A Deep-Reinforcement-Learning-Based Approach to Dynamic eMBB/URLLC Multiplexing in 5G NR

Yan Huang, *Student Member, IEEE*, Shaoran Li, *Student Member, IEEE*, Chengzhang Li, *Student Member, IEEE*, Y. Thomas Hou, *Fellow, IEEE*, and Wenjing Lou, *Fellow, IEEE*

***Abstract*—This article investigates the dynamic multiplexing of enhanced mobile broadband (eMBB) and ultrareliable and low latency communications (URLLC) on the same channel in 5G NR. Due to significant difference in transmission time scale, URLLC employs a preemptive puncturing technique to multiplex its traffic onto eMBB traffic for transmission. The optimization problem to solve is to minimize the adverse impact of such preemptive puncturing on eMBB users. We present DEMUX—a model-free deep reinforcement learning (DRL)-based solution to this problem. The essence of DEMUX is to use deep function approximators (neural networks) to learn an optimal algorithm for determining the preemption solution in each eMBB transmission time interval (TTI). Our novel contributions in the design of DEMUX include the first use of the DRL method with a large and continuous action domain for resource scheduling in NR, a mechanism to ensure fast and stable learning convergence by exploiting the intrinsic properties of the problem, and a mechanism to obtain a feasible preemption solution from the unconstrained output of a neural network while minimizing loss of information. The experimental results show that DEMUX significantly outperforms state-of-the-art algorithms proposed in the 3GPP standards body and the literature.**

***Index Terms*—5G NR, deep reinforcement learning (DRL), enhanced mobile broadband (eMBB)/ultrareliable and low latency communication (URLLC) multiplexing, preemption, puncturing, resource allocation.**

I. Introduction

A MAJOR challenge in 5G NR cellular networks is to support service types with extremely diverse performance requirements with a unified air interface [1]. Enhanced mobile broadband (eMBB) and ultrareliable and low latency communications (URLLC) are two such types of services in NR [2]. eMBB aims to offer a per-user data rate higher than 100 Mb/s [3], while URLLC targets at mission-critical applications with low data rates (0.1–10 Mb/s) but extremely stringent latency requirements on ∼1-ms time scale [4]. To meet their service requirements, eMBB and URLLC employ very different time duration for data transmission. In NR, each time slot is divided into a number of mini-slots [5]. The eMBB uses time slots for transmission in order to achieve high data throughput. On the other hand, the URLLC data are transmitted in mini-slots so as to minimize latency.

When eMBB and URLLC services are multiplexed on the same channel, the optimal allocation of radio resources becomes a challenging problem. To address this problem, a novel resource allocation scheme termed "URLLC preemption," was proposed in 3GPP standards body [6]. The essence of this scheme is to have an arriving URLLC packet preempt resources that have already been allocated to the eMBB users. Specifically, when a URLLC packet arrives, its transmission is scheduled immediately for the next mini-slot without waiting for the end of eMBB transmissions. Depending on the packet size and the selected modulation and coding scheme (MCS), a URLLC packet requires a certain number of subcarriers (SCs) to transmit. These SCs are obtained by directly puncturing the time–frequency resource grid in the scheduled mini-slot. The punctured SCs spread across resources allocated to different eMBB users. To keep track of (pinpoint) which resources have been taken by URLLC, special indication signals are generated and sent to the eMBB users, which will be used for decoding. It has been shown that this preemptive puncturing approach is more spectrally efficient than static or semistatic spectrum separation between eMBB and URLLC, due to the sporadic and random nature of the URLLC traffic [7].

Under this preemptive puncturing approach, an important optimization problem is the allocation of SCs preempted by each URLLC transmission among the eMBB users. The decoding result of an eMBB user's received transmission directly depends on the URLLC preemption: the more SCs being punctured by URLLC, the more likely the decoding will fail. Such a decoding failure will lead to a loss of eMBB utility (e.g., the sum of eMBB users' data rates or weighted rates). Therefore, the objective of our optimization problem is to minimize the loss of eMBB utility through properly spreading out preempted resources (for URLLC) among the eMBB users in each transmission.

Unfortunately, the above optimization problem cannot be formulated and solved through traditional model-based optimization approaches. This is because the objective (cost) function of the problem requires the modeling of eMBB decoding behavior (e.g., failure probability) as a function of the amount of URLLC preemption in exact closed form. However, due to the complexities of NR PHY layer technologies such as LDPC channel code, it is impossible to obtain a closed-form expression for this cost function. As will

Manuscript received October 29, 2019; revised February 6, 2020 and February 25, 2020; accepted February 29, 2020. Date of publication March 5, 2020; date of current version July 10, 2020. This work was supported in part by NSF under Grant 1642873. *(Corresponding author: Y. Thomas Hou.)*

The authors are with the Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA (e-mail: huangyan@vt.edu; shaoran@vt.edu; licz17@vt.edu; thou@vt.edu; wjlou@vt.edu).

Digital Object Identifier 10.1109/JIOT.2020.2978692

be explained in Section IV, other approximate modeling or numerical approaches are also unsuitable for addressing this problem.

In this article, we present "DEMUX," a model-free deep (DE)-reinforcement-learning-based multiplexer (MUX) for the eMBB and URLLC services. The most significant advantage of DEMUX is that it offers a solution to the URLLC preemption problem without a prior explicit problem formulation. DEMUX uses deep function approximators (also known as, neural networks) to learn an optimal algorithm for determining the preemption solution. Architecturally, DEMUX consists of a learning plane and a scheduling plane. The goal of the learning plane is to learn an optimal algorithm for the URLLC preemption. The goal of the scheduling plane is to use the algorithm learned by learning plane to compute the preemption solution for URLLC. Our main contributions in the design of DEMUX are summarized as follows.

1) This article presents the first solution to the URLLC preemption problem that utilizes deep reinforcement learning (DRL). In the broader context, this is also the first work (to the best of our knowledge) that employs the DRL method with large and continuous action domain to address resource scheduling problems in NR. This is significant as deep Q-learning (DQL) methods (the most commonly used learning methods for wireless communications) are only effective to address problems with discrete and small action space but are not suitable to solve problems with large and continuous action domain, such as the URLLC preemption problem that we study in this article.
2) On the learning plane of DEMUX, although our design is inspired by the deep deterministic policy gradient (DDPG) method [27], we find that there exists a serious convergence issue with this approach in addressing our problem. To mitigate this issue, we propose to augment the DDPG method by exploiting some intrinsic properties of the URLLC preemption problem. This includes adapting the learning objective of DDPG based on the scheduling mechanism of eMBB and eliminating the use of target networks in DDPG without hampering the stability of the learning process.
3) On the scheduling plane, a major problem is that the learned algorithm, which is an unconstrained neural network, cannot guarantee to offer a feasible solution for the URLLC preemption. To address this problem, we propose a novel approach based on the concept of relative entropy in the information theory to translating the raw output of the learned algorithm (neural network) into a feasible preemption solution. We show that our proposed solution is optimal in terms of minimizing the loss of information caused by such translation.
4) In our implementation, we build a PHY–MAC NR simulator for dynamic eMBB/URLLC multiplexing and a learning module based on our design of DEMUX. Our implementation captures essential functions in PHY and MAC layers of NR that are related to URLLC preemption as well as interaction mechanisms between an NR base station (BS) and a DRL learning module. Our NR simulator is tuned and validated through extensive experiments to ensure the correctness of the PHY- and MAC-layer functions [e.g., the measurement of working SNRs to ensure the target block error rates (BLERs) for different MCS levels, validation for soft demodulation and decoding under URLLC preemption, among others].
5) Using our NR simulator, we validate the performance of DEMUX through benchmarking with two other preemption schemes proposed in the literature (resource proportional (RP) from [10] and fixed-frequency-part (FFP) from 3GPP standards body [8], [9]) under different multipath channel fading models. These benchmark schemes were developed based on various modeling assumptions and thus are considered as heuristic solutions to the URLLC preemption problem. Experiment results show that DEMUX significantly outperforms RP and FFP in terms of both sum utility and throughput in a cell. In particular, when the URLLC traffic load is high, DEMUX is able to achieve more than 130% and 75% performance gains over RP and FFP, respectively, in our test network scenarios.

The remainder of this article is organized as follows. In Section II, we provide the necessary background on dynamic eMBB/URLLC multiplexing and introduce the optimal URLLC preemption problem. In Section III, we review existing works on the URLLC preemption problem in the literature. In Section IV, we go deep into technical challenges in solving the problem and show why model-based optimization approaches cannot be applied. In Section V, we give an overview of DEMUX—our proposed solution to the URLLC preemption problem. In Section VI, we present DEMUX's scheduling plane, while in Section VII, we present DEMUX's learning plane. Our implementation of DEMUX is presented in Section VIII. Section IX presents experimental results based on our implementation. Section X concludes this article.

II. Dynamic Multiplexing of eMBB and URLLC

When the eMBB and URLLC services are multiplexed on the same downlink channel, the allocation of time–frequency resources becomes very complicated. This is because eMBB and URLLC transmissions have different duration and are scheduled on very different time scales. In NR, the time domain of a channel is divided into time slots, with each time slot being further divided into multiple mini-slots. The duration of an eMBB transmission, termed as the transmission time interval (TTI), may span one or multiple time slots. In contrast, each URLLC packet is transmitted using a mini-slot.

URLLC Preemptive Puncturing: In this article, we investigate "URLLC resource preemption," a mechanism for dynamic multiplexing of eMBB and URLLC that was proposed in the 3GPP standards body [6]. Fig. 1 illustrates this mechanism. The frequency domain of a channel consists of a large number of contiguous SCs. In each TTI, all SCs on the channel are allocated to the eMBB users using some scheduling algorithm (e.g., proportional fairness (PF) [16], [17]). On the other hand,

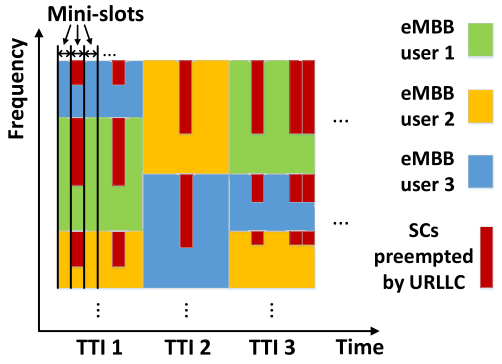


Fig. 1. Illustration of URLLC resource preemption.

URLLC packet arrivals are sporadic and random in time. When a URLLC packet arrives, its transmission is scheduled immediately for the next mini-slot.[1] Each URLLC packet requires a certain number of SCs to transmit (smaller than the total number of SCs on the channel). As all SCs have already been allocated to eMBB users, an incoming URLLC packet obtains its required SCs by puncturing/preempting resources from ongoing eMBB transmissions. That is, a portion of SCs allocated to each eMBB user are reassigned to URLLC in the scheduled mini-slot. As shown in Fig. 1, the preempted SCs do not need to be contiguous in the frequency [10]. Then, the data bits of eMBB are flushed out on those SCs preempted by URLLC.

In the example shown in Fig. 1, there are three, two, and three eMBB transmissions in TTI 1, 2, and 3, respectively. The numbers of URLLC transmissions in TTI 1, 2, and 3 are two, one, and three, respectively. Specifically, in TTI 1, there is a URLLC transmission in the second mini-slot and another one in the fifth mini-slot. Note that the SCs punctured by each URLLC packet are spread out across multiple eMBB users.

Under this preemption mechanism, eMBB code word bits are corrupted on SCs punctured by URLLC. To help the eMBB users decode, the BS sends an indication signal to each eMBB user at the end of a TTI to specify positions of preempted resources in time and frequency [22], [24]. Based on this indication, the eMBB users can precisely eliminate the corrupted bits before LDPC decoding. This will avoid polluting other code word bits during decoding and improve the decoding success probability.

