Distributional-Utility Actor-Critic for Network Slice Performance Guarantee

Jingdi Chen
George Washington University
Washington, DC, USA
jingdic@gwu.edu

Tian Lan
George Washington University
Washington, DC, USA
tlan@gwu.edu

Nakjung Choi
Nokia & Bell Labs
Murray Hill, USA
nakjung.choi@nokia-bell-labs.com

ABSTRACT

Optimizing distributional utilities (such as mitigating performance tails and maximizing risk-aware objectives) is crucial for online network slice management to meet the diverse requirements of different services and applications. While Reinforcement Learning (RL) has been successfully applied to autonomous online decision-making in many network slice management problems, existing solutions often focus on maximizing the expected cumulative reward or are limited to specific distributional utilities. This paper proposes a new RL algorithm for general Distributional Utilities Optimization (DUO) in an actor-critic framework for online network slice management. In particular, we derive a DUO Temporal Difference Learning algorithm for updating distributional utilities in the critic through stochastic gradient descent. It is proven that the Distributional Optimal Bellman Operator for distributional utilities is a γ-contraction and thus is guaranteed to converge. In addition, we parameterize the policy by another neural network and prove a revised policy gradient theorem for distributional utilities, which shows that the derived policy update converges to at least a stationary point of the DUO problem. Our proposed algorithm works with arbitrary smooth utility functions on the return distributions, making it suitable for optimizing various network slice performance objectives in an online setting. Our solution is implemented and validated by building a hybrid trace-driven network simulator, which was built using an open-source O-RAN dataset, along with data collected from a 5G O-RAN testbed. Results demonstrate a significant improvement over heuristic and RL baselines.

CCS CONCEPTS

• **Networks** → **Network resources allocation**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Distributional Reinforcement Learning, Network Slicing.

ACM Reference Format:
Jingdi Chen, Tian Lan, and Nakjung Choi. 2023. Distributional-Utility Actor-Critic for Network Slice Performance Guarantee. In *International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '23), October 23–26, 2023, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3565287.3610255

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MobiHoc '23, October 23–26, 2023, Washington, DC, USA

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-9926-5/23/10…$15.00
https://doi.org/10.1145/3565287.3610255

1 INTRODUCTION

The growth of mobile services in the Internet of Things (IoT) and smart devices has presented challenges for 5G network management [11]. Network slicing has emerged as a key technology to address the diverse performance requirements of different services and applications [15]. Jointly managing network slices in a 5G network requires solving an online optimization problem over various control knobs such as scheduling policies, resource allocation, and spectrum actions [18] in dynamic wireless environments.

Intelligent management of wireless network slices is becoming a must. Recently, O-RAN Alliance[1] is defining open RAN interfaces and infrastructure to make more programmable RANs cross operators and vendors. In particular, the O-RAN framework can provide rich third-party support for network assurance and control through near-real-time (nRT) Radio Intelligent Controller (RIC) by using an E2 interface and Service Model (E2SM), e.g., E2SM-RC (Radio Control)[2]. However, data-driven methods for optimizing various distributional performance objectives, e.g., 95-percentile tail latency and throughput outage probabilities, are still limited. For example, a special URLLC slice for robot control may require performance guarantees defined on the distributions of throughput and latency during the whole time interval – rather than the standard on-average guarantee – which is our focus in this paper.

To this end, Reinforcement Learning (RL) has gained attention for intelligent wireless network optimization, as it enables agents to learn an optimal policy through interaction with the environment [14]. These works formulate online network slice management as a Markov Decision Process for utility maximization and apply RL techniques to learn an optimal policy. However, they often focus on optimizing expected performance objectives modeled as discounted cumulative rewards, rather than distributional utilities.

In this paper, we consider the learning problem of optimizing distributional utilities defined w.r.t. return/reward distributions and develop a new RL algorithm for general Distributional Utilities Optimization (DUO) in an actor-critic framework. In practice, there are many scenarios where the optimization must go beyond considering the expected return values and optimize the distribution of certain performance metrics/rewards. These include robust network optimization that considers outage probabilities [26], managing slice SLAs defined through availability and performance tails [15], and resource allocation taking risk into account in decision making such as [10, 22]. We model relevant network performance metrics as random rewards in RL and recast the problem of online network slice management by optimizing certain utility functions over the

distribution of such random returns. This approach allows us to meet different design objectives (e.g, balancing x-percentile latency for different network slices or minimizing the probability of each slice's throughput falling below y_i) by maximizing appropriate utility function on the return distribution. In contrast to existing work on distributional RL that is often limited to specific types of utility functions such as the expected value [8, 27] and the Conditional Value at Risk [22], our goal in this paper is to build an RL framework with respect to general distributional utilities.

More precisely, we optimize arbitrary smooth, differentiable utility functions on return distributions using an actor-critic framework – Deep Neural Networks (DNNs) estimate return/reward distributions (for each slice) from dynamic network state observations, such as network conditions and available slice/user statistics. The Temporal Difference Learning algorithm updates target distributional utilities in the critic through stochastic gradient descent, with proof of convergence by the Distributional Optimal Bellman Operator's γ-contraction property. We parameterize the policy (for generating slice management actions) with another neural network and derive a revised policy gradient theorem for distributional utilities, which shows convergence to at least a stationary point for optimization. To the best of our knowledge, this is the first proposal to optimize general distributional utility functions using actor-critic RL.

The proposed algorithm allows us to readily solve many network slice management problems involving distributional utilities. To demonstrate this, we consider the problem of jointly apportioning network resources among multiple slices (in terms of physical resource blocks) and optimizing intra-slice scheduling policies (i.e., selecting proportional fair, water-filling, or round-robin schedulers for each network slice), with the goal of mitigating performance tails (e.g., the probability of received data rates falling below prescribed SLAs). The problem can be easily formulated as RL with heterogeneous distributional utilities (due to varying SLAs for different slices) and solved efficiently using the proposed algorithms.

We implement and validate our solution by building a hybrid trace-driven network simulator. It trains a DNN to generate the reward trajectories given network slice management actions in each episode (i.e., resource allocation and intra-slice scheduling policy) by using data from both an open-source Colosseum O-RAN COMMAG Dataset [7] and in OTA (Over-The-Air) 5G O-RAN testbed. The dataset contains time-varying network state information and slice/UE statistics for multiple cells and under different network conditions. Modeling each slice as an individual learning agent, we implement our actor-critic framework for optimizing distributional utilities. It jointly manages multiple slices with highly-variant performance expectations. We show that our DUO algorithm can achieve the optimal policy and significantly outperform both heuristic policies and baseline RL algorithms, including D4PG [3] and RMIX [22], in particular, by reducing performance tail probabilities by up to one order of magnitude using a log-percentile utility.

The main contributions of this paper are as follows:

- We propose an RL algorithm for optimizing general distributional utilities in an actor-critic framework. The $N-$step distributional bellman operator is shown to be a $\gamma-$contraction;
- We leverage an $N-$step Distributional TD learning algorithm for a critic update. A policy gradient theorem is proven to guarantee convergence to at least a stationary point of the network utility optimization;
- The results work with arbitrary smooth utilities and enables Distributional Multi-Agent Actor-Critic algorithms for online network slice management;
- Our solutions are implemented and validated by building a hybrid trace-driven network simulator using an open-source ORAN dataset and collecting additional datasets with real UEs in a shield room in our OTA (Over-The-Air) 5G O-RAN testbed. Significant improvement is achieved over heuristic and RL baselines.

2 RELATED WORK

Network utility maximization (NUM) is often solved through optimization methods such as gradient descent, but obtaining the complicated mathematical expressions of users' utility functions in real-world network circumstances can be challenging [24]. Deep learning and deep reinforcement learning have been applied to the problem, but they do not directly model the required distributions for network utility optimization, limiting their ability to guarantee specific network slice requirements beyond expected performance/rewards [14] Distributional RL has made significant advancements in various single-agent domains [4, 8, 30], but a multi-agent RL (MARL) system [20] is required for network optimization problems with large action spaces. Value-based distributional MARL [19, 31] faces the challenge of integrating individual distributional Q-values into a global distributional Q-value under the Distributional-Individual-Global-Max (DIGM) principle [27]. Limited research has addressed distributional MARL. RMIX [22] models individual Q-values with distributions, while DMIX [27] parameterizes both individual and global Q-values with a quantile function. However, neither approach considers general distributional network utilities.

