazadeh, H. Chergui, L. Alonso, *et al.*, "Sliceops: Explainable mlops for streamlined automation-native 6g networks," *IEEE Wireless Communications*, vol. 31, no. 5, pp. 224–230, 2024. DOI: 10.1109/MWC.007.2300144

Approaches to Achieve Explainability: Example

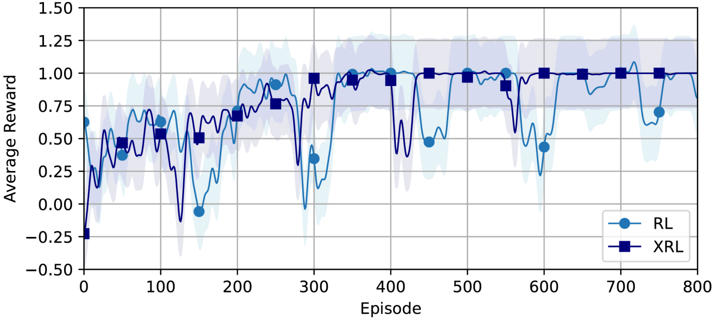


Figure: Explanation guided DRL maximize the decision confidence compared to traditional DRL[35].

35 F. Rezazadeh, H. Chergui, L. Alonso, *et al.*, "Sliceops: Explainable mlops for streamlined automation-native 6g networks," *IEEE Wireless Communications*, vol. 31, no. 5, pp. 224–230, 2024. DOI: 10.1109/MWC.007.2300144

Open Research Challenges and Future Directions

Future Perspectives toward Trustworthy RL for 6G

- Developing wireless **benchmark challenges** are essential to foster reproducible research that builds on the collective progress of the wireless research community.
  - Foster a culture where **limitations of AI** are encouraged and reported as challenges for others to pursue.
- **Industrial** collaboration to better understand and model the challenges of trustworthiness.
- **Foundational** "Generalist" 6G DRL agents that are trustworthy.
  - Can we build **generalizable** and **explainable** DRL agents for wireless?
  - Incorporating safety and robustness as well.
- Continual and **life-long learning** that is sample efficient.
- Real-time **scalability**, efficiency, and low-latency.

RL Resources

RL Resources: Concepts

- **Reinforcement Learning: An Introduction**
  `incompleteideas.net/book/RLbook2020.pdf`
- **RL Theory Seminars:**
  `sites.google.com/view/rltheoryseminars`
- **Safe Reinforcement Learning Online Seminars:**
  `sites.google.com/view/saferl-seminar`
- **Mila Tea Talks:**
  `sites.google.com/lisa.iro.umontreal.ca/tea-talks`
- **Reinforcement Learning Specialization on Coursera**
  `coursera.org/specializations/reinforcement-learning`
- **Reinforcement Learning Mailing List:**
  `groups.google.com/g/rl-list`

RL Resources: Concepts Specific to Wireless Networks

- **Single and Multi-Agent Deep Reinforcement Learning for AI-Enabled Wireless Networks: A Tutorial:** `ieeexplore.ieee.org/document/9372298`
- **Ericsson Blog Series on RL:** `ericsson.com/en/blog/2023/11/reinforcement-learning`

RL Resources: Tools

- **Denny Britz's RL Repository:**
  github.com/dennybritz/reinforcement-learning
- **MinimalRL-PyTorch:**
  github.com/seungeunrho/minimalRL
- **Tools for RL in Python**
  https://neptune.ai/blog/the-best-tools-for-reinforcement-learning-in-python

Generalizable RL Resources

- **Quantifying Generalization in RL:** `github.com/openai/coinrun`

Safe RL Resources

- **Safe RL Baselines:**
  `github.com/chauncygu/Safe-Reinforcement-Learning-Baselines`
- **Safe Policy Optimization (SafePO):**
  `github.com/PKU-Alignment/Safe-Policy-Optimization`
- **OmniSafe:**
  `github.com/PKU-Alignment/omnisafe`
- **Safety Gymnasium:**
  `github.com/PKU-Alignment/safety-gymnasium`
- **Safe Reinforcement Learning from Human Feedback (RLHF):**
  - **Beaver:** `github.com/PKU-Alignment/safe-rlhf`

Safe RL Resources

- **Safe Control Gym:** github.com/utiasDSL/safe-control-gym
- **Fast Safe RL (FSRL):** github.com/liuzuxin/FSRL
- **Offline Safe RL (OSRL):** github.com/liuzuxin/OSRL
- **Safety Gym and Starter Agents (Archived):**
  - github.com/openai/safety-gym
  - github.com/openai/safety-starter-agents

Robust RL Resources

- **StateAdvDRL:** `github.com/chenhongge/StateAdvDRL`
  Robust RL against adversarial perturbations on state observations.

- **Adversarial Reinforcement Learning Reading List:**
  `github.com/EzgiKorkmaz/adversarial-reinforcement-learning`

Explainable RL Resources

- **Awesome Explainable RL:**
  `github.com/Plankson/awesome-explainable-reinforcement-learning`

- **SHAP (SHapley Additive exPlanations):**
  `github.com/shap/shap`

RL for Next-Generation Wireless Networks

- **Available Resources**
  - List of RL-based wireless environments.
- **Planned Resources:**
  - Trustworthy RL algorithm implementations in NGWN literature.

Follow My GitHub

`github.com/ahmadnagib`

Conclusion

Final Thoughts

- **Adaptive Trustworthy RL Algorithms Needed**
  - Traditional RL struggles with dynamic and heterogeneous O-RAN environments.
  - Tailored RL approaches are essential for next-generation networks.

- **Trade-offs in Trustworthy RL**
  - Balancing safety, explainability, and performance is challenging.
  - Careful design is required to meet competing objectives.

Collaboration Opportunities

- We encourage community involvement in building Trustworthy RL methods for next-generation wireless networks.

- Reach out to us to explore opportunities for collaborative research and development.

Q&A and Acknowledgments

Thank you for your attention, please contact me if you have any questions!

Ahmad Nagib