Problem Statement: For each TTI, eMBB transmissions are scheduled with a specific objective, e.g., maximizing the sum utility of the eMBB users (such as the sum cell throughput or PF utility). The actually achieved eMBB utility, however, depends on whether or not each eMBB user can successfully decode its received transmission. If all transmissions are successful, then the maximum utility is preserved, while failed transmissions result in a loss of utility.

As expected, the decoding failure probability of an eMBB transmission increases with the amount of URLLC preemption. Since each URLLC packet can preempt SCs from multiple eMBB users, an important optimization problem to solve is: For each URLLC transmission, how many SCs should we preempt from each eMBB user so that the loss of sum eMBB utility is minimized? This is the problem that we will investigate in this article.

For each eMBB user, we assume that a contiguous block of SCs starting from the lowest/highest frequency will be preempted (when preemption happens). We will not further optimize positions of preempted SCs within each eMBB user's resources since such optimization will incur excessive overhead for feedback and control signaling. A practical set of constraints for URLLC puncturing is that the number of SCs preempted from each eMBB user cannot exceed the number of SCs allocated to her. Such constraints must be satisfied by any feasible preemption solution for URLLC transmissions.

III. Related Work

There have been a number of previous works on dynamic multiplexing of eMBB and URLLC based on preemptive puncturing [8]–[11]. In 3GPP standards body [8], [9], a very simple (and easy to implement) preemption scheme was proposed. We call this scheme FFP in this article. FFP statically assigns a contiguous block of SCs on the channel for each URLLC transmission. As we will show in Section IX, such a static allocation (preemption) scheme achieves poor performance in terms of maximizing eMBB utility. This demonstrates the critical need for optimizing URLLC preemption among the eMBB users, which is the problem that we study in this article.

The works in [10] and [11] explored optimizations of the URLLC preemption based on assumptions of specific mathematical models for the cost function. Here, the cost function represents the loss of eMBB utility as a function of the amount of resources (SCs) preempted from each eMBB user. Under this approach, Anand *et al.* [10] assumed different cost function models (linear or nonlinear) and developed a customized solution for each model. Specifically, under a linear cost function model, a solution that preempts SCs in proportion to each eMBB user's resource allocation was found to be optimal. Under a nonlinear cost function model, a nonclosed-form solution based on the standard optimization formulation was proposed. Bairagi *et al.* [11] assumed a linear model for cost function and formulated an optimization problem for resource allocation between eMBB and URLLC. Then, a matching-based algorithm was proposed to solve the formulated problem.

A recent work [12] studied machine-learning-based resource scheduling for eMBB/URLLC multiplexing on the same channel. The flexible setting of TTI duration for different services was considered as an enhancement to the traditional fixed-length TTI-based scheduling. However, the multiplexing mechanism considered in [12] is fundamentally different from the URLLC puncturing scheme that we study in this article. The proposed random forest-based classification algorithm cannot be easily extended to address our URLLC preemption optimization problem.

[1] In this article, we assume that at most one URLLC packet can be transmitted in a mini-slot. This is because a URLLC transmission generally requires very broad frequency bandwidth (may be more than half of the entire channel bandwidth).

IV. Why Model-Based Approaches Will Not Work

Although a model-based approach is attractive from a mathematical perspective, it suffers from the following major flaw. Specifically, the underlying assumption that the optimal URLLC preemption problem can be abstracted and modeled with a mathematical formulation does not hold. The main reason here is that the cost (objective) function of the problem cannot be modeled in closed form. Inputs to this cost function include URLLC preemption among the eMBB users, MCS selection and SC allocation for each eMBB user, channel conditions across SCs, among others. As we shall discuss in the rest of this section, it is mathematically intractable to express this cost function through either exact analytical approaches, simplified approximations, or numerical methods.

First, the cost function cannot be obtained through analytical abstraction and modeling due to the complexities involved in the NR PHY layer. In NR, LDPC is employed as the forward error correction (FEC) channel code for data transmission [21], [24]. It is well known that LDPC decoding performance is very difficult to analyze and cannot be modeled exactly. There is no closed-form expression for the error-correction capability of LDPC.[2] As a result, we cannot obtain an explicit expression for the BLER performance of eMBB. In practice, the BLER curves can only be determined numerically through extensive experiments or simulations [24].

Second, given that an exact model for the cost function is not obtainable, one might attempt to develop a simplified approximate model and use it for analysis and optimization. Unfortunately, such an approach is unlikely to be successful because simple approximations are unable to accurately characterize the complex impact of the URLLC preemption on eMBB performance. As an example, in [10], a linear model was assumed for the cost function and subsequently, a so-called *RP placement* algorithm was proposed and found to be optimal under this linear cost function. But its actual performance under practical system settings is rather poor (see Section IX), due to the inaccuracy of a linear model.

Finally, one might wonder if it is possible to estimate the cost function numerically by enumerating all possible inputs, which include, among others, the amount of URLLC preemption, MCS and SC assignment for each eMBB user, and channel conditions across SCs. However, the input space of this approach is prohibitively large, rendering this numerical approach practically infeasible.

V. DEMUX: Overview

In light of discussions in the last section, it is clear that a very different approach is needed to address the URLLC preemption problem. In this article, we propose DEMUX—a novel model-free DRL-based solution to this problem. In this section, we offer an overview of the overall architecture of DEMUX. More design details will be presented in Sections VI and VII.

[2]Refer to [25] for a state-of-the-art lower bound for the error-correction capability of LDPC, which is not in closed form.

A. Why DRL?

Model-free DRL is a machine learning technique that employs deep function approximators (neural networks) to learn an *unknown* optimal solution algorithm without prior problem modeling. For addressing our URLLC preemption problem, the most attractive feature of model-free DRL is that it does not require an explicit cost function formulation. In addition, DRL is particularly suitable for solving problems with very large input spaces (e.g., the space of all possible inputs for our problem). An algorithm (a neural network) learned properly through DRL can perform well for input instances that have never been tried in the learning phase (the so-called *generalization capability* of neural networks [32]). Thus, it is possible to learn an optimal algorithm from a relatively small subset of input instances. Finally, DRL is extremely versatile and can learn a wide range of complex algorithms with suitable neural network structures [32]. For example, by adding more fully connected layers to a neural network, one can generally enhance its capability of representing more complex functions. But with more layers, a neural network will have longer forward propagation time. When addressing problems with real-time requirements (as our problem), there is a design tradeoff between the function representation capability of a neural network and its computational time cost.

B. Architecture of DEMUX

At the beginning of each TTI, we have no knowledge of whether or not and how many URLLC packets will arrive among the mini-slots in this TTI. The only available information is URLLC's estimated arrival rate (based on online measurements). With this limited information, the best thing we can do is to determine a preemption solution based on the estimated URLLC arrival rate, which will be used for all URLLC packets that will actually arrive in this TTI. This preemption solution specifies how many SCs should be preempted in a mini-slot from each eMBB user for a URLLC transmission (see Fig. 1). Therefore, at the beginning of each mini-slot, if a URLLC packet is present, then its required SCs are obtained by puncturing resources from the eMBB users following the preemption solution for this TTI. On the other hand, if there is no URLLC transmission in a mini-slot, then no preemption will occur.

The goal of DEMUX is, therefore, to obtain an optimal algorithm to compute a URLLC preemption solution for each TTI. Suppose there exists an *unknown* optimal algorithm for the URLLC preemption, denoted by π, that can minimize the loss of eMBB utility in each TTI (under a given URLLC arrival rate). Then, our goal for DEMUX is to find an approximation μ of this optimal algorithm π using a DRL-based approach. In particular, μ is a deep function approximator (neural network) with a large number of trainable parameters within its structure. In fact, since the URLLC packet arrival rate may change over time, we will learn a different algorithm μ for each different (sampled) URLLC arrival rate (see Section VIII-C).

Table I lists notation used in this article. The architecture of DEMUX consists of a *scheduling plane* and a *learning*

TABLE I
NOTATION

Symbol	Definition
C_n	The long-term average data rate of eMBB user n
f	The translator function
g	The post-processor function
K	The subset size for random sampling from replay buffer
L_{URLLC}	The number of SCs required by a URLLC transmission
M^{SC}	Total number of SCs on the channel
M^{RBG}	Total number of RBGs on the channel
Q	The critic function approximator (neural network)
$r(t)$	The reward function in TTI t
$\boldsymbol{s}(t)$	The instance of eMBB transmissions scheduled in TTI t
t	The TTI index
$U(t)$	The eMBB utility metric in TTI t
$\boldsymbol{\alpha}(t)$	The output preemption instruction of μ in TTI t
$\tilde{\boldsymbol{\alpha}}(t)$	The preemption instruction refined by g in TTI t
$\boldsymbol{\beta}(t)$	The URLLC preemption solution in TTI t
$\boldsymbol{\gamma}(t)$	The resource allocation vector for eMBB in TTI t
$\boldsymbol{\delta}(t)$	The MCS selection vector for eMBB in TTI t
$\boldsymbol{\epsilon}(t)$	The indicator for eMBB transmission failures in TTI t
Φ	The collection of trainable parameters in Q
Θ	The collection of trainable parameters in μ
λ	URLLC packet arrival rate (in unit of arrivals/TTI)
μ	The deep function approximator (neural network) used for approximating π
π	An (unknown) optimal algorithm for URLLC preemption
$\bar{\pi}$	$= f \circ g \circ \mu$, the overall algorithm in the scheduling plane

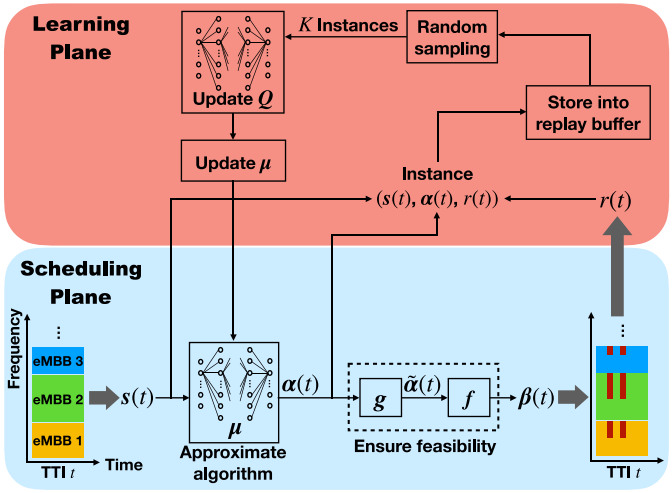


Fig. 2. Scheduling plane and learning plane in DEMUX.

plane, similar to the data plane and the control plane in computer networks. Fig. 2 illustrates our algorithms on both planes. The role of the learning plane is to optimize parameters in μ through a large number of learning steps. The role of the scheduling plane is to use algorithm μ for real-time multiplexing of eMBB and URLLC traffic. In the learning phase, the two planes work in a closed loop through a series of procedures to optimize μ so that it can approximate an optimal algorithm π for the URLLC preemption. The controller of the whole learning process resides at the BS. The eMBB users only need to report their decoding results (success or failure) to the BS, which will then use them to construct a reward function.