3 PROBLEM FORMULATION

We consider a base station (BS) with K network slices in a mobile network. Multiple network slices are created to serve their associated users (UEs). The slices are modeled as the learning agents managing the resource allocation among different network slices and deciding relevant control actions/configurations, such as intra-slice scheduling policies, intending to maximize the overall utilities of all slices. Each network slice k serves UEs in a distinctive service class with unique service-level agreements (SLA) requirements. Let a_k^t denote the set of network slice management decisions such as the choice of intra-slice scheduling policies or the number of physical resource blocks (PRBs) assigned to slice k at each training step t. Given a_k^t, the resulting performance received by all UEs in network slice k is denoted by a random reward r_k^t (e.g., delay, data rate, or reliability). Thus, we consider the random return $Z_k^t = \sum_t \gamma^t r_k^t$ as the sum of discounted reward (with a factor γ) along network slice k's trajectory (i.e., the sequence of network conditions, scheduling decisions and reward signals). We define the distributional utility of network slice k at time slot t as a function of the random return distribution, i.e., $\mathbb{P}_{U_k^t} = \mathbb{P}_{U_k}(Z_k^t)$, where $\mathbb{P}_{U_k}(\cdot)$ is smooth utility function over the distribution of Z_k^t.

Such distributional utility functions capture a wide range of network design objectives. For instance, it could be (i) the 95-percentile tail of the random return, i.e., $\mathbb{P}_U(Z) = \{x : P(Z > x) \geq 0.95\}$, (ii) the Conditional Value at Risk (CVaR) function $\mathbb{P}_U(Z) = \frac{1}{1-c}\int_{-1}^{\alpha} zf(z)dz, \alpha > 0$ [23] over the distribution $f(z)$, as a risk assessment measure used in portfolio optimization for effective risk management, (iii) the prospect utility function $\mathbb{P}_U(Z) = \int_z zf(z)w(z)dz$ with S-shaped weights $w(z)$ to model risk-seeking and risk-aversion behaviors in decision making [10], or (iv) the α-fair utility functions $\mathbb{P}_U(z) = E[Z^{1-\alpha}/(1-\alpha)]$ for $\alpha \neq 1, \alpha > 0$, $\mathbb{P}_U(z) = E[\log(z)]$ for $\alpha = 1$, which are widely used in network optimization [17].

The objective of network slice management is to maximize the sum of distributional utility of all network slices, which can be expressed as $\sum_k \mathbb{P}_{U_k}(\lim_{T\to\infty} 1/T \sum_{t=0}^{T} \gamma^t r_k^t)$. We consider this stochastic programming problem for network slice management with a finite T time horizon [24] and define the *Distributional Utility Optimization (DUO)* problem:

$$\max \quad \sum_k \mathbb{P}_{U_k}(Z_k^t), \quad \text{s.t. } Z_k = \sum_{t=0}^{T} \gamma^t r_k^t, \tag{1}$$

where distributional utilities $\mathbb{P}_{U_k}(\cdot)$ model different performance expectations as defined by SLA requirements over Z_k distribution. Due to dynamic network conditions, this network slice management problem requires online decision-making. RL is a natural way to solve this. However, RL algorithms typically aim to maximize the expected discounted long-term reward of all agents - i.e., $\mathbb{E}_\pi[\sum_t \sum_k \gamma^t r_k^t]$ - and fail to consider distributional utilities that are defined over the distributions over random return Z_k and also could vary among different network slices. Due to this difference, distributional utilities cannot be expressed as the accumulation of rewards. Even if the reward function is appropriately shaped to incorporate the distributional properties of the return, the resulting expected reward would still be an incomplete representation of the return distribution. The key to developing a solution is to estimate the distribution of the random return and devise a policy update for maximizing such distributional utility objectives.

4 LEARNING ALGORITHM AND ANALYSIS

4.1 Solution Overview

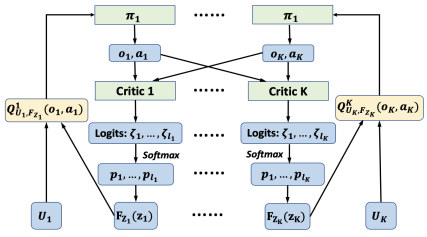


Figure 1: Our framework

In this section, we address the challenges outlined in the previous section 3. We reformulate the problem using a distributional RL approach, which involves learning return distributions of individual agents and defining distributional value functions calculated analytically based on the return distributions as well as each target utility. The defined distributional value functions are incorporated into the TD error objective and act as auxiliary local rewards during centralized training for updating policies. Return distributions are updated using cross-entropy loss.

Our design, illustrated in Figure 1, adopts the Centralized Training Decentralized Execution (CTDE) framework commonly employed in Multi-Agent Reinforcement Learning (MARL). Unlike CTDE, our approach does not utilize deterministic centralized critic values for each agent. Instead, the critic outputs logits, which are converted into probability density functions using the softmax function. These distributions form the basis for the random return Z_k. We introduce a distributional value function $Q_{U_k,F_{Z_k}}(s,a)$ that is dependent on the arbitrary target utility function $\mathbb{P}_{U_k}$ and the return distribution Z_k, represented by the Cumulative Distribution Function (CDF) F_{Z_k}, based on the critic outputs. Lemma 1 shows the convergence of such distributional value function. We present a novel policy gradient theorem, as proven in Theorem 1, that extends the standard policy gradient theorem by incorporating the randomness of the return distribution and an arbitrary utility function. The standard policy gradient theorem only considers the expected return, whereas our proposed theorem introduces an additional level of complexity by incorporating an arbitrary utility function.

The proposed utility-based distributional value function and policy gradient theorem are employed to approximate distributional value functions through discretization, allowing for computation with a DNN. This approach is extended to address the DUO problem in multi-agent scenarios for network slicing problems, providing a scalable and flexible solution. The discretization and DNN-based computation of distributional value functions can be applied to multiple agents, facilitating independent multi-agent learning based on specific requirements for each slice. This approach is particularly useful for solving large-scale problems with multiple agents and allows for efficient policy updates for each agent, ultimately leading to optimal resource allocation for each slice.

4.2 Convergence of the Distributional Value Function

We optimize network utility by modeling the network as an environment and the network slices as learning agents. Agents learn policies by interacting with the environment to optimize arbitrary utilities $\mathbb{P}_{U_k}(Z_k)$ of random return. This is considered as an infinite horizon discounted Markov Decision Process (MDP) $\mathcal{M}$ that is defined by the tuple (S, A, P, K, R, γ). Here S denotes a set of states, A a set of actions, and $P : S \times A \to S$ the transition probability distribution. K is the total number of agents. And R is the reward function, which in this work we explicitly treat it as a random variable.

We use a stochastic policy $\pi(a|s)$ to map each state $s \in S$ to a probability distribution over the action space A. Distributional RL aims to approximate the distribution of returns (i.e., the sum of discounted rewards along the agent's trajectory of interactions with the environment) denoted by a random variable $Z_\pi(s,a) = \sum_t \gamma^t R_t$, whose expectation is the scalar state-action value function $Q_\pi(s,a)$, i.e., $Q_\pi(s,a) = E[Z_\pi(s,a)] = E[\sum_t \gamma^t R_t]$. Similar to the Bellman equation of Q-value function $Q_\pi(s,a) \overset{D}{:=} E[R(s,a)] + \gamma E_{P,\pi} Q'_\pi(s',a')$, $Z_\pi(s,a)$ is also described by a recursive

equation of a distributional nature:

$$Z_\pi(s,a) \overset{D}{:=} R(s,a) + \gamma Z_\pi(s',a'), \tag{2}$$

where $s' \sim P(\cdot|s,a)$, $a' \sim \pi(\cdot|s')$. Then we have the distributional Bellman optimality operator $\mathcal{T}$ as follows:

$$(\mathcal{T}_\pi Z)(s,a) \overset{D}{:=} R(s,a) + \gamma E[Z(s',\pi(s'))|s,a], \tag{3}$$

based on the distributional Bellman optimality operator above, the objective of traditional distributional RL is to reduce the distance between the distribution $Z(s,a)$ and the target distribution $\mathcal{T}^q Z(s,a)$.