Scheduling Plane: The input to this plane is a vector $\boldsymbol{s}(t)$ that describes the state of eMBB transmissions scheduled in TTI t. $\boldsymbol{s}(t)$ contains information of resource allocation and MCS selection for each eMBB user. The input vector $\boldsymbol{s}(t)$ first goes through algorithm μ (neural network), which produces an output vector $\boldsymbol{\alpha}(t)$. Since $\boldsymbol{\alpha}(t)$ is the unconstrained output of a neural network, it is not guaranteed to be a *feasible* solution for URLLC preemption. To address this issue, we design a post-processor g and a translator f that can obtain a feasible preemption solution $\boldsymbol{\beta}(t)$ based on $\boldsymbol{\alpha}(t)$. The purpose of g is to refine $\boldsymbol{\alpha}(t)$ so that its output $\tilde{\boldsymbol{\alpha}}(t)$ will match the eMBB resource allocation structure as $\boldsymbol{s}(t)$. But $\tilde{\boldsymbol{\alpha}}(t)$ may still violate the constraints that URLLC preemption cannot exceed resource allocation for each eMBB user. Next, we use f to convert $\tilde{\boldsymbol{\alpha}}(t)$ into a feasible solution $\boldsymbol{\beta}(t)$ with a minimum loss of information (based on the Kullback–Leibler (KL) divergence). Entries of $\boldsymbol{\beta}(t)$ indicate the proportion of SCs to be preempted from each eMBB user. Finally, $\boldsymbol{\beta}(t)$ will be used for all URLLC transmissions scheduled in mini-slots within TTI t.

Denote $\bar{\pi}$ as the overall algorithm on the scheduling plane (which takes $\boldsymbol{s}(t)$ as input and outputs preemption solution $\boldsymbol{\beta}(t)$). Then, we have $\bar{\pi} = f \circ g \circ \mu$, where $\circ$ represents the composition of functions. Details for the design of $\bar{\pi}$ will be presented in Section VI. As a performance measure of $\bar{\pi}$, a reward function $r(t)$ is generated and sent to the learning plane.

An important requirement for the scheduling plane is that the total processing time of $\bar{\pi}$ must be less than the duration of a TTI. This is to ensure that preemption solution $\boldsymbol{\beta}(t)$ can be updated in each TTI in a real-time manner. For example, when Numerology 1 (500-μs time slot duration) is used for the channel, the duration of a single-slot TTI is 0.5 ms. Then, the aggregate processing time of μ, g, and f must be no more than 0.5 ms. This real-time requirement is one of our key design targets for the scheduling plane.

Learning Plane: The learning plane is inspired by the DDPG method [27] but with our own contributions to address the URLLC preemption problem. A fundamental issue with using DDPG for our problem is that the learning process is extremely slow (i.e., not converging even over half a million iterations). To tackle this issue, we adapt the learning objective of DDPG based on intrinsic properties of the problem and cut down the number of deep function approximators in DDPG from four to two, i.e., only having μ and Q, where Q is a critic neural network. In Section VII-D, we will show that the proposed learning method is effective for our objective (i.e., minimizing the loss of eMBB utility in each TTI).

Our learning method has an actor–critic structure similar to DDPG (more details in Section VII-B). The learning of the approximate algorithm μ is done with a large number of iterations. In each iteration, a 3-tuple instance $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t), r(t))$ is first received from the scheduling plane. This 3-tuple instance is stored in a replay buffer of finite capacity. When the buffer is full, the oldest instance in it is pushed out (discarded). Then, a subset of K random samples $(\boldsymbol{s}(i), \boldsymbol{\alpha}(i), r(i)), i = 1, \ldots, K$, are taken from the replay buffer. These samples are first used for updating parameters in critic neural network Q. As a key component of the learning plane, Q is used for predicting the expected reward $\hat{r}(t)$ for an instance $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t))$, i.e., the reward one would expect to receive when μ outputs $\boldsymbol{\alpha}(t)$ with input eMBB instance $\boldsymbol{s}(t)$. The parameters in Q are updated by minimizing $\sum_{i=1}^{K}(r(i) - \hat{r}(i))^2$. After updating Q, the K samples

are further used for updating parameters in the algorithm μ. A gradient of Q with respect to parameters in μ is computed with the K samples. Then, the parameters in μ are updated along this gradient. After these updates, the algorithm μ is sent to the scheduling plane for the multiplexing of eMBB and URLLC.

VI. Design of the Scheduling Plane

The overall algorithm on the scheduling plane is $\bar{\pi} = f \circ g \circ \mu$. This section presents how each of these three components of $\bar{\pi}$ is designed. To ease our notation, we omit t after TTI when there is no confusion.

A. Example

To help us understand how $\bar{\pi}$ works in each TTI, consider the following illustrative example (not entirely conforming to the NR standard). Suppose there is a total of 100 SCs on the channel and the minimum resolution for resource allocation to eMBB is 20 SCs. This means that an eMBB user if chosen for SC allocation, can be allocated with either 20, 40, 60, 80, or 100 SCs. It also suggests that there are at most five eMBB users that can be chosen (from the total user pool) for SC allocation and transmission per TTI.

Suppose at the beginning of a TTI, we observe an eMBB instance that is characterized by the following vector:

$$\boldsymbol{s} = \Big[\underbrace{0.2 \;\; 0.4 \;\; 0.2 \;\; 0.2 \;\; \text{null}}_{\text{SC allocation}} \;\; \underbrace{0.21 \;\; 0.58 \;\; 0.10 \;\; 0.39 \;\; \text{null}}_{\text{MCS selection}}\Big]$$

where the left five elements inside the brackets represent SC allocation for the scheduled eMBB users (up to five) and the right five elements represent MCS selection for each eMBB user (in terms of code rates).

The eMBB instance $\boldsymbol{s}$ is fed into the deep function approximator μ. Then, μ outputs an unconstrained preemption instruction $\boldsymbol{\alpha}$ as follows:

$$\boldsymbol{\alpha} = [0.35 \;\; 0.11 \;\; 0.48 \;\; 0 \;\; 0.06].$$

Each element in $\boldsymbol{\alpha}$ represents the proportion of SCs (required by a URLLC packet) that should be preempted from an eMBB user. All elements in $\boldsymbol{\alpha}$ add up to one.

Apparently, $\boldsymbol{\alpha}$ is not a feasible solution, as the fifth element of $\boldsymbol{\alpha}$ should be zero since there are only four eMBB transmissions. To make this correction, $\boldsymbol{\alpha}$ is sent to a post-processor g. g first zeros out redundant entry in $\boldsymbol{\alpha}$ (the fifth entry). Then, a small random positive disturbance (0.02 in this example) is added to the fourth zero entry. This operation is required by the translator f (will be explained in Section VI-C). Finally, g normalizes the sum of all elements to 1. The output of g, denoted as $\tilde{\boldsymbol{\alpha}}$, is

$$\tilde{\boldsymbol{\alpha}} = [0.36 \;\; 0.12 \;\; 0.50 \;\; 0.02 \;\; \text{null}].$$

But $\tilde{\boldsymbol{\alpha}}$ still may not meet feasibility constraints for the URLLC preemption. For example, suppose the current URLLC transmission requires 50 SCs. The third element of $\tilde{\boldsymbol{\alpha}}$ says that 50% of the 50 SCs (=25 SCs) should be preempted from eMBB user 3. But eMBB user 3 is only allocated with 20 SCs ($= 100 \times 20\%$). To address this infeasibility, $\tilde{\boldsymbol{\alpha}}$ is sent to a translator f, which will produce a feasible preemption solution $\boldsymbol{\beta}$ as follows:

$$\boldsymbol{\beta} = [0.43 \;\; 0.15 \;\; 0.40 \;\; 0.02 \;\; \text{null}].$$

Then, $\boldsymbol{\beta}$ will be used for all URLLC transmissions in this TTI.

In the rest of this section, we elaborate on the design details of μ, g, and f.

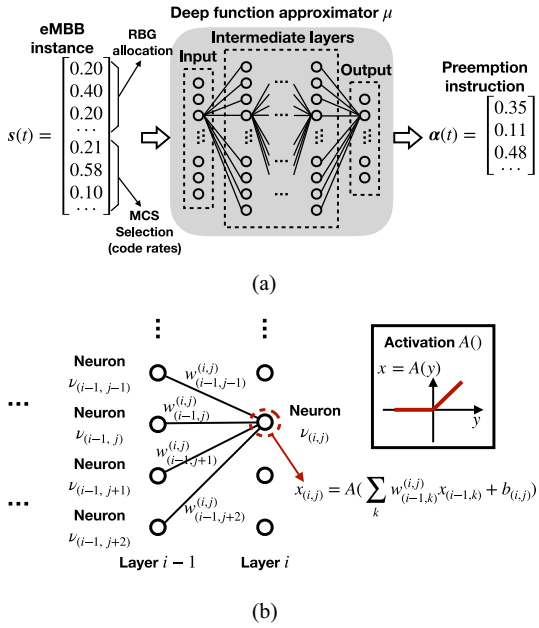


Fig. 3. Deep function approximator μ. (a) Layered structure of μ. (b) Forward propagation between two adjacent layers.

B. Approximate Algorithm μ

The approximate algorithm μ in the scheduling plane is a neural network consisting of an input layer, a number of intermediate layers, and an output layer. An illustration of μ is given in Fig. 3. μ contains a large set of trainable parameters, which we denote as Θ. These parameters are optimized by the learning plane as illustrated in Fig. 2.

Input and Output: At the beginning of each TTI, the input to μ is an eMBB instance vector $\boldsymbol{s}$, as described in the example in Section VI-A. In general, $\boldsymbol{s}$ can be expressed as

$$\boldsymbol{s} = [\boldsymbol{\gamma} \;\; \boldsymbol{\delta}] \tag{1}$$

where $\boldsymbol{\gamma}$ is the normalized SC allocation vector (i.e., percentage of SCs allocated to each eMBB user) and $\boldsymbol{\delta}$ is the MCS selection vector (i.e., MCS selection for each eMBB user). In NR, a resource block group (RBG) is the minimum frequency resolution for eMBB resource allocation [23]. Each RBG consists of a number of contiguous resource blocks (RBs), while an RB contains 12 contiguous SCs. Let M^{RBG} denote the total number of RBGs on the channel. In each TTI, an RBG can be allocated to at most one eMBB user. Thus, the maximum number of eMBB users that can be scheduled per TTI is M^{RBG}. For example, in Section VI-A, there are $M^{\text{RBG}} = 5$ RBGs on the channel with each RBG containing 20 SCs. To ensure a uniform input format for each TTI, we fix the lengths of both $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ to M^{RBG}. Thus, the size of the input vector $\boldsymbol{s}$ is

$2M^{\mathrm{RBG}}$. When the number of scheduled eMBB users is less than M^{RBG}, we can pad the unspecified elements in $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ to null (zero).

The output of μ is a vector $\boldsymbol{\alpha}$ with M^{RBG} elements. The softmax function is used before the output layer $\boldsymbol{\alpha}$.[3] Then, each element of $\boldsymbol{\alpha}$ is a real number between 0 and 1, representing the proportion of SCs required by a URLLC transmission that should be preempted from the corresponding eMBB user. Denote L_{URLLC} as the number of SCs required by a URLLC transmission.[4] The sum of entries in $\boldsymbol{\alpha}$ is equal to 1, meaning that the number of SCs preempted from all the eMBB users is equal to L_{URLLC}.

Intermediate Layers: Between the input and output layers of μ, we have a multilayer structure that is fully connected between two adjacent layers. Such a structure can offer a strong function approximating capability, which is what we need for μ to approximate an optimal algorithm π. Under this structure, each intermediate layer consists of a number of "neurons." A neuron within a layer is fully connected to all neurons in the next layer (as shown in Fig. 3(a)). Forward propagation from input to output between two adjacent layers is shown in Fig. 3(b). For example, consider the jth neuron in the ith layer of μ, which we denote as $v_{(i,j)}$. Its input is the weighted sum of output of all neurons in the $(i-1)$th layer: $\sum_k w_{(i-1,k)}^{(i,j)} x_{(i-1,k)}$, where $x_{(i-1,k)}$ is the output of the neuron $v_{(i-1,k)}$ and $w_{(i-1,k)}^{(i,j)}$ is the weight on the connection from the neuron $v_{(i-1,k)}$ to the neuron $v_{(i,j)}$. Then, the output of this neuron is: $x_{(i,j)} = A(\sum_k w_{(i-1,k)}^{(i,j)} x_{(i-1,k)} + b_{(i,j)})$, where $b_{(i,j)}$ is the bias of the neuron $v_{(i,j)}$ and $A(\cdot)$ is a nonlinear activation function (e.g., rectified nonlinearity). Weights ws and biases bs are included in Θ and optimized through the learning plane.

Under the real-time requirement in NR, the aggregate processing time of μ, g, and f must be no more than the duration of a TTI. To meet this requirement, we must avoid using too many intermediate layers in μ since forward propagation time of μ increases with the number of layers. In our implementation of DEMUX in Section VIII, in order to satisfy the requirement for NR Numerology 0 and 1 (with slot duration of 1 and 0.5 ms, respectively, most suitable for eMBB), a four-layer structure (with two intermediate layers) is used for μ to achieve a forward propagation time of 0.276 ms with an Intel Xeon E5-2687W v4 CPU. For NR Numerology 2 and 3 (with slot duration of 0.25 and 0.125 ms, respectively), an additional mechanism (e.g., GPU [37]) for speedup would be necessary.

C. Guaranteeing Feasibility With g and f

Infeasibility: An issue with approximate algorithm μ is that its output $\boldsymbol{\alpha}$ is unconstrained and may not be a feasible solution for the URLLC preemption (as shown in the example in Section VI-A). Such infeasibility may come from two sources.

1) $\boldsymbol{\alpha}$ may have nonzero elements corresponding to null elements in $\boldsymbol{\gamma}$.
2) $\boldsymbol{\alpha}$ may ask for more SCs to preempt than what the corresponding eMBB users' SCs can offer.

Next, we present a post-processor g and a translator f to address these potential infeasibility issues, respectively.

Post-Processor g: The goal of the post-processor g is to address the first infeasibility issue. To do this, g performs the following operations on $\boldsymbol{\alpha}$. First, we use a mask vector of size M^{RBG} that has the same number of null entries as $\boldsymbol{\gamma}$ and ones elsewhere. We use this mask vector to nullify those "infeasible" entries in $\boldsymbol{\alpha}$.

Also, if there is any entry in $\boldsymbol{\alpha}$ that equals to zero while the same entry in $\boldsymbol{\gamma}$ is nonzero, we will add a very small positive disturbance to this entry in $\boldsymbol{\alpha}$. This operation has minimal impact on the final integral preemption solution but is required by the translator f (to be explained later). Finally, we normalize the sum of nonzero elements from the previous steps in g to 1. Through the processing of g, we obtain a new preemption instruction $\tilde{\boldsymbol{\alpha}}$.

Translator f: But $\tilde{\boldsymbol{\alpha}}$ may still be infeasible, due to the second infeasibility issue. The goal of translator f is to address the second infeasibility so that we can get a final feasible preemption solution $\boldsymbol{\beta}$. Given a resource allocation vector $\boldsymbol{\gamma}$ for the eMBB users, a feasible $\boldsymbol{\beta}$ should ensure that the number of SCs preempted from each eMBB user is no greater than the total number of SCs allocated to this user. That is, $\boldsymbol{\beta}$ must satisfy $L_{\mathrm{URLLC}}\beta_n \le M^{\mathrm{SC}}\gamma_n$ for each scheduled eMBB user n, where M^{SC} is the total number of SCs on the channel, and β_n and γ_n are the nth entries of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, respectively. Then, we have the following feasibility constraints:

$$\beta_n \le \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}}, \quad n = 1, 2, \ldots, M^{\mathrm{RBG}}. \tag{2}$$

Per our discussion about the second infeasibility issue, $\tilde{\boldsymbol{\alpha}}$ (output of g) may not satisfy constraints (2). That is, there might be some entry $\tilde{\alpha}_n$ with $\tilde{\alpha}_n > \gamma_n(M^{\mathrm{SC}})/(L_{\mathrm{URLLC}})$. In fact, by experiments, we found that such a violation is very common. To address this infeasibility, we propose a novel translator function f as follows.

Consider an approach where SCs are preempted from the eMBB users randomly using $\boldsymbol{\beta}$ as a probability distribution (instead of fixed percentages). It is easy to see that in terms of the expected number of SCs preempted from each eMBB user, this probabilistic approach and fixed percentage approach are equivalent. Therefore, we can view $\tilde{\boldsymbol{\alpha}}$ as a probability distribution for random preemption. Following this approach (i.e., treating $\tilde{\boldsymbol{\alpha}}$ and $\boldsymbol{\beta}$ as two probability distributions), a plausible way to construct a feasible $\boldsymbol{\beta}$ from the infeasible $\tilde{\boldsymbol{\alpha}}$ is to minimize the *distance* between the two. In this regard, we can use the KL divergence as the measure of the distance between the two probability distributions [28] and minimize this distance.

Denote $D^{\mathrm{KL}}(\cdot\|\cdot)$ as the KL divergence between two probability distributions. Then, the KL divergence between $\tilde{\boldsymbol{\alpha}}$ and $\boldsymbol{\beta}$ can be defined by[5]

$$D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}\|\boldsymbol{\beta}) = \sum_n \tilde{\alpha}_n \log \frac{\tilde{\alpha}_n}{\beta_n}. \tag{3}$$

[3]For a given vector $\boldsymbol{x} \in \mathbb{R}^n$, the softmax function is defined as $\mathrm{softmax}(\boldsymbol{x})_i = e^{x_i}/\sum_{j=1}^n e^{x_j}$ for $i = 1, 2, \ldots, n$ [32]. Clearly, we have $\mathrm{softmax}(\boldsymbol{x})_i \in (0, 1)$ and $\sum_{i=1}^n \mathrm{softmax}(\boldsymbol{x})_i = 1$.

[4]L_{URLLC} is determined by the packet size (in number of bits), the MCS used for URLLC, and the number of OFDM symbols per mini-slot. For example, for a 50-B packet size, QPSK modulation, LDPC with 1/3 code rate, and two OFDM symbols per mini-slot, we have $L_{\mathrm{URLLC}} = 300$.

[5]We follow the convention that $0\log(0/0) = 0$, $0\log(0/c) = 0$ and $c\log(c/0) = +\infty$, where c is a positive constant.

In the information theory, the KL divergence is also called the relative entropy and is widely used for quantifying the loss of information from one probability distribution to the other probability distribution [29], [30]. Here, we will use (3) to measure how much information is lost if we use $\boldsymbol{\beta}$ to approximate $\tilde{\boldsymbol{\alpha}}$.

It is easy to show that $D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}\|\boldsymbol{\beta}) \geq 0$. Then, the role of the translator f is to find $\boldsymbol{\beta}$ that minimizes the KL divergence with respect to $\tilde{\boldsymbol{\alpha}}$ while meeting the feasibility constraints (2). Denote $\boldsymbol{\beta}^* = \{\beta_1^*, \beta_2^*, \ldots\}$ as the optimal solution found by f. To determine $\boldsymbol{\beta}^*$, f performs the following steps.

1) Denote $\mathcal{N}^0 = \{n|\tilde{\alpha}_n = 0, n = 1, 2, \ldots, M^{\mathrm{RBG}}\}$, i.e., the set of indices of the null elements in $\tilde{\boldsymbol{\alpha}}$. The first step is to set $\beta_n^* = 0$ for all $n \in \mathcal{N}^0$. This operation ensures optimality with respect to the KL divergence according to (3).
2) Denote $\mathcal{N}^+ = \{n|\tilde{\alpha}_n > 0, n = 1, 2, \ldots, M^{\mathrm{RBG}}\}$, i.e., the set of indices of nonzero entries in $\tilde{\boldsymbol{\alpha}}$. For all $n \in \mathcal{N}^+$, we determine the optimal β_n^*s by solving the following optimization problem:

$$\begin{aligned} &\textbf{OPT-}\boldsymbol{\beta} \\ &\text{minimize} \quad D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}\|\boldsymbol{\beta}) \\ &\text{subject to} \quad \beta_n \leq \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}}, \quad n \in \mathcal{N}^+ \\ &\qquad\qquad \beta_n \geq 0, \quad n \in \mathcal{N}^+ \\ &\qquad\qquad \sum_{n\in\mathcal{N}^+} \beta_n = 1. \end{aligned}$$

In OPT-β, $\tilde{\alpha}_n$s and γ_ns are constant inputs. Due to the masking and small positive disturbance applied to $\tilde{\boldsymbol{\alpha}}$ by the post-processor g, we have $\tilde{\alpha}_n > 0$ and $\gamma_n > 0$ for all $n \in \mathcal{N}^+$, and $\sum_{n\in\mathcal{N}^+} \gamma_n = 1$. Then, the optimal solution to OPT-β can be obtained by using Algorithm 1. The optimality of Algorithm 1 can be proved by checking the KKT conditions since OPT-β is a convex optimization problem. Algorithm 1 has a computational complexity of $O(M^{\mathrm{RBG}})$ and requires no more than M^{RBG} iterations to determine $\boldsymbol{\beta}^*$. Although OPT-β can be solved using standard convex optimizers, our Algorithm 1 uses a far fewer number of iterations and thus is more suitable for real-time execution.

Optimal solution $\boldsymbol{\beta}^*$ will be used for actual URLLC preemption in the current TTI.

Algorithm 1 Optimal Solution to OPT-β

Initialize $\mathcal{N} := \emptyset$, $\tilde{\mathcal{N}} := \mathcal{N}^+$, $\psi := 1$;
while $\tilde{\mathcal{N}} \neq \emptyset$ **do**
  Determine $\mathcal{N}' := \left\{ n \in \tilde{\mathcal{N}} \middle| \tilde{\alpha}_n/\psi > \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}} \right\}$;
  if $\mathcal{N}' = \emptyset$ **then**
    for each n in $\tilde{\mathcal{N}}$ **do**
      $\beta_n^* := \tilde{\alpha}_n/\psi$;
    end for
    $\tilde{\mathcal{N}} := \emptyset$;
  else
    for each n in $\mathcal{N}'$ **do**
      $\beta_n^* := \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}}$;
    end for
    $\tilde{\mathcal{N}} := \tilde{\mathcal{N}} \backslash \mathcal{N}'$;
    $\mathcal{N} := \mathcal{N} \cup \mathcal{N}'$;
    $\psi := \sum_{n\in\tilde{\mathcal{N}}} \tilde{\alpha}_n \Big/ \left(1 - \sum_{n\in\mathcal{N}} \gamma_n \frac{M^{\mathrm{SC}}}{L_{\mathrm{URLLC}}}\right)$
  end if
end while
return $\boldsymbol{\beta}^*$;

VII. Design of the Learning Plane

The goal of the learning plane is to optimize the approximate algorithm μ using a DRL-based approach. This section presents our design of the learning plane.

A. Challenge and Proposed Approach

Currently, the most widely used DRL method to address resource allocation problems in wireless networks is the so-called DQL [34]–[36]. DQL was developed for problems with *discrete* action space with a small number of possible solutions. In each learning iteration, DQL attempts to find an action that has the maximum expected return among all possible actions. That is, DQL requires solving a maximization problem over a discrete action domain in each iteration.

Unfortunately, DQL cannot be used to solve our URLLC preemption problem. The main reason is that our space of all possible preemption solutions is prohibitively large. L_{URLLC} is typically on the order of hundreds (see footnote 3) and there is a prohibitively large number of different ways to preempt SCs from the eMBB users in each TTI. For example, consider that there are ten eMBB users scheduled in a TTI. For a URLLC packet requiring 300 SCs, the total number of preemption solutions is $\binom{300+10-1}{10-1} \approx 6.3 \times 10^{16}$ (including infeasible solutions).[6] It is simply impractical to apply DQL to such an enormous discrete action space as the computational time to solve all the maximization problems (one for each learning iteration) is prohibitively large.