In order to solve the DUO problem in section 3, we are typically interested in acting so as to maximize the target utilities w.r.t. the return distributions. Therefore, we define a distributional value function, $Q_{U_k,F_{Z_k}}(s,a)$, in terms of the arbitrary target utility function, $\mathbb{P}_{U_k}$, and the CDF of the random return F_{Z_k}, which guides each agent's decision-making process via its utilization as the actor loss.

Definition 1. ***(Distributional value function.)*** *The distributional value function is formally defined as Q_{U,F_Z}, where U denotes the utility function, and F_Z represents the Cumulative Distribution Function (CDF) of the random return variable Z.*

To maximize the distributional value function of each agent k (we omit k and treat each agent k identical in the rest of our paper for notation brevity), we define a Bellman operator $\mathcal{T}^q$ for the distributional value function as:

$$\mathcal{T}^q Q_{U,F_Z}(s,a) \overset{D}{:=} E[R(s,a)] + \gamma E_P\left[max_{a'\in A} Q_{U,F_Z}(s',a')\right]. \tag{4}$$

where we view distributional value functions as vectors in $\mathbb{R}^{S\times A}$ and the expected reward function as on such vector. This operator describes the expected behavior of the agents. In particular, this is a contraction mapping with repeated application to some initial $Q^0_{U,F_Z}(s,a)$ converges exponentially to a unique fixed point $Q^*_{U,F_Z}(s,a)$ which is the optimal distributional value function corresponding to the set of optimal policies π^*, where π^* satisfies the equation $E_{a\sim\pi^*} Q^*_{U,F_Z}(s,a) = max_a Q^*_{U,F_Z}(s,a)$ [5]. This operator $\mathcal{T}^q$ is fundamentally different from the usual Bellman operator $\mathcal{T}Q(s,a) = E[R] + \gamma E_P[max_{a'\in A} Q(s',a')]$ for traditional state-action value function $Q(s,a)$. In particular, the randomness in the reward R, utility function distribution $\mathbb{P}_U$ and next-state return distribution $Z(s',a')$ define the compound distribution $\mathcal{T}^q Q_{U,F_Z}(s,a)$. In particular, we make the usual assumption that these quantities are independent. In this section, we will show that Eq. 4 is a contraction mapping whose unique fixed point is the random distributional value function $Q_{U,F_Z}(s,a)$.

Consider the process $Q^{(k+1)}_{U,F_Z} = \mathcal{T}^q Q^{(k)}_{U,F_Z}$ starting from initial value $Q^0_{U,F_Z} \in \mathcal{Q}$, we now show that this process converges in a strong sense that $\mathcal{T}^q$ is a contraction, which implies that all moments also converge exponentially quickly to a fixed point $Q^*_{U,F_Z}(s,a)$.

Lemma 1. *$\mathcal{T}^q: \mathcal{Q} \mapsto \mathcal{Q}$ is a γ-contraction.*

Proof. We show that the sup-norm contraction satisfies:

$$\begin{aligned} &|\,\mathcal{T}^q Q^{(1)}_{U,F_Z}(s,a) - \mathcal{T}^q Q^{(2)}_{U,F_Z}(s,a)\,| \\ &\le \gamma ||\mathcal{T}^q Q^{(1)}_{U,F_Z}(s,a) - \mathcal{T}^q Q^{(2)}_{U,F_Z}(s,a)||_\infty, \end{aligned} \tag{5}$$

for all $s \in S$, $a \in A$. The sup-norm is defined as $||Q_{U,F_Z}(s,a)||_\infty = sup_{s\in S, a\in A}|Q_{U,F_Z}(s,a)|$. For give distributional value function $Q \in R$ with fixed hyper-parameters (e.g., corresponding to different SLA requirements), we show that for two different return distributions $Z_{(1)}$ and $Z_{(2)}$, we have:

$$\begin{aligned} &|\mathcal{T}^q Q^{(1)}_{U,F_Z}(s,a) - \mathcal{T}^q Q^{(2)}_{U,F_Z}(s,a)| \\ &\le max_{s,a}|[\mathcal{T}^q Q^{(1)}_{U,F_Z}](s,a) - [\mathcal{T}^q Q^{(2)}_{U,F_Z}](s,a)|, \\ &= max_{s,a}|\gamma \sum_{s'} P(s'|s,a)[max_{a'} Q^{(1)}_{U,F_Z}(s',a') - max_{a'} Q^{(2)}_{U,F_Z}(s',a')]|, \\ &\le \gamma max_{s'}|max_{a'}|Q^{(1)}_{U,F_Z}(s',a') - Q^{(2)}_{U,F_Z}(s',a')||, \\ &\le \gamma max_{s',a'}|Q^{(1)}_{U,F_Z}(s',a') - Q^{(2)}_{U,F_Z}(s',a')|, \\ &= \gamma ||Q^{(1)}_{U,F_Z}(s,a) - Q^{(2)}_{U,F_Z}(s,a)||_\infty, \end{aligned} \tag{6}$$

where the second step uses the state transition probabilities $P(s'|s,a)$ and the third step considers the maximum $|Q^{(1)}_{U,F_Z}(s,a) - Q^{(2)}_{U,F_Z}(s,a)|$. This further implies that:

$$\begin{aligned} &|\,\mathcal{T}^q Q^{(1)}_{U,F_Z}(s,a) - \mathcal{T}^q Q^{(2)}_{U,F_Z}(s,a)\,| \\ &\le \gamma ||\mathcal{T}^q Q^{(1)}_{U,F_Z}(s,a) - \mathcal{T}^q Q^{(2)}_{U,F_Z}(s,a)||_\infty,\ \forall s \in S, a \in A. \end{aligned} \tag{7}$$

where we use the upper bound of the left-hand-side of Eq. 6 in step 1 above, and use MDP state transition probabilities to move one step forward from (s,a) to (s',a') in step 2, use the upper bound of step 2 by maximizing over the whole state space in step 3, and extract the action outside of the absolute value we could prove the result above. □

With lemma 1, we use the Banach fixed-point theorem to derive that $\mathcal{T}^q$ has a unique fixed point assuming all moments are bounded, this is sufficient to conclude that the sequence $\{Q^k_{U,F_Z}\}$ converges to a stationary point $\{Q^*_{U,F_Z}\}$. Therefore, we can leverage the TD learning to compute the maximal distributional value functions of each agent thus leading to the maximal global distributional value functions. The estimation of the value of Q_{U,F_Z} can be achieved through either sampling or computation from the parameterized return (Z) distribution. However, the sampling method is typically computationally expensive [29]. Therefore, we let each agent k learn its return distribution parameterized by a mixture of Dirac Delta (δ) functions. This approach has been demonstrated to be highly expressive and computationally efficient [4]. The parameterized return distribution of each agent at time t is defined as:

$$Z^t(\tau^t, a^{t-1}) = \sum_{j=1}^{M} \mathscr{P}_j(\tau^t, a^{t-1})\delta_j(\tau^t, a^{t-1}), \tag{8}$$

where M represents the number of Dirac Delta functions. The j-th Dirac Delta function, denoted as $\delta_j(\tau^t, a^{t-1})$, specifies the estimated value that may be effectively parameterized by neural networks in practice. $\mathscr{P}_j(\tau^t, a^{t-1})$ denotes the corresponding probability of the estimated value given local observations and actions. The notation τ^t signifies the trajectories that span up to the t-th time step, whereas a^{t-1} represents the action of agent k.

The individual return distribution, $Z^t(\tau^t, a^{t-1}) \in \mathcal{Z}$, and its corresponding Cumulative Distribution Function (CDF), F_{Z^t}, are used to define a distributional value function operator Π^Q over

return in terms of the target utility function U: $\Pi^Q Z^t(\tau^t, a^{t-1}) := Q^t_{U,F_{Z^t}}(\tau^t, a^{t-1})$, where $Q \in \boldsymbol{Q}$. The distributional value function could be estimated in a non-parametric way given the ordering of Dirac Delta functions $\{\delta_j\}_{j=1}^m$ by leveraging its individual distribution as(we will omit t for notation brevity):

$$Q_{U,F_Z} = \sum_{j=1}^{m} \mathscr{P}_j \delta_j \mathbf{1}\{\delta_j \le \hat{v}_m\}, \tag{9}$$

where $\mathbf{1}\{\cdot\}$ denotes the indicator function, $\hat{v}_m$ represents the estimated value conditioned on the distributional value function operator Π^Q, and $\hat{v}_m = \lfloor \delta_m | \Pi^Q Z \rfloor$ with $\lfloor \cdot \rfloor$ being floor function. This is a closed-form formulation and can be easily implemented in practice. The optimal action for agent k can be calculated by computing $argmax_a Q_{U,F_Z}$, which will be elaborated on in detail in Section 4.3.