To mitigate this problem, we employ a different approach. Instead of using (exact) integer numbers for the SC preemption from the eMBB users, as described in Section VI, we use a vector $\boldsymbol{\beta}$ whose entries are fractions within $[0, 1]$ to indicate what *proportion* of SCs for a URLLC transmission should be preempted from each eMBB user. Since L_{URLLC} is large, the space of all possible $\boldsymbol{\beta}$s is "dense." For instance, for $L_{\mathrm{URLLC}} = 300$, the ℓ^2-norm gap between two adjacent $\boldsymbol{\beta}$s (with difference of one SC allocation) is $\sqrt{2(1/300)^2} = 4.7 \times 10^{-3}$. With this level of closeness, we can consider the space for all $\boldsymbol{\beta}$s as approximately continuous. For any solution within this approximately continuous space, we can easily find a feasible integral preemption action by adjusting a small number of SCs. This prompts us to consider the DDPG method for learning algorithm μ, which turns out to be an excellent choice for this problem. This is because unlike DQL, DDPG was designed to address problems with large and continuous action domains [26], [27]. In each learning iteration, instead of solving a maximization problem as in DQL, DDPG uses a deep function approximator μ to find

[6] Finding a solution for preempting L_{URLLC} SCs from N eMBB users is equivalent to determining a nonnegative integer solution to the equation: $\sum_i^N x_i = L_{\mathrm{URLLC}}$. It has been shown that the total number of such solutions is $\binom{L_{\mathrm{URLLC}}+N-1}{N-1}$ [38, Proposition 6.2].

the preemption action. In the next section, we offer essential background on DDPG, laying the foundation for our own contributions in Section VII-D.

B. DDPG in Nutshell

DDPG is a model-free DRL method that was developed to solve problems with high-dimensional continuous action domains. DDPG is based on a key notion of "actor" and "critic." An actor is a deep function approximator that takes an observed input instance and outputs an action. A critic is another deep function approximator that predicts the expected return of an action under a given input instance. The role of the critic is to offer an *approximate* objective function of the underlying optimization problem in each learning iteration. The ultimate objective of DDPG is to obtain an optimal actor algorithm (neural network) that is able to maximize the expected sum of discounted rewards starting from the first iteration, i.e., $E[\sum_{t=1}^{T} d^{t-1} r(t)]$, where $d \in [0, 1]$ is a discount factor and T represents the total number of iterations.

More formally, denote Q as the critic function approximator. Q is used for predicting the following expected return in each iteration:

$$Q(\boldsymbol{s}(t), \boldsymbol{\alpha}(t)) = E\left[\sum_{i=t}^{T} d^{i-t} r(i) \Big| \boldsymbol{s}(t), \boldsymbol{\alpha}(t)\right]. \tag{4}$$

Equation (4) is called the "action value" in DDPG. It represents the expected return of taking the action $\boldsymbol{\alpha}(t)$ under the input instance $\boldsymbol{s}(t)$, assuming that the current actor algorithm will be used for all future iterations.

The learning process of DDPG is to optimize parameters in actor and critic neural networks through a large number of iterations. Denote Φ as the set of trainable parameters in Q (weights and biases similar to those in Θ). In each iteration, the actor takes action for a given input instance. Then, a reward signal is received for this action. This action execution instance is then stored in a replay buffer. Next, a subset of K stored execution instances are randomly sampled from the replay buffer. Using these samples, the critic network Q is updated based on received reward signals to improve its prediction accuracy. Then, the updated Q is used to optimize the actor. The parameters within the actor network are updated by taking a small step along the gradient of Q with respect to these parameters.

To ensure the learning can converge eventually (i.e., obtain a stable and maximized return), DDPG employs two additional neural networks, called *target networks* to help stabilize the learning of Q. Specifically, based on the Bellman equation [27], the action value in (4) can be rewritten as

$$Q(\boldsymbol{s}(t), \boldsymbol{\alpha}(t)) = E\Big[r(t) + d \cdot Q(\boldsymbol{s}(t+1), \boldsymbol{\alpha}(t+1)) \Big| \boldsymbol{s}(t), \boldsymbol{\alpha}(t)\Big]. \tag{5}$$

The use of target networks is to approximate (substitute) $Q(\boldsymbol{s}(t+1), \boldsymbol{\alpha}(t+1))$ in (5), i.e., the action value in iteration $t+1$. Then, the sum of the reward $r(t)$ for the current iteration and the action value for the next iteration predicted by target networks (discounted by d) provides a "target" for updating the critic network Q. By using target networks, the update of Q in each learning iteration becomes more stable. The downside of using target networks, however, is that it slows down learning, as target networks are constrained to have a very small update in each iteration (also known as, "soft" update using a coefficient $\tau \ll 1$ [27]).

C. Convergence Problem

For our URLLC preemption problem, we find that if we follow DDPG as it is, then the learning is extremely slow to converge. In our experiments, we found that there was no significant improvement in the objective (i.e., reducing the loss of eMBB utility) after more than half a million iterations. The reason behind this is that the learning for the critic Q is too slow. The accuracy of Q for predicting the action value in (4) does not show much improvement as the number of learning iterations increases. Without an accurate critic Q, DDPG cannot converge in learning the algorithm μ. Furthermore, the inclusion of target networks in DDPG also slows down the learning progress since target networks are constrained to be updated very slowly using a coefficient $\tau \ll 1$. In the next section, we present our design to mitigate this convergence problem.

D. Our Design

Our design of the learning plane is shown in Fig. 2. It retains an actor–critic structure similar to DDPG (with the actor being neural network μ described in Section VI-B and the critic being another neural network Q) but with some significant differences.

1) Based on our observation of some intrinsic properties associated with the URLLC preemption problem, we propose to simplify the learning objective of DDPG and the action value that the critic Q needs to predict. We show that these simplifications can achieve a much faster and more stable convergence compared to original DDPG.
2) Following the above simplifications, we propose to remove the target networks in DDPG from our design. This will eliminate the delay associated with target networks but will not hamper the stability of the learning process.

The rest of this section is organized as follows. We first describe our design of the reward function, which serves as an input to the learning plane. Then, we describe how to design a learning method that can speed up convergence. A summary of the proposed learning method is given in Algorithm 2.

Construction of the Reward Function: The reward function in each TTI is a component of the action value in (4) that the critic Q needs to approximate. There are two aspects that must be considered when constructing a reward function for our problem.

1) The reward function must offer a critical assessment (evaluation) for a preemption instruction $\boldsymbol{\alpha}(t)$ in terms of its impact on eMBB utility.
2) The reward function should include a penalty for any infeasibility incurred in μ's output $\boldsymbol{\alpha}(t)$.

For the first consideration (critical assessment), we need to determine the loss of eMBB utility in a TTI due to the

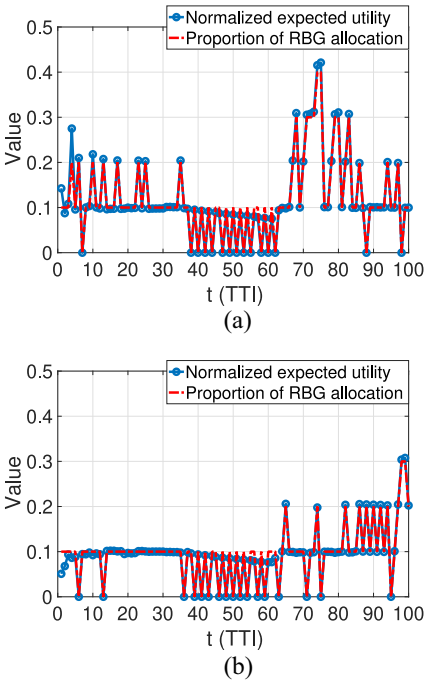


Fig. 4. Relationship between the proportion of RBGs allocated to an eMBB user and its normalized expected PF utility. (a) eMBB user 1. (b) eMBB user 2.

URLLC preemption. Denote $U_n(t)$ as the utility that eMBB user n will obtain if its received transmission in TTI t can be successfully decoded. Denote $U(t)$ as the maximum utility that can be achieved across all the eMBB users scheduled in TTI t. Then, we have

$$U(t) = \sum_n U_n(t).$$

If all eMBB transmissions are successful, then the maximum $U(t)$ is achieved. But in the presence of URLLC preemption, some eMBB user(s) may not be able to decode its received signal, resulting in a loss of eMBB utility.

Denote $\boldsymbol{\epsilon}(t)$ as an indication vector for whether or not each scheduled eMBB transmission succeeds, i.e., for each eMBB user n

$$\epsilon_n(t) = \begin{cases} 0, & \text{if transmission to eMBB user } n \text{ succeeds} \\ -1, & \text{if transmission to eMBB user } n \text{ fails.} \end{cases}$$

Then, the normalized loss of eMBB utility in TTI t is given by $\sum_n \epsilon_n(t) U_n(t)/U(t)$.

Denote $r^{\mathrm{LU}}(t)$ as the reward for preserving eMBB utility in TTI t, which we define as

$$r^{\mathrm{LU}}(t) = \sum_n \epsilon_n(t) \frac{U_n(t)}{U(t)}.$$

Then, the range of $r^{\mathrm{LU}}(t)$ is between $[-1, 0]$. For example, if all eMBB transmissions are successful, then $r^{\mathrm{LU}}(t) = 0$; if all eMBB transmissions fail, then $r^{\mathrm{LU}}(t) = -1$.

If we were to use this reward function, then we would have to add additional information ($U_n(t)/U(t)$s) to the input of μ. This is undesirable as it requires to add more intermediate layers in μ, which will result in a longer execution time. Instead, we find that we can obtain a good approximation for $r^{\mathrm{LU}}(t)$ by using information already included in the input $\boldsymbol{s}(t)$. Through experiments, we found that the normalized expected utility $U_n(t)/U(t)$ of an eMBB user is highly correlated with the proportion of RBGs allocated to it, i.e., $\gamma_n(t)$. The reason is that a user with a higher expected utility is more likely to be allocated with more RBGs. This correlation has been verified by our experimental results shown in Fig. 4, where we run a PHY–MAC NR simulator (see Section VIII-A) with PF scheduling for ten eMBB users and show the results for the first and second users for 100 TTIs. In Fig. 4, we can see that $\gamma_n(t)$ has a nearly perfect match to $U_n(t)/U(t)$. Based on this insight, we propose to use

$$r^{\mathrm{LU}}(t) = -\sum_n \epsilon_n(t)\gamma_n(t) \tag{6}$$

as a reward function. This innovative design allows us to avoid bringing any additional input to μ.

For the second consideration (i.e., to penalize infeasibilites in μ's output $\boldsymbol{\alpha}(t)$), we propose to include penalties for both the redundant elements in $\boldsymbol{\alpha}(t)$ and the KL divergence $D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}(t)\|\boldsymbol{\beta}^*(t))$. That is, for redundant elements in $\boldsymbol{\alpha}(t)$, we introduce a penalty

$$r^{\mathrm{RE}}(t) = -\sum_{n \in \mathcal{N}^0(t)} \alpha_n(t). \tag{7}$$

For the KL divergence $D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}(t)\|\boldsymbol{\beta}^*(t))$, we use the penalty

$$r^{\mathrm{KL}}(t) = -D^{\mathrm{KL}}(\tilde{\boldsymbol{\alpha}}(t)\|\boldsymbol{\beta}^*(t)). \tag{8}$$

The goal of (7) and (8) is to reduce the distance between the algorithm μ's output $\boldsymbol{\alpha}(t)$ and the final feasible preemption solution $\boldsymbol{\beta}^*(t)$ through the learning process. This design helps minimize the loss of information caused by g and f.

Putting (6)–(8) together, we construct the complete reward function $r(t)$ as follows:

$$r(t) = w^{\mathrm{LU}} r^{\mathrm{LU}}(t) + w^{\mathrm{RE}} r^{\mathrm{RE}}(t) + w^{\mathrm{KL}} r^{\mathrm{KL}}(t) \tag{9}$$

where w^{LU}, w^{RE}, and w^{KL} are positive weights. These weights reflect relative importance of $r^{\mathrm{LU}}(t)$, $r^{\mathrm{RE}}(t)$, and $r^{\mathrm{KL}}(t)$ in the reward function $r(t)$.

Mechanisms to Accelerate Convergence: To address the convergence problem, we take a closer look at our URLLC preemption problem. The objective of our problem is to minimize the loss of eMBB utility due to the URLLC preemption. To characterize eMBB utility, let us consider some of the most-widely used schedulers for eMBB, such as PF [15]–[19], maximum throughput (MT), and other weighted-rate-based schedulers [18]. A common characteristic of these eMBB schedulers is that they are mostly greedy algorithms that aim to maximize the eMBB utility in each TTI independently. An attractive feature of these greedy algorithms is that they have low computational complexities and thus are suitable for real-time execution. In this article, we consider the PF scheduler [15]–[18] as an example. The same design of the learning method also applies to other greedy eMBB schedulers.

The PF scheduler is known for its ability to strike a tradeoff between fairness and total cell throughput among the users (i.e., avoiding starvation at cell edge). The utility function of

PF is: $\sum_n \log \bar{C}_n$, where $\bar{C}_n$ is the long-term average data rate of the user n. An asymptotically optimal solution to the PF scheduling is to allocate RBGs in each TTI t with the following objective:

$$\text{maximize} \quad U(t) = \sum_n \frac{\hat{C}_n(t)}{\bar{C}_n(t-1)} \tag{10}$$

where $\hat{C}_n(t)$ is the expected data rate to the user n in TTI t, and $\bar{C}_n(t-1)$ is the user n's exponentially smoothed average data rate up to TTI $(t-1)$ and is updated as

$$\bar{C}_n(t-1) = (1-\eta)\bar{C}_n(t-2) + \eta C_n(t-1) \tag{11}$$

where $C_n(t-1)$ is user n's achieved data rate in TTI $(t-1)$, and η is a small positive coefficient (e.g., 0.01). It has been shown that this per-TTI scheduling maximizes PF utility when $t \to \infty$ [19].

For our URLLC preemption problem, to minimize the loss of eMBB utility, it is sufficient to minimize the loss of utility $U(t)$ in each TTI. Following the same proof for per-TTI scheduling (10), it is easy to show that this approach is asymptotically optimal for minimizing the loss of eMBB utility. Note that this property holds for the MT scheduler and other weighted-rate-based schedulers as well. For these schedulers, maximizing per-TTI utility metrics is globally optimal.

Now, the question to ask is: how can we exploit this property to design a learning method? Our first observation is that the optimization objective of the learning plane can be reduced to: maximize $E[r(t)]$, i.e., only maximizing the expected reward in each TTI without accounting for rewards in future TTIs. As described earlier, a major component of $r(t)$ is the reward for preserving the per-TTI utility $U(t)$. Thus, to minimize the loss of eMBB utility, it is sufficient to maximize $E[r(t)]$ for each TTI. Then, the action value in (4) is simplified to

$$Q(\boldsymbol{s}(t), \boldsymbol{\alpha}(t)) = E\Big[r(t)\Big|\boldsymbol{s}(t), \boldsymbol{\alpha}(t)\Big]. \tag{12}$$

These simplifications accelerate convergence on the learning plane in two aspects. First, the learning for the critic Q becomes much easier and faster because Q is no longer required to predict the expected return in future TTIs. Second, there is no need to include target networks since action value (12) only involves the expected reward in the current TTI. Moreover, the removal of target networks will not impact the stability of learning.

Learning Method: Algorithm 2 summarizes our proposed learning method based on the action value in (12). For initialization, parameters in neural networks μ and Q are randomly generated. Then, the memory is allocated for the replay buffer. At the beginning of learning, an initial eMBB instance $\boldsymbol{s}(1)$ for TTI $t = 1$ is received. Then, the learning process of Algorithm 2 goes through a *for* loop with a total number of T iterations (lines 4–13). In each iteration, an algorithm execution instance $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t), r(t))$ is obtained from the scheduling plane and stored in the replay buffer (lines 5–9). Then, a subset of K samples are randomly selected from the replay buffer. These samples are used for updating parameters in the critic network Q and the algorithm μ. Q is updated by solving

Algorithm 2 Method for Learning Algorithm μ

1: Initialize parameters (randomly) in Θ and Φ for algorithm μ and critic network Q, respectively;
2: Initialize replay buffer;
3: Receive initial eMBB instance $\boldsymbol{s}(1)$;
4: **for** each TTI $t = 1, 2, \cdots, T$ **do**
5: Obtain preemption instruction from μ: $\boldsymbol{\alpha}(t) \leftarrow \mu(\boldsymbol{s}(t)|\Theta)$;
6: Apply post-processor and obtain: $\tilde{\boldsymbol{\alpha}}(t) \leftarrow g(\boldsymbol{\alpha}(t))$;
7: Obtain preemption solution using translator: $\boldsymbol{\beta}(t) \leftarrow f(\tilde{\boldsymbol{\alpha}}(t), \boldsymbol{s}(t))$;
8: Use $\boldsymbol{\beta}(t)$ for all URLLC transmissions within TTI t, receive reward $r(t)$, and observe new eMBB instance $\boldsymbol{s}(t+1)$;
9: Store the result $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t), r(t))$ into replay buffer;
10: Randomly sample a subset of K results $(\boldsymbol{s}(i), \boldsymbol{\alpha}(i), r(i))$ from replay buffer;
11: Update Q's parameters in Φ by solving (13);
12: Update μ's parameters in Θ using the sample gradient (14);
13: **end for**
14: **return** Learned approximate algorithm μ;

$$\text{minimize} \quad \frac{1}{K}\sum_{i=1}^{K}(r(i) - Q(\boldsymbol{s}(i), \boldsymbol{\alpha}(i)|\Phi))^2. \tag{13}$$

An efficient implementation of (13) is to update parameters in Q (Φ) by taking a small step along the gradient of the minimization objective with respect to Q's parameters. Note that in (13), the update of Q is only based on rewards $r(i)$s without any target network. This corresponds to our simplification for action value in (12). After updating Q, parameters in the algorithm μ (Θ) are updated using the following sample gradient (SG):

$$\text{SG} = \frac{1}{K}\sum_{i=1}^{K} \nabla_{\boldsymbol{\alpha}} Q(\boldsymbol{s}, \boldsymbol{\alpha}|\Phi)|_{\boldsymbol{s}=\boldsymbol{s}(i), \boldsymbol{\alpha}=\mu(\boldsymbol{s}(i))} \nabla_{\Theta} \mu(\boldsymbol{s}|\Theta)|_{\boldsymbol{s}(i)}. \tag{14}$$

When the *for* loop completes, a learned algorithm μ is returned.

Stability of Learning: From (13), we can see that the learning of action value in (12) is very similar to supervised learning. Theoretically, the neural network Q is used to approximate the expected reward $E[r(t)|\boldsymbol{s}(t), \boldsymbol{\alpha}(t)]$ in each TTI t. In practical implementation, we use the reward $r(t)$ as an unbiased estimation for $E[r(t)|\boldsymbol{s}(t), \boldsymbol{\alpha}(t)]$. During the learning process, network Q is trained with tuples $(\boldsymbol{s}(i), \boldsymbol{\alpha}(i), r(i))$ for a large number of TTIs (i.e., (13)). The goal is to have a network Q that can precisely predict expected reward $E[r(t)]$ given an input instance $(\boldsymbol{s}(t), \boldsymbol{\alpha}(t))$. Such learning of Q resembles a supervised learning process (with a large amount of training data $(\boldsymbol{s}(i), \boldsymbol{\alpha}(i), r(i))$). Moreover, since the learning target for the network Q is simply $r(t)$ and Q is not used to calculate the target value, our design eliminates the divergence/stability issue in DDPG.

VIII. Implementation

This section presents our implementation of DEMUX in a simulated 5G NR network environment. Our implementation has the following two components.

1) A PHY–MAC NR simulator supporting dynamic eMBB/URLLC multiplexing with URLLC preemption.
2) A DRL module implementing our learning method in Section VII-D.

TABLE II
MCS LEVELS AND MEASURED WORKING SNRS (dB) UNDER BLER = 0.1

MCS level	1	2	3	4	5	6	7	8	9	10
Modulation	QPSK				16QAM				64QAM	
Code rate	0.10	0.21	0.33	0.42	0.29	0.39	0.49	0.58	0.46	0.55
Working SNR (dB)	-1.4	1.3	4.2	6.1	9.6	11.8	14.0	16.2	18.3	21.0

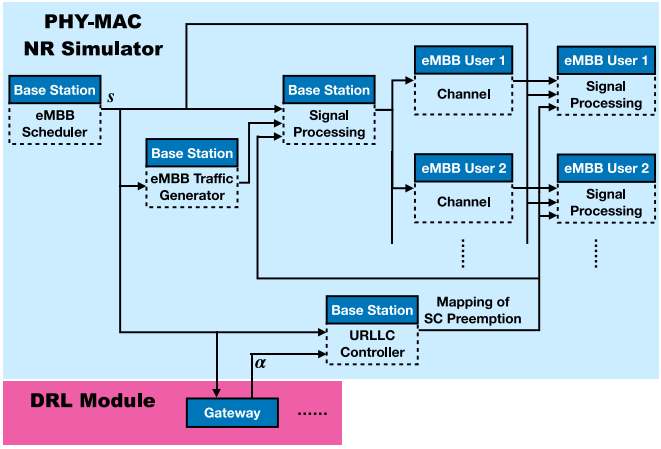


Fig. 5. Implementation diagram of our NR simulator.

We employ object-oriented programming to implement both components. Each component involves a number of classes. Procedures within the closed-loop across scheduling and learning planes (see Fig. 2) are implemented as different methods encapsulated in these classes. In the rest of this section, we give details of our implementation.

A. PHY–MAC NR Simulator

Our NR simulator implements NR PHY and MAC layer functions that are most relevant to the URLLC preemption problem. This simulator is built using MathWorks' 5G Toolbox [14], which offers standard-compliant signal processing functions for 5G NR end-to-end communications, including channel coding, modulation, OFDM modulation, and multipath channel fading. In addition to these signal processing functions, we implement modules to support the SC preemption for URLLC packet transmissions, simultaneous multilink eMBB transmissions, RBG scheduling, and MCS selection for the eMBB users, and TCP/IP socket-based communication with the DRL module.

Fig. 5 illustrates our implementation of the NR simulator. As shown in this figure, the NR simulator has a BS class and an *eMBB user* class. The BS class contains methods for per-TTI downlink processing procedures, including eMBB scheduling, URLLC control, data generation, and transmitter-side signal processing. The eMBB user class contains methods for channel fading and noise generation and receiver-side signal processing. A simulated NR cell instance consists of an object of the BS class and multiple objects of the eMBB user class. This characterizes multiple communication links between the BS and the eMBB users.

The BS class has the following methods.