Using Lemma 1, we can leverage the distributional TD learning to compute the maximal distributional value function Q_{U,F_Z} of each agent by replacing the standard TD error with some metric d that measures the distance between two distributions. Here we utilize $N-$step returns when estimating the distributional TD error, which is widely used in the context of many policy gradient algorithms [21] and Q-learning variants [12]. This modification can be applied analogously to the distributional Bellman operator to make use of it when updating the distributional critic. By replacing the distributional Bellman operator with an $N-$ step variant, we have:

$$\begin{aligned} L(\omega) &= E_\rho \left[d(Y, Q^\omega_{U,F_Z}(s_0, a_0)) \right], \\ Y &= \sum_{n=0}^{N-1} \gamma^n r_n + \gamma^N E[Q^{\omega'}_{U,F_Z}(s_N, a_N) | (a_0, s_0)], \end{aligned} \tag{10}$$

where $d(Y, Q^\omega_{U,F_Z}(s_0, a_0)$ is distributional value function TD error for updating value functions w.r.t. the $N-$step transition dynamics. ω is the parameters of utility that can be modeled by a DNN, and ω' indicates the parameters of the target critic network that is periodically copied from ω for stabilizing training.

For local return (Z) distribution learning, we first consider utility values as dummy rewards of each agent due to its property of modeling the potential return loss and then leverage the cross-entropy loss used in Distributional RL to explicitly update the local distributions [3, 4]. More concretely, we model the return distribution using a discrete distribution parameterized by $N \in \mathcal{N}$ and $V_{min}, V_{max} \in \mathcal{R}$, and whose support is the set of the atoms $\{z_i = V_{min} + i\Delta z, 0 \le i < N\}$, $\Delta z = \frac{V_{max} - V_{min}}{N-1}$. In a sense, these atoms are the 'canonical returns' of Z distribution. The atom probabilities are given by a DNN model parametrized by ζ:

$$Z_\zeta(s,a) = z_i, \;\; w.p.\; p_i(s,a) = \frac{e^{\zeta_i(s,a)}}{\sum_j e^{\zeta_j(s,a)}} \propto exp\{\zeta_i\}, \tag{11}$$

observe that this distributional layer corresponds to a linear layer from the critic's torso to the logits $\zeta_i(s,a)$ followed by a softmax activation. Since this distribution is not closed under the bellman operator defined earlier and the fact that adding and scaling these values will no longer lie on the support defined by the atoms. This support is explicitly defined by the (V_{min}, V_{max}) hyperparameters. As a result, we instead use a projected version of the distributional Bellman operator [4] and let p' be the probabilities of the projected distributional Bellman operator $\Phi\mathcal{T}_\pi$ applied to the target distribution Z', While training, gradients from Z are blocked to avoid changing the weights of agents' utility function network. We write the cross-entropy loss for updating Z_ζ as:

$$d(\Phi\mathcal{T}_\pi Z', Z) = \sum_{i=0}^{l-1} p'_i \frac{e^{\zeta_i(s,\pi(s))}}{\sum_j e^{\zeta_j(s,\pi(s))}}. \tag{12}$$

4.3 Policy Gradient Theorem

Based on the analysis in section 4.2, our policy aims at selecting actions to maximize the distributional value function $Q_{U,F_Z}(s,a)$ instead of deterministic state-action values $Q(s,a)$. To update the policy, we derive it directly from a parameterized critic $Q^\omega_{U,F_{Z_\zeta}}(s,a)$ (denote as $Q^\omega_{U,\zeta}(s,a)$ for brevity). This is accomplished by considering a parameterized policy π_θ and maximizing its expected value through optimization of the following expression:

$$J(\theta) = E_{\pi_\theta}[Q^\omega_{U,\zeta}(s,a)|_{a=\pi_\theta(s)}]. \tag{13}$$

Unfortunately, the policy gradient theorem for RL with cumulative rewards [28] no longer holds when we consider a general function instead of cumulative rewards. Therefore, we derive a Distributional Policy Gradient Theorem in Theorem 1 with distributional value function distribution $Q^\omega_{U,\zeta}(s,a)$ which establishes that the parameterized policy gradient is the solution to optimize the arbitrary target utility functions $\mathbb{P}_U$ over the distribution of the random return Z_ζ.

Theorem 1. *The following policy gradient update:*

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[\nabla_a Q^\omega_{U,\zeta}(s,a) \left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(s_t) \right) |_{a_t=\pi_\theta(s_t)} \right], \tag{14}$$

converges to a stationary point of the distributional value function optimization.

Proof. We consider a standard Gradient Descent Method to update the policy parameter θ (and thus policy π_θ) to maximize objective $J(\theta) = E_{\pi_\theta}[Q^\omega_{U,\zeta}(s,a)|_{a=\pi_\theta(s)}]$ in Eq. 13, i.e.,

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J(\theta_t). \tag{15}$$

The policy parameter θ^* (and thus policy π_{θ^*}) converges to a stationary point for the expectation of the distributional value function $E_{\pi_\theta}[Q^\omega_{U,F_{Z_\zeta}}(s, \pi_\theta(s))]$. To this end, we need to obtain gradient $\nabla_\theta J(\theta)$ for sampled trajectories.

We begin with reformulating the gradient starting with the expansion of expectation (with a slight abuse of notation):

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta E_{\pi_\theta}\left[Q^\omega_{U,\zeta}(s,a)|_{a=\pi_\theta(s)} \right] \\ &= \nabla_\theta \int \pi_\theta(\tau) Q^\omega_{U,\zeta}(s,a)|_{a=\pi_\theta(s)} d\tau, \\ &= \int \nabla_\theta \pi_\theta(\tau) \nabla_a Q^\omega_{U,\zeta}(s,a)|_{a=\pi_\theta(s)} d\tau, \\ &= \int \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} \nabla_a Q^\omega_{U,\zeta}(s,a)|_{a=\pi_\theta(s)} d\tau, \\ &= \int \pi_\theta(\tau) \nabla_\theta \ln \pi_\theta(\tau) \nabla_a Q^\omega_{U,\zeta}(s,a)|_{a=\pi_\theta(s)} d\tau, \\ &= E_{\pi_\theta}\left[\nabla_a Q^\omega_{U,\zeta}(s,a)|_{a=\pi_\theta(s)} \nabla_\theta \ln \pi_\theta(\tau) \right], \end{aligned} \tag{16}$$

where τ is a given trajectory. Here we calculate the expectation using policy π_θ in step 1 and change the sequence of taking derivative and integral in step 2. By multiplying $\pi_\theta(\tau)/\pi_\theta(\tau)$ in step

3, we could use $\nabla_\theta \ln \pi_\theta(\tau)$ to represent the term $\nabla_\theta \pi_\theta(\tau)/\pi_\theta(\tau)$ in step 4 and get the expectation over policy π_θ in the final step. This means that the derivative of the expected utility for a given trajectory is the expectation of the product of the utility function and gradient of the ln of the policy π_θ, i.e.,

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[\nabla_a Q^{\omega}_{U,\zeta}(s,a)|_{a=\pi_\theta(s)} \nabla_\theta \ln \pi_\theta(\tau)\right]. \quad (17)$$