1) *eMBB Scheduler:* This method schedules eMBB transmissions in each TTI and assigns RBGs and MCS level for each transmission. For RBG allocation, we employ the PF scheduler described in Section VII-D. In each TTI, RBGs are allocated iteratively to the eMBB users based on a *copy* of long-term average rates $\bar{C}_n(t)$s that is updated after allocating each RBG as in (11). The original $\bar{C}_n(t)$s are updated after all RBGs are allocated in each TTI. For eMBB MCS selection, we consider ten MCS levels given in Table II. Each MCS level requires a minimum working SNR to ensure that the BLER is no greater than a target threshold. The working SNRs for 0.1 BLER (typical value for eMBB) of these MCS levels are measured through experiments using our NR simulator. The measurement is done for the cluster delay line (CDL)-C multipath channel fading with 300 ns delay spread on 25 RBs. For each eMBB user, the BS selects an MCS level so that the user's average SNR is no less than the MCS's working SNR. This method returns an eMBB instance vector $\boldsymbol{s}(t)$. In particular, $\boldsymbol{s}(t)$ will be sent to the DRL module via Gateway. The Learning Agent will use $\boldsymbol{s}(t)$ to produce a preemption instruction $\boldsymbol{\alpha}(t)$ based on the algorithm μ.
2) *URLLC Controller:* This method controls URLLC transmissions and preemptive puncturing in each TTI. The input to this method includes eMBB instance $\boldsymbol{s}(t)$ and puncturing instruction $\boldsymbol{\alpha}(t)$ from the DRL module. The URLLC controller performs the following tasks. First, it obtains a feasible preemption solution $\boldsymbol{\beta}(t) = (f \circ g)(\boldsymbol{\alpha}(t))$ using g and f as described in Section VI-C. Second, URLLC packets are generated using a Poisson process with arrival rate λ (arrivals/TTI). λ is a configurable parameter of the simulator. Arrived URLLC packets are stored in a first-in–first-out buffer. Third, each URLLC packet is scheduled to transmit in the earliest available mini-slot after its arrival. In each mini-slot, at most one URLLC packet is transmitted. Fourth, the preemption of SCs for each URLLC transmission is determined based on the solution $\boldsymbol{\beta}(t)$. As discussed in Section II, the preemption on each eMBB transmission proceeds from the SC with the lowest index that is allocated to it. Finally, a frequency-time resource mapping for preemption is generated, which indicates OFDM symbols and SCs that are punctured by URLLC in the current TTI. The preemption mapping is returned as the output of this method.
3) *eMBB Traffic Generator:* This method generates data bits for scheduled eMBB transmissions. For an eMBB

transmission, the number of data bits within its transport block (TB) depends on the number of allocated RBGs and assigned MCS level. In general, a TB contains more data bits with more RBGs and a higher MCS level. The data bits within each TB are first generated as a random binary sequence. Then, cyclic redundancy check (CRC) bits are computed and attached to TBs. These CRC bits will be used for checking whether each eMBB transmission succeeds.

4) *Transmitter-Side Signal Processing:* This method implements the downlink signal processing chain at the BS. Input to this method includes eMBB instance $\boldsymbol{s}(t)$, preemption mapping from the URLLC controller method and TBs generated by the eMBB traffic generator. The processing procedures include LDPC encoding, rate matching, scrambling, modulation (QPSK, 16QAM, or 64QAM), resource preemption, and OFDM modulation. After modulation, eMBB symbols on preempted resources are flushed out according to the preemption mapping. In each TTI, this method processes TBs for all scheduled eMBB transmissions.

The eMBB user class has the following methods.

1) *Channel:* This method generates frequency and time domain channel fading and noise for an eMBB transmission. It is called by an eMBB user object before user-side signal processing. Input to this method is an eMBB transmission after BS-side signal processing. Practical multipath channel fading models, such as the CDL model and the tap delay line (TDL) model [13] are employed in this method.
2) *Receiver-Side Signal Processing:* This method implements a signal processing chain at the eMBB user side. The input to this method includes $\boldsymbol{s}(t)$, preemption mapping, and returned eMBB transmission from the channel method. The processing procedures include OFDM demodulation, soft demodulation, preemption flushing, descrambling, rate recovery, and LDPC decoding. The operation of preemption flushing is to nullify corrupted bits after soft demodulation according to the preemption mapping. This operation prevents error propagation during LDPC decoding. After decoding, the CRC checks results indicate whether each eMBB transmission is successful.

B. DRL Module

We implement the DRL module using TensorFlow version 1.13.1 [31]. The DRL module involves a *Gateway* class and a *Learning Agent* class. The Gateway class offers a communication interface between the NR simulator and the DRL module using TCP/IP socket. During our experiment, the transfer of eMBB instance $\boldsymbol{s}(t)$ and reward $r(t)$ from the NR simulator to the DRL module and the transfer of preemption instruction $\boldsymbol{\alpha}(t)$ from the DRL module to the NR simulator all go through an object of this class.

The Learning Agent class implements our learning method in Section VII-D. Both neural networks μ and Q are contained in this class. Tasks for this class include: 1) learning μ using Algorithm 2 and 2) executing learned μ with the input instance $\boldsymbol{s}(t)$ received via Gateway. It has the following methods.

1) *Model Setup:* This method sets up the neural networks μ and Q, optimizers used for updating parameters in μ and Q, and replay buffer for storing algorithm execution instances. The input to this method is a configuration profile specifying the numbers of layers and neurons per layer, nonlinearity for each neuron, input and output formats for networks μ and Q, and other configuration settings. The optimizers for Q and μ first compute gradients of Q with respect to parameters in Θ and Φ, respectively, and then update the parameters in Q and μ along the gradients.
2) *Learner:* This method implements Algorithm 2 for learning algorithm μ. We employ a very flexible implementation of the learning method. In each learning iteration, the algorithm μ is executed for a number of consecutive TTIs. Then, the parameters in Q and μ are updated multiple times using different subsets of samples. Settings in this implementation (i.e., how many algorithm executions and parameter updates per iteration) can be tuned through the experiment to achieve an optimized learning performance.
3) *Actor:* This method takes an eMBB instance $\boldsymbol{s}(t)$ as input (obtained via Gateway) and uses learned algorithm μ to produce a preemption instruction $\boldsymbol{\alpha}(t)$, which is then sent to the NR simulator through Gateway.

C. Putting Our Implementation to Work

We now describe how to use our NR simulator and DRL module for learning algorithm μ. The first step is to set up an experiment instance consisting of an object of the BS class, a number of objects of the eMBB user class, an object of the Gateway class, and an object of the Learning Agent class. Since in each TTI at most M^{RBG} eMBB users can be scheduled (see Section VI-B), we only need to generate M^{RBG} eMBB user objects in the experiment instance. To fully explore the space of all possible eMBB instances $\boldsymbol{s}(t)$s, we randomly allocate RBGs on the channel to the eMBB users in each TTI. The MCS level for each eMBB user is also randomly selected from Table II with average received SNRs equal to working SNRs of the selected MCS levels. Thus, every possible eMBB instance $\boldsymbol{s}(t)$ has the same probability to appear during learning, leading to a full exploration of input space for μ.

Then, we run the learning process shown in Fig. 2 for T iterations (e.g., on the order of 10^5 or higher). During learning, we turn off multipath channel fading and noise effects in the NR simulator (in eMBB user objects). The reason is that with fading and noise, an eMBB transmission could fail even without the URLLC preemption, making it hard to deduce whether or not a transmission failure is due to the underlying preemption solution. Our experiment results in Section IX show that algorithms learned under this approach perform well in all network scenarios when different channel fading models are present.

Another issue for learning is that with the Poisson packet arrivals, the number of URLLC transmissions in each TTI is a

TABLE III
KEY PARAMETER SETTINGS IN THE NR SIMULATOR

NR numerology	Numerology 1 (30 kHz SC spacing)
System bandwidth	20 MHz (50 RBs in total, 5 RBs/RBG)
Carrier frequency	4 GHz
Cell radius	1000 m
eMBB config.	1 TTI = 1 time slot (14 OFDM symbols) Full buffer traffic model
URLLC config.	1 mini-slot = 2 OFDM symbols Poisson arrival of 50-byte packets
Fading channel	CDL-C/TDL-C with 300 ns RMS delay spread [13]
Pathloss model	3D UMa NLOS [13]
Channel estimation	Ideal channel estimation
Antenna config.	1 Tx antenna and 1 Rx antenna
BS Tx power	46 dBm
Noise floor	-91.9 dBm

random number between zero and the number of mini-slots per time slot. Thus, eMBB transmission failure not only depends on the preemption solution $\boldsymbol{\beta}(t)$ but also on the number of URLLC transmissions in a TTI. This again makes it difficult to learn the quality of a preemption solution. We propose the following approach to addressing this issue. In realistic cells, the URLLC packet arrival rate λ always has a finite range. Assume that there are N mini-slots per TTI. The BS should perform proper URLLC traffic control to prevent overflow of URLLC packets, e.g., ensure that no more than N packets will arrive in each TTI. Then, the range of λ is $[0, N]$. Based on this knowledge, one can uniformly sample a set of discrete arrival rates from $[0, N]$ (e.g., $\{1, 2, \ldots, N\}$) and learn a separate actor network μ offline under each sampled arrival rate. In the learning phase, we fix the number of URLLC transmissions in each TTI to the given arrival rate. Since each network μ is trained under a given *known* arrival rate, it does not need to learn the packet arriving pattern. The sole objective of learning is to let μ approximate the optimal puncturing solution under the given arrival rate.

During cell operating time, the first task for the BS is to estimate the URLLC packet arrival rate empirically from the data collected in the cell. This estimation can be done through statistical methods and is independent of the learning of actor networks μs. Then, the BS scheduler can choose a network μ (learned under a known arrival rate) that is closest to the cell's actual URLLC traffic condition. The chosen network will perform well if it matches the estimated arrival rate.

Therefore, we do not need to explicitly include the arrival rate into the input layer of the network μ since a separate network is learned under a given arrival rate. Moreover, there is no need to include the arrival rate in the reward function.

IX. VALIDATION

This section validates the performance of DEMUX based on our implementation in Section VIII. All experiments are done on a Dell desktop computer with an Intel Xeon E5-2687W v4 CPU (3.0 GHz).

A. Experimental Settings

NR Simulator: We use our NR simulator to set up a downlink NR cell environment with a BS, a number of eMBB users, and a given URLLC traffic load. The key parameter settings of the cell environment are given in Table III.

All eMBB users are uniformly and randomly distributed within the cell's coverage. The Poisson URLLC packet arrival rate is chosen from $\{2, 3, 4, 5, 6\}$. For eMBB PF scheduling, the coefficient η used for updating long-term average rates $\bar{C}_n(t)$s in (11) is set to 0.01. The MCS level for each eMBB user is selected by finding the highest level in Table II that can be supported by the user's average received SNR. The MCS selection vector $\boldsymbol{\delta}(t)$ is set to code rates of eMBB transmissions. This ensures all entries of the vector $\boldsymbol{s}(t)$ (μ's input) are within [0, 1]. We employ QPSK modulation with a 1/3 code rate for URLLC transmissions [4]. Under this settings, we have $L_{\text{URLLC}} = 300$. All eMBB and URLLC transmissions are with single transmit and receive antenna and one code word.

We consider two multipath channel fading models—TDL and CDL, that are widely used for link-level simulations. TDL involves power, delay, and the Doppler spectrum information for each multipath channel tap. CDL is a spatial extension of TDL that includes more detailed information such as angle for each cluster of channel taps.

DRL Module: For the actor neural network μ, we employ a feedforward neural network with two fully connected intermediate layers each with 256 neurons. The rectified nonlinearity is used as the activation function for intermediate layers. The input layer is the eMBB instance vector $\boldsymbol{s}(t)$ of size $2M^{\text{RBG}}$. A softmax function is used before the output layer $\boldsymbol{\alpha}(t)$ to ensure that the sum of entries in $\boldsymbol{\alpha}(t)$ equals to one. To ensure a broader exploration of the action space, we add a small random noise to $\boldsymbol{\alpha}(t)$. This action noise is assumed to have the Gaussian distribution with zero mean and a standard deviation of 0.02. Then, all entries in $\boldsymbol{\alpha}(t)$ are clipped to ensure they are within [0, 1] and normalized afterward.