Now, we expand the definition of $\pi_\theta(\tau)$:

$$\pi_\theta(\tau) = P(s_0)\prod_{t=1}^{T} \pi_\theta(s_t) p(s_{t+1}, r_{t+1}|s_t, a_t). \quad (18)$$

where P represents the ergodic distribution of starting in state s_0. Then we apply the product rule of probability because each new action probability is independent of the previous one. At each step, we take action using the policy π_θ, and the environment dynamics p decide which new state to transition into. Those are multiplied over T time steps representing the length of the trajectory. Equivalently, taking the logarithm on both sides of Eq. 18, we have:

$$\begin{aligned}\ln \pi_\theta(\tau) &= \ln P(s_0) + \sum_{t=1}^{T} \ln \pi_\theta(s_t) + \sum_{t=1}^{T} \ln p(s_{t+1}, r_{t+1}|s_t, a_t),\\ &= \sum_{t=1}^{T} \ln \pi_\theta(s_t),\end{aligned} \quad (19)$$

Plugging this into the right-hand side of Eq. 17 to replace the term $\ln \pi_\theta(\tau)$ and considering the summation over T time steps first, we have:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta E_{\pi_\theta}\left[Q^{\omega}_{U,\zeta}(s,a)|_{a=\pi_\theta(s)}\right]\\ &= E_{\pi_\theta}\left[\nabla_a Q^{\omega}_{U,\zeta}(s,a)|_{a=\pi_\theta(s)} \left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(s_t)\right)\right].\end{aligned} \quad (20)$$

which is the desired result. □

The result shows that we could use a model-free policy gradient method instead of knowing the ergodic distribution of states P nor the environment dynamics p. The algorithm samples trajectories from a replay buffer of experiences stored throughout training to approximate the expectation E_{π_θ}. For updating the distributional value function $Q^{\omega}_{U,\zeta}(s,a)|_{a=\pi_\theta(s)}$, we will use critic loss in Eq. 10 discussed in section 4.2.

Lemma 1 and Theorem 1 inspire an actor-critic algorithm to solve the DUO problem defined in Eq 1. In particular, to compute the adjusted actor loss in Eq. 20, we can leverage an estimate of utility-based distributional value functions' distributions on the existing trajectory up to epoch t (instead of the expectation of the return under the current policy). Let $Q^{\omega}_{U,F_Z}(s_t,a_t)$ be a DNN approximate of the distributional value function parameterized by ω. We train $Q^{\omega}_{U,F_Z}(s_t,a_t)$ using the modified N-step distributional Temporal Different (TD) error and guide the actor network to choose action $argmax_{a_t}\{Q^{\omega}_{U,F_Z}(s_t,a_t)\}$. Note that the learning process is a sampling from some replay table of size R and performing the necessary network updates using this data. Additionally, sampling is implemented using non-uniform priorities p_i as in [25]. This requires importance sampling by weighting the critic update by a factor of $1/Rp_i$ [13]. Here the actor and critic parameters are updated using stochastic gradient descent with learning rates, α_t and β_t respectively, which are adjusted online using ADAM [16]. Our proposed Distributional Utility Actor-Critic (DUAC) algorithm is summarized in Algorithm 1.

Algorithm 1 DUAC Algorithm

Input: **max-episode**, **max-episode-length**, batch size M, trajectory length N, reply buffer size R, exploration constant ϵ, τ, learning rate α_0, β_0;
Initialize network weights (θ, ω) for each agent k at random;
Initialize target weights $(\theta', \omega') \leftarrow (\theta, \omega)$ for each agent k;
for episode= $1, 2, \ldots,$ **max-episode do**
  Start a new episode;
  Initialize a random process $\mathcal{N}$ for action exploration;
  for $t = 1$ to **max-episode-length do**
    for agent $k = 1$ to K **do**
      Receive initial state s;
      Select action $a = \pi_\theta + \mathcal{N}_t$;
      Execute actions a, observe reward r, new state s';
      Store (s, a, r, s') in replay buffer D;
      $s \leftarrow s'$;
      Estimate the local return distribution $Z(s,a)$;
      Sample M transitions of length N $(s_{i:i+N}, a_{i:i+N}, r_{i:i+N}, s'_{i:i+N})$ from D with priority p_i;
      Construct the target distributions:

$$Y_i = \Big(\sum_{n=0}^{N-1} \gamma^n r_{i+n}\Big) + \gamma^N Q^{\omega}_{U,F_Z}(s_{i+N}, a_{i+N})|_{a_{i+N}=\pi_{\theta'}(s_{i+N})}$$

      Compute the critic updates:

$$\delta_\omega = \frac{1}{M}\sum_i \nabla_\omega (Rp_i)^{-1} d(Y^i, Q^{\omega}_{U,F_Z}(s_i,a_i))$$

      Compute the actor updates:

$$\delta_\theta = \frac{1}{M}\sum_i \nabla_\theta \ln \pi_\theta(s_i)[\nabla_{a_i} Q^{\omega}_{U,F_Z}(s_i,a_i)|_{a_i=\pi_\theta(s_i)}]$$

      Update actor network parameters: $\theta \leftarrow \theta + \alpha_t \delta_\theta$;
      Update critic network parameters: $\omega \leftarrow \omega + \beta_t \delta_\omega$;
      Update the local return Z via loss in Eq. 12;
      Update actor target network parameters:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

      Update critic target network parameters:

$$\omega' \leftarrow \tau\omega + (1-\tau)\omega'$$

    end for
  end for
end for

5 EVALUATION

We implement and validate the proposed Distributional Utility Actor-Critic (DUAC) algorithm by building a hybrid trace-driven network simulator. It is able to leverage realistic network traces either collected from our real-world 5G testbed or availble from public datasets such as the open-source ORAN dataset [7]. Using the network traces as input, our simulator provides an RL enviroment by replaying time-varying network conditions and generating rewards/outcomes for different network slice management actions. We show that the DUAC algorithm can significantly outperform many heuristic and learning algorithms.

5.1 Applying DUAC Algorithm to Network Slice Management

In this part, we apply the proposed DUAC algorithm to solve the DUO problem in network slice management formulated in section 3. We model K network slices as K agents. The network conditions will be modeled as the state information s_k of each agent k. The action a_k of each agent contains two parts: intra-slice scheduling policy H_k and number of PRBs B_k assigned to the agent k. We use a hybrid trace-driven model, where the state transition of each agent follows the trace execution. We use our simulator developed in section 5.3 to re-play the network record traces (take a_k and s_k as input), predict the reward (i.e., total data rate) r_k and give back to each agent k for training.

Our DUAC resource scheduler assigns the scheduling policy H_k and number of PRBs B_k to agent k with a constraint of the total number of PRBs after receiving the initial states. The scheduling policy H_k can be chosen from three types of policies: policy 0 – proportionally fair (PF), policy 1 – water-filling (WF), and policy 2 – round-robin (RR) [6]. The action vector $a_k = (H_k, B_k)$ will be modeled as an one-hot vector with one entry set at 1 indicating a combination of H_k and B_k, where $H_k \in \{0, 1, 2\}$ and $B_k \in \{2, 4, 6, 8\}$(four possible number of PRBs in ORAN dataset). Then a softmax function applies to B_k and gets a corresponding weight W_k to agent k. We use the weight W_k multiple the total number of PRBs to get the exact number of PRBs for agent k and pass this information along with H_k to our simulator in section 5.3 in order to get the reward signal r_k. The DUO problem in network slice management can be formally defined as:

$$\max \sum_k \mathbb{P}_{U_k}(Z_{\zeta_k}(s, \pi_\theta(s))), \text{ s.t. } Z_{\zeta_k} = \sum_{t=0}^{T} \gamma^t r_k^t, \tag{21}$$

where r_k^t is the data rate in network slice k at time step t, and γ is a constant discount factor, ζ_k, π_k are neural network parameters estimating return distributions and policy described in section 4.

5.2 Training Data Collection

An open-source Colosseum O-RAN COMMAG Dataset which contains time-varying network state information and slice/UE statistics for multiple cells under different network conditions is provided in [7]. The datasets emulated a 5G network with 4 BSs and 40 UEs in the dense urban scenario of Rome, Italy. It provides datasets for a multi-slice scenario in which UEs are statically assigned to a slice of the network and request three different traffic types: eMBB (1 Mb/s constant bit rate traffic), URLLC (Poisson traffic, with 10 pkt/s of 125 bytes), and MTC (Poisson traffic, with 30 pkt/s of 125 bytes). The BSs serve each slice with a dedicated and different scheduling policy, selecting among proportionally fair (PF), water-filling (WF), and round-robin (RR) [6] and the number of PRBs allocated to each slice varies over time [9].

We also conducted similar experiments in our OTA (Over-The-Air) 5G O-RAN testbed and collected additional datasets with real UEs in a shield room. As shown in Figure 2, this experiment setup has open5gs for 5G core, B210/X310-based gNBs and 9 COTS UEs. Each gNB is configured with 40 MHz of n78 (C-band 3.5 GHz) and 3 slices (i.e., eMBB, URLLC, MTC) each of which has 3 UEs. However, application traffic scaled up to make some congestion

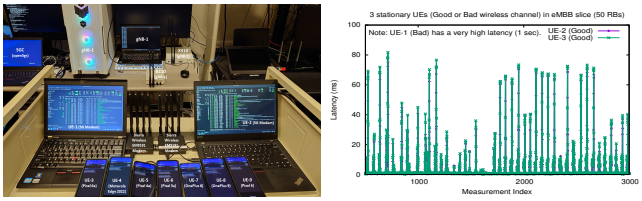


Figure 2: Data Collection System and Example of Measurement.

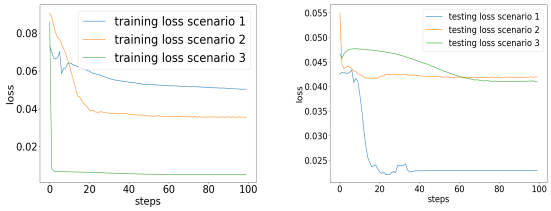
in a high-capacity 5G network. Our datasets show performance dynamics even with stationary UEs in a single eMBB slice. In this example, UE-1 experiences fluctuating low throughput due to its poor channel condition, then a high delay. The PF schduling (for intra-slice) allocates more RBs for UE-1 time-to-time to compensate, resulting in latency variations for other UEs with good channel conditions, i.e., UE-2 and UE-3.

5.3 Hybrid Trace-driven Network Simulator

To evaluate the DUAC algorithm, we built a hybrid trace-driven network simulator that trains a DNN to generate the reward trajectories (i.e., total data rate) given network slice management actions in each episode (i.e., resource allocation and intra-slice scheduling policy) based on an open-source O-RAN dataset and the real data collected from testbed. These datasets contain time-varying network state information and slice/UE statistics for multiple cells under different network conditions. We evaluate three scenarios in O-RAN datasets after data collection procedure described in section 5.2:

- **Scenario 1**(*slice mixed slow close*): UEs belonging to different traffic classes are randomly distributed across slices(slice mixed) and uniformly distributed within $20m$ of each BS(close). UE's mobility is slow ($3m/s$);
- **Scenario 2** (*slice mixed static close*): UEs belonging to different traffic classes are randomly distributed across slices(slice mixed) and uniformly distributed within $20m$ of each BS(close). UE's mobility is static (no mobility);
- **Scenario 3** (*slice traffic static far*): are divided per slice based on traffic types(slice traffic), i.e., Slice 0: eMBB UEs; Slice 1: MTC UEs; Slice 2: URLLC UEs. And UEs are uniformly distributed within $100m$ of each BS(far). UEs has no mobility;

To build a realistic simulator, we extract metrics for various types of UEs from multiple training sessions to ensure diversity of features. The extracted metrics are combined into a dataset and sorted in ascending order based on the *Timestep* variable to create a time series sequential data. To predict the download link data rate, we formulate a multivariate time series forecasting problem using a Long Short-Term Memory (LSTM) model developed with Keras. The dataset is framed as a forecasting problem where the download link data rate at the next time step is predicted based on the network conditions and download link data rate at the previous time step. The dataset is normalized using MinMaxScaler, and split into training and testing samples with a 7/3 ratio. The LSTM model consists of 50 neurons in the first hidden layer and one neuron in the output layer. The Mean Absolute Error (MAE) loss function and the efficient Adam version of stochastic gradient descent are used



(a) Training Loss (b) Testing Loss

Figure 3: Training and Testing losses(Y-axis) are decreasing as training steps(X-axis) increase indicating the simulator predicts download link data rates efficiently in all 3 scenarios.

to optimize the model. The simulator is trained for 100 epochs with a batch size of 72. Figure 3 shows the progress of simulator model training and testing loss over 100 steps for all three scenarios. The loss values decrease monotonically, indicating increased accuracy in predicting download link data rates. The model does not overfit and generalizes well, as evidenced by the convergence of both training and testing loss towards a minimum value.

5.4 Evaluation Results

In the evaluations, we consider the utility U as a class of α-fair utility functions, which are widely used in network optimization [17]. For some constant $\alpha \geq 0$, define α-fair utility as:

$$U(x) = \begin{cases} x^{1-\alpha}/(1-\alpha) & \text{for } \alpha \neq 1, \\ \log(x) & \text{for } \alpha = 1. \end{cases} \tag{22}$$

We evaluate our DUAC algorithm in a hybrid trace-driven network simulator designed in Section 5.3 and compare its performance with four baselines, including heuristic and learning methods:

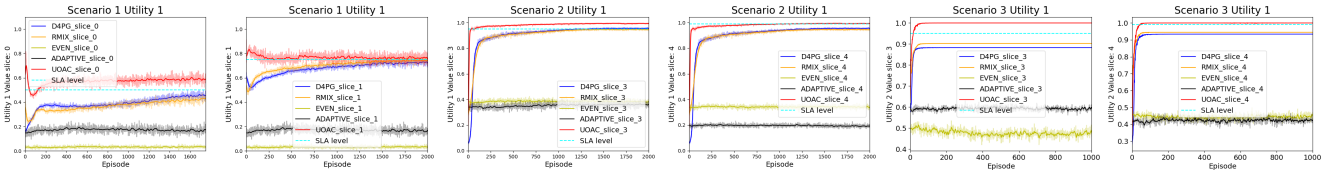
- *DUAC*: The proposed DUAC algorithm optimizing distributional utilities.
- *Even*: A heuristic policy that evenly distributes the PRBs to all slices. Random discounted data rate Z_k will be calculated manually, scheduling policy will be randomly assigned, same as Adaptive policy.
- *Adaptive*: A heuristic policy that dynamically assigns the PRBs in proportion to slices' SLA requirements levels;
- *D4PG*: The D4PG algorithm in [3] is for optimizing the expected state-action function Q_π by taking the expectation over distribution Z_π which is modeled by a DNN.
- *RMIX*: The RMIX algorithm in [22] is an RL method learning the Conditional Value at Risk (CVaR) measure over the distributions of individuals' Q values; it optimizes the policies to maximize the CVaR value.

The actor and critic neural networks in our experiments consist of three hidden layers with 64 neurons and ReLu activation functions. We use the Adam optimizer to train the learning algorithms until convergence, with each episode consisting of 100 steps. The initial learning rate for the actor network is 1×10^{-4}, while the critic network has an initial learning rate of 1×10^{-3} to enable faster learning. We set $\gamma = 0.95$ in the critic network to discount the utility value. For the distributional return Z updates, we use 51 atoms for the categorical distribution and the absolute distributional TD-error as described in Section 4.

We compare DUAC and four baselines using utility function $U_k(Z_k) = P\{Z_k > \eta_k\}$ (**Utility 1**), which is the α-fair utility $x^{1-\alpha}/(1-\alpha)$ for $\alpha = 0$ with x representing the distribution function of Z_k, i.e., $P\{Z_k > \eta_k\}$. We set η_k to 0.76, 0.42, 0.30, 0.24, 0.58, and SLA requirement levels σ_k to 50%, 75%, 90%, 95%, 99% for five different network slices across all scenarios described in Section 5.3 (*values remains the same for other utilities*). Since Z is the random return of the total download link data rate, maximizing this utility can be regarded as minimizing the probability of received data rates falling below the prescribed SLAs. The convergence of Utility 1 values is plotted for slices with SLA levels of 50% and 75% in Scenario 1(Figures 4(a) to 4(b)), and for slices with SLA levels of 95% and 99% in Scenarios 2 and 3 (Figures 4(c) to 4(f)). The blue dotted line represents the current slice's SLA requirement level. Our proposed DUAC algorithm outperforms D4PG and RMIX, as D4PG aims to optimize long-term rewards without considering specific distributional utilities; and RMIX cannot meet different agent requirements due to its centralized framework that only optimizes the centralized risk value. DUAC not only meets SLA requirements and achieves higher values than other baselines when the SLA requirements are feasible, but it also satisfies higher SLA requirements where other baselines fail to surpass the requirements.

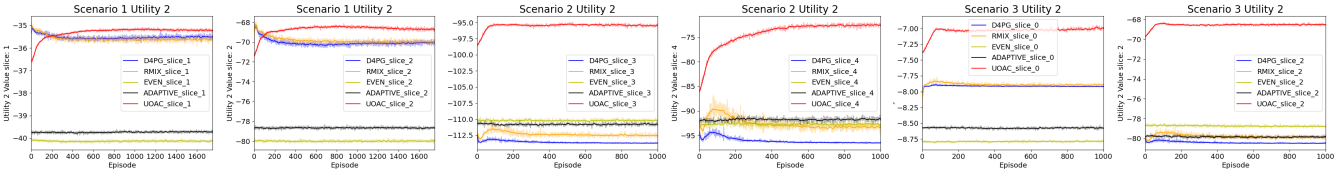
Next, we evaluate the ability of the DUAC algorithm to optimize different utility functions by considering the proportional-fair utility function $\mathbb{P}_{U_k}(Z_k) = \log(P\{Z_k > \eta_k\} - \sigma_k)$(**Utility 2**), which is a special case of the α-fair utility with $\alpha = 1$. Maximizing this utility aims to increase the likelihood of achieving higher data rates η_k closer to the SLA level σ_k. Results in Figure 5 show that DUAC can converge to higher utility values compared to all other baselines. The improvements in distributional utility should be interpreted in the decibel sense due to the logarithm utility function. For instance, as the distributional utility in Figure 5(d) for network slice 4 with SLA requirement $\sigma_4 = 99\%$ improves from -95 (achieved by *RMIX*) to -73 (achieved by *DUAC*), the 23.16% improvement in utility corresponds to $10^{(-73+95)/5} - 1 = 2.51 \times 10^4$ improvement in the geometric-mean reward.

To further investigate the strength of the DUAC algorithm and demonstrate DUAC algorithm's ability to optimize more utility functions, Figure 6 shows the Cumulative Distribution Function (CDF) of the discounted total throughput of each slice (i.e., the CDF of Z_k, k=0,…,4) before and after being trained by our proposed DUAC algorithm in Scenario 2 with **Utility 3**, which is the α-fair utility with $\alpha = 1/2$, i.e, $\sqrt{(P\{Z_k > \eta_k\} - \sigma_k)}$, this function could work for balancing σ_k-percentile latency for different slices or minimizing the probability of each slice's throughput Z_k falling below η_k. Figure 6 shows that the CDF of Z_k generated randomly before training (represented by the blue line) fails to satisfy the performance requirements for each slice k, Taking slice 4 (the last subfigure) with the highest SLA requirement $\sigma_4 = 99\%$ as an example, the probability of Z_4 exceeding the threshold value η_4 is given as $P\{Z_4 > \eta_4\} = 1 - F_{Z_4}(0.58) = 15\% \leq 99\%$, indicating that slice 4 fails to meet the desired requirements prior to training. Upon being trained by the DUAC algorithm, the distribution of the total



(a) slice 0 (b) slice 1 (c) slice 3 (d) slice 4 (e) slice 3 (f) slice 4

Figure 4: Learning curves of Utility 1 $P\{Z_k > \eta_k\}$ values (Y-axis) in Scenario 1(a-b), Scenario 2(c-d) and Scenario 3(e-f) for slices with different target utility parameters η_k and SLA requirement levels σ_k(represented by the blue dotted line). The slice learning agents trained by DUAC algorithm(red curve) converge to higher values of utility 1 compared to other baseline models and can surpass the specified SLA requirement level σ_k, whereas other baselines fail to meet the SLA requirements.



(a) slice 1 (b) slice 2 (c) slice 3 (d) slice 4 (e) slice 0 (f) slice 2

Figure 5: Learning curves of Utility 2 $\log(P\{Z_k > \eta_k\} - \sigma_k)$ values (Y-axis) in Scenario 1(a-b), Scenario 2(c-d) and Scenario 3(e-f) for slices with different target utility parameters η_k and SLA requirement levels σ_k. The slice learning agents trained by DUAC algorithm(red curve) have the ability to converge towards higher utility values when compared to other baseline models.

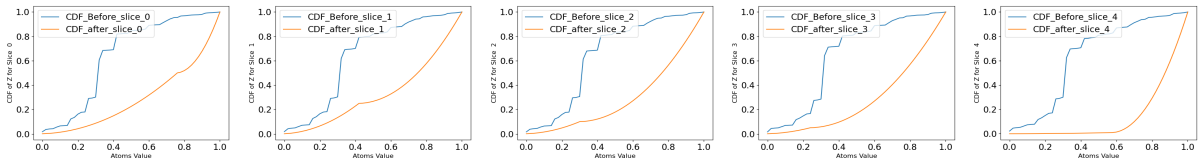


Figure 6: The DUAC algorithm adjusts return Z_k distributions for every slice k represented by the Cumulative Distribution Function (CDF) F_{Z_k} (with the original CDF shown in blue curves and the DUAC-trained CDF shown in orange curves) to successfully meet the SLA requirement of each slice while considering Utility 3 $\sqrt{(P\{Z_k > \eta_k\} - \sigma_k)}$ in scenario 2.

throughput Z_k is adjusted to increase the probability of Z_k exceeding the threshold value η_k beyond the level of σ_k in accordance with distinct requirements for each slice k. Taking slice 4 as an example again, after being trained by DUAC, the corresponding orange line shows that $P\{Z_4 > \eta_4\} = 1 - F_{Z_4}(0.58) = 1 - 0.01 = 99\%$, indicating that the SLA requirement level 99% has been met for network slice 4. The previous calculations are equally applicable to slices 0 to 3, thereby demonstrating DUAC's capability to optimize utility values that satisfy various requirements in network slicing problems by adjusting the total throughput distribution for each slice k based on distinct requirements η_k and σ_k. To assess the DUAC algorithm's capability to meet the SLA requirements with Utility 3 in scenarios 1 and 3, Table 1 presents the results. It reveals that following the training process by the DUAC algorithm, all the network slice performances could meet the targeted SLA requirements.

To demonstrate overall performance of all utilities in solving the optimization problem presented in Eq. 1 across all scenarios, the total utility values of all slices over the entire throughput distribution were plotted in Figure 7 and 8. Specifically, Figure 7 compares the total utility values for utility 1 ($U_k(Z_k) = P\{Z_k > \eta_k\}$) and utility 3 ($\sqrt{(P\{Z_k > \eta_k\} - \sigma_k)}$) in all scenarios, while Figure 8 shows the comparison for utility 2 ($\log(P\{Z_k > \eta_k\} - \sigma_k)$) as they exhibit negative values. The results indicate that DUAC algorithm(red bars in both figures) can achieve a higher utility value of the total throughput distribution compared to all baselines, thereby demonstrating its effectiveness in handling arbitrary utility functions.

Table 1: All network slices satisfy the SLA requirements when considering utility 3 $\sqrt{(P\{Z_k > \eta_k\} - \sigma_k)}$ in scenarios 1 and 3.

Slice	η_k	σ_k	Scenario 1	Scenario 3
0	0.76	50%	59.2%	52.4%
1	0.42	75%	78.5%	80.3%
2	0.30	90%	93.8%	90.6%
3	0.24	95%	97.5%	97.3%
4	0.58	99%	99.2%	99.8%

6 CONCLUSION

We propose a new RL algorithm for online network slice management using distributional utilities. Our approach updates the critic's value function through stochastic gradient descent and shows convergence to a stationary point of the DUO problem. Validation with a hybrid network simulator demonstrates significant improvements

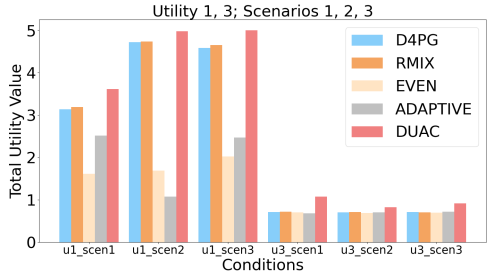


Figure 7: Comparing the total utility values w.r.t. utility 1 $U_k(Z_k) = P\{Z_k > \eta_k\}$) and utility 3 ($\sqrt{(P\{Z_k > \eta_k\} - \sigma_k)}$, summed over all network slices in each scenario. Y-axis: total utility values, X-axis: indicates conditions of utility i in scenario k. The results indicate that DUAC (represented by red bars) achieves higher total utility values in all scenarios.

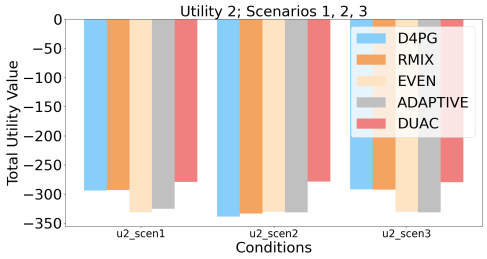


Figure 8: Comparing the total utility values w.r.t. utility 2 $\log(P\{Z_k > \eta_k\} - \sigma_k)$, summed over all network slices in each scenario. The results indicate that DUAC (represented by red bars) achieves higher total utility values. Note that Y-axis is in log due to the use of proportional-fair utility.

over baselines. Future work includes real-world deployment in a 5G testbed. This work was supported by ONR Grant N000142012146.

REFERENCES

[1] O-RAN Alliance. 2022.03. O-RAN Architecture-Description. Version 6.00.

[2] O-RAN Alliance. 2022.07. O-RAN Near-Real-time RAN Intelligent Controller E2 Service Model (E2SM) RC. Version 1.02.

[3] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. Distributed Distributional Deterministic Policy Gradients. https://doi.org/10.48550/ARXIV.1804.08617

[4] Marc G. Bellemare, Will Dabney, and Rémi Munos. 2017. A Distributional Perspective on Reinforcement Learning. https://doi.org/10.48550/ARXIV.1707.06887

[5] Dimitri Bertsekas and John N Tsitsiklis. 1996. *Neuro-dynamic programming*. Athena Scientific.

[6] Leonardo Bonati, Salvatore D'Oro, Lorenzo Bertizzolo, Emrecan Demirors, Zhangyu Guan, Stefano Basagni, and Tommaso Melodia. 2020. CellOS: Zero-touch Softwarized Open Cellular Networks. *Computer Networks* 180 (oct 2020), 107380. https://doi.org/10.1016/j.comnet.2020.107380

[7] Leonardo Bonati, Salvatore D'Oro, Michele Polese, Stefano Basagni, and Tommaso Melodia. 2021. Intelligence and Learning in O-RAN for Data-Driven NextG Cellular Networks. *IEEE Communications Magazine* 59, 10 (oct 2021), 21–27. https://doi.org/10.1109/mcom.101.2001120

[8] Jingliang Duan, Yang Guan, Shengbo Eben Li, Yangang Ren, Qi Sun, and Bo Cheng. 2021. Distributional Soft Actor-Critic: Off-Policy Reinforcement Learning for Addressing Value Estimation Errors. *IEEE Transactions on Neural Networks and Learning Systems* (2021), 1–15. https://doi.org/10.1109/tnnls.2021.3082568

[9] Salvatore D'Oro, Francesco Restuccia, Alessandro Talamonti, and Tommaso Melodia. 2019. The Slice Is Served: Enforcing Radio Access Network Slicing in Virtualized 5G Systems. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications* (Paris, France). IEEE Press, 442–450. https://doi.org/10.1109/INFOCOM.2019.8737481

[10] Romano Fantacci and Benedetta Picano. 2020. When Network Slicing Meets Prospect Theory: A Service Provider Revenue Maximization Framework. *IEEE Transactions on Vehicular Technology* 69, 3 (2020), 3179–3189. https://doi.org/10.1109/TVT.2019.2963462

[11] Xenofon Foukas, Georgios Patounas, Ahmed Elmokashfi, and Mahesh K. Marina. 2017. Network Slicing in 5G: Survey and Challenges. *IEEE Communications Magazine* 55, 5 (2017), 94–100. https://doi.org/10.1109/MCOM.2017.1600951

[12] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2017. Rainbow: Combining Improvements in Deep Reinforcement Learning. https://doi.org/10.48550/ARXIV.1710.02298

[13] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. 2018. Distributed Prioritized Experience Replay. https://doi.org/10.48550/ARXIV.1803.00933

[14] Yohan Kim, Sunyong Kim, and Hyuk Lim. 2019. Reinforcement Learning Based Resource Management for Network Slicing. *Applied Sciences* 9 (06 2019), 2361. https://doi.org/10.3390/app9112361

[15] Yohan Kim and Hyuk Lim. 2021. Multi-Agent Reinforcement Learning-Based Resource Management for End-to-End Network Slicing. *IEEE Access* 9 (2021), 56178–56190. https://doi.org/10.1109/ACCESS.2021.3072435

[16] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980 cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[17] Tian Lan, David T. H. Kao, Mung Chiang, and Ashutosh Sabharwal. 2009. An Axiomatic Theory of Fairness. *CoRR* abs/0906.0557 (2009). arXiv:0906.0557 http://arxiv.org/abs/0906.0557

[18] Qiang Liu and Tao Han. 2019. DIRECT: Distributed Cross-Domain Resource Orchestration in Cellular Edge Computing. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing* (Catania, Italy) *(Mobihoc '19)*. Association for Computing Machinery, New York, NY, USA, 181–190. https://doi.org/10.1145/3323679.3326516

[19] Yongsheng Mei, Hanhan Zhou, and Tian Lan. 2023. ReMIX: Regret Minimization for Monotonic Value Function Factorization in Multiagent Reinforcement Learning. arXiv:2302.05593 [cs.LG]

[20] Yongsheng Mei, Hanhan Zhou, Tian Lan, Guru Venkataramani, and Peng Wei. 2023. MAC-PO: Multi-Agent Experience Replay via Collective Priority Optimization. arXiv:2302.10418 [cs.LG]

[21] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. (2016). https://doi.org/10.48550/ARXIV.1602.01783

[22] Wei Qiu, Xinrun Wang, Runsheng Yu, Xu He, Rundong Wang, Bo An, Svetlana Obraztsova, and Zinovi Rabinovich. 2021. RMIX: Learning Risk-Sensitive Policies for Cooperative Reinforcement Learning Agents. https://doi.org/10.48550/ARXIV.2102.08159

[23] R. Tyrrell Rockafellar and Stanislav Uryasev. 2000. Optimization of conditional value-at-risk. https://doi.org/10.21314/JOR.2000.038

[24] Josep Xavier Salvat, Lanfranco Zanzi, Andres Garcia-Saavedra, Vincenzo Sciancalepore, and Xavier Costa-Perez. 2018. Overbooking Network Slices through Yield-Driven End-to-End Orchestration. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies* (Heraklion, Greece) *(CoNEXT '18)*. Association for Computing Machinery, New York, NY, USA, 353–365. https://doi.org/10.1145/3281411.3281435

[25] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized Experience Replay. https://doi.org/10.48550/ARXIV.1511.05952

[26] Melvyn Sim. 2004. *Robust optimization*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[27] Wei-Fang Sun, Cheng-Kuang Lee, and Chun-Yi Lee. 2021. DFAC Framework: Factorizing the Value Function via Quantile Mixture for Multi-Agent Distributional Q-Learning. https://doi.org/10.48550/ARXIV.2102.07936

[28] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* 12 (1999).

[29] Yichuan Charlie Tang, Jian Zhang, and Ruslan Salakhutdinov. 2019. Worst Cases Policy Gradients. https://doi.org/10.48550/ARXIV.1911.03618

[30] Derek Yang, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tieyan Liu. 2019. Fully Parameterized Quantile Function for Distributional Reinforcement Learning. https://doi.org/10.48550/ARXIV.1911.02140

[31] Hanhan Zhou, Tian Lan, and Vaneet Aggarwal. 2022. PAC: Assisted Value Factorization with Counterfactual Predictions in Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems* 35 (2022), 15757–15769.

Received 10 March 2023; accepted 14 July 2023