The critic network Q is also a feedforward neural network with the same intermediate layer structure and activation as μ. The eMBB instance vector $\boldsymbol{s}(t)$ is the input to the first intermediate layer. The output $\boldsymbol{\alpha}(t)$ of μ is an additional input to the second intermediate layer. Q's output is a single unconstrained neuron that predicts the action value. We use the Adam algorithm [33] for learning parameters in Q and μ with a learning rate of 10^{-3} per iteration. A subset size $K = 128$ for random sampling and replay buffer size of 10^5 are employed.

B. Benchmark Comparison

For performance comparison, we implement two other state-of-the-art URLLC preemption schemes on the same NR simulator platform. The first is the RP algorithm proposed in [10]. For each URLLC transmission, RP preempts SCs from each eMBB user in proportion to its RBG allocation, i.e., the preemption solution is $\boldsymbol{\beta}^{\text{RP}}(t) = \boldsymbol{\gamma}(t)$. It was shown in [10] that

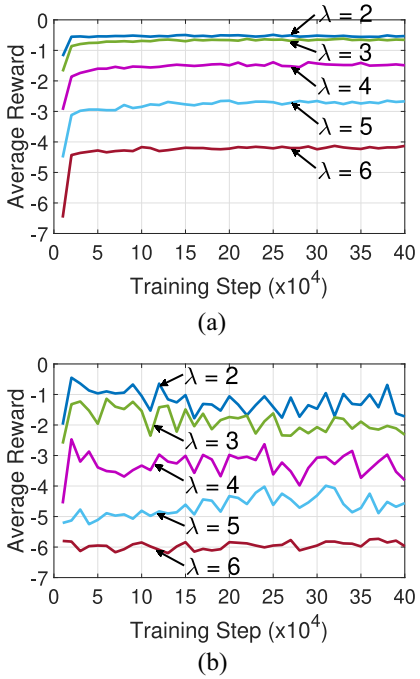


Fig. 6. Learning curves of (a) DEMUX and (b) original DDPG under different URLLC packet arrival rates.

RP is optimal if the cost function of the URLLC preemption is linear.[7]

The second benchmark scheme is FFP that was proposed in 3GPP standards body [8], [9]. This scheme divides the channel bandwidth into multiple nonoverlapping and fixed frequency parts. All URLLC packets are transmitted using one specified frequency part. Under our setting $M^{\mathrm{SC}}/L_{\mathrm{URLLC}} = 2$, the channel bandwidth is divided into two parts and all URLLC transmissions will use the first part.

C. Results

Convergence of Learning: We use our implementation in Section VIII to learn an algorithm μ for each URLLC arrival rate $\lambda \in \{2, 3, 4, 5, 6\}$. Weights for the three components in the reward function (9) are set to $w^{\mathrm{LU}} = 10$, $w^{\mathrm{RE}} = 3$, and $w^{\mathrm{KL}} = 5$. Other settings are also possible while w^{LU} should generally be greater than w^{RE} and w^{KL} due to the significance of $r^{\mathrm{LU}}(t)$ in the reward function. Fig. 6(a) shows the learning curves of DEMUX over 4×10^5 steps under different arrival rates. Each point on a curve represents the reward value averaged over the past 100 TTIs. We can see that the learning processes of DEMUX converge very quickly and smoothly within 5×10^4.

As a comparison, Fig. 6(b) shows the learning curves of the original DDPG (refer to Section VII-B) under the same reward function setting ($w^{\mathrm{LU}} = 10$, $w^{\mathrm{RE}} = 3$, and $w^{\mathrm{KL}} = 5$). In contrast to DEMUX, the learning curves of DDPG do not converge even after 4×10^5 steps and the average rewards fluctuate significantly as the number of steps increases. As we pointed out in Section VII-C, the main reason is that DDPG cannot effectively learn an accurate critic Q for our URLLC preemption problem.

[7]Anand *et al.* [10] also proposed an algorithm for convex cost function. But this algorithm requires an explicit closed-form cost function, which is not available in practice.

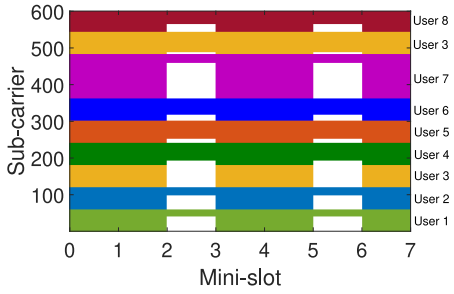


Fig. 7. Example of URLLC preemption by DEMUX. Resources allocated to each eMBB user are represented by a different color. Resources preempted by URLLC are in white blank.

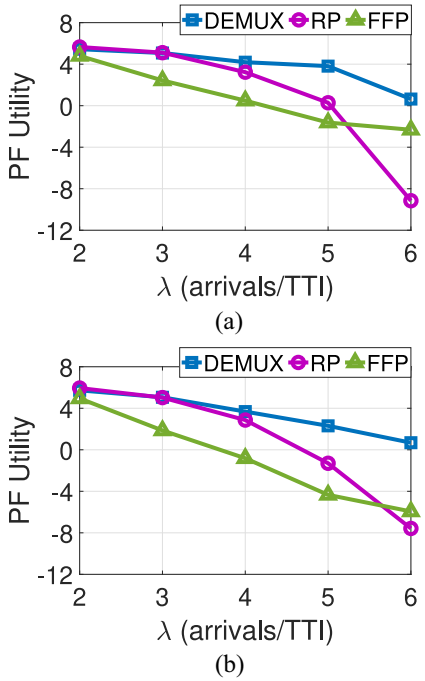


Fig. 8. Sum of utility comparison for ten eMBB users under (a) CDL and (b) TDL channel models.

Comparison of Performance Objectives: In the following experiments, we compare the performance of DEMUX, RP, and FFP. We consider two performance objectives.

1) *eMBB Users' Sum PF Utility:* $\sum_n \log \bar{C}_n$, which is our optimization objective (i.e., minimizing the loss of eMBB PF utility).
2) *eMBB Cell Throughput:* $\sum_n \bar{C}_n$, which is an important performance metric that is of major concern to cellular operators.

We show results for 10 and 30 eMBB users, which are typically used for performance evaluation in 3GPP standards body (e.g., [2, Table 6.1.2-1] for dense urban scenario).

10 eMBB Users: We first run experiments with ten eMBB users under different URLLC traffic loads and channel fading models. Each preemption scheme (DEMUX, RP, or FFP) is executed for 10^4 consecutive TTIs under a given URLLC arrival rate $\lambda \in \{2, 3, 4, 5, 6\}$ and a fading model.

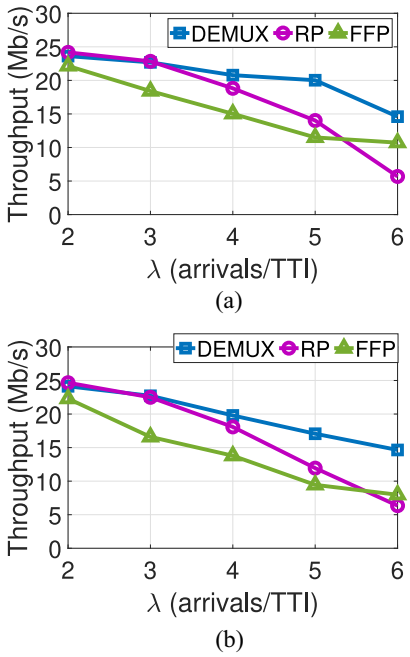


Fig. 9. Cell throughput comparison for ten eMBB users under (a) CDL and (b) TDL channel models.

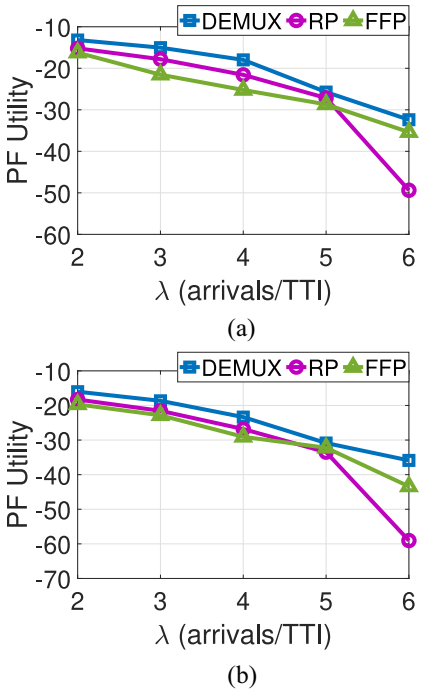


Fig. 10. Sum of utility comparison for 30 eMBB users under (a) CDL and (b) TDL channel models.

To understand how URLLC preemption works, in Fig. 7, we provide a live example. Following Table III, a mini-slot contains two OFDM symbols and a time slot consists of seven mini-slots. As shown in Fig. 7, there are eight eMBB transmissions scheduled (out of ten users) in this TTI. The numbers of SCs allocated to the eight eMBB users are 60, 60, 120, 60, 60, 60, 120, and 60, respectively. There are two URLLC transmissions occurring at the 3rd and 6th mini-slots, respectively. Under DEMUX, the numbers of SCs preempted from the eight eMBB users are 43, 37, 66, 9, 8, 15, 96, and 26, respectively. In contrast (not shown in Fig. 7), FFP preempts SCs from 0 to 300 while for RP, the numbers of SCs preempted from the eight eMBB users are 30, 30, 60, 30, 30, 30, 60, and 30, respectively.

In Fig. 8(a) and (b), we compare the sum eMBB utility ($\sum_n \log \bar{C}_n$) achieved by the three schemes under CDL and TDL channel fading models, respectively. As expected, the sum utility decreases as the URLLC arrival rate increases under each scheme. Among the three schemes, DEMUX offers the best performance. When URLLC traffic load is low ($\lambda = 2$ and 3), the performance of DEMUX and RP is comparable but still better than FFP. The performance gap between DEMUX and the other two schemes widens when the URLLC traffic load increases.

Fig. 9(a) and (b) shows the eMBB cell throughput ($\sum_n \bar{C}_n$) achieved by the three schemes under CDL and TDL fading models, respectively. Again, DEMUX has the best performance among the three and the performance gap between DEMUX and the other two schemes widens as arrival rate increases. Specifically, the largest performance gains of DEMUX over FFP are 76% under CDL and 81% under TDL (when $\lambda = 5$), respectively. The largest gains of DEMUX over RP are 157% under CDL and 131% under TDL (when $\lambda = 6$), respectively.

30 eMBB Users: Next, we run experiments with 30 eMBB users under the same settings. Fig. 10(a) and (b) shows the sum eMBB utility achieved by DEMUX, RP, and FFP under CDL and TDL fading models, respectively. We can see that DEMUX outperforms the other two schemes under all URLLC traffic loads and fading models.

The total cell throughput performance of the three schemes under CDL and TDL fading models is shown in Fig. 11(a) and (b), respectively. Again, DEMUX achieves the highest throughput among the three. In particular, the largest gains of DEMUX over RP are 75% under CDL and 121% under TDL (when $\lambda = 6$), respectively. The largest gains of DEMUX over FFP are 27% under CDL and 21% under TDL (when $\lambda = 4$), respectively.

The fundamental reason that DEMUX can outperform RP and FFP is that DEMUX is designed to learn an optimal solution to the URLLC preemption problem, while RP and FFP are heuristics that one could only hope to find "good" solutions. Our results presented above demonstrate that DEMUX can successfully accomplish its goal and achieve superior performance than RP and FFP.

Computation Time: To check whether or not DEMUX can meet the requirement for real-time scheduling, it is only necessary to examine its execution time on the scheduling plane, which consists of the forward propagation time of neural network μ, and computation time of post-processor g and translator f. The execution time of DEMUX is evaluated using an Intel Xeon E5-2687W v4 CPU and the results are shown in Table IV. First, we find that our CPU-based implementation of DEMUX meets the real-time requirement under NR Numerology 0 and 1, where time slot duration is 1 and 0.5 ms, respectively. For Numerology 2 and 3 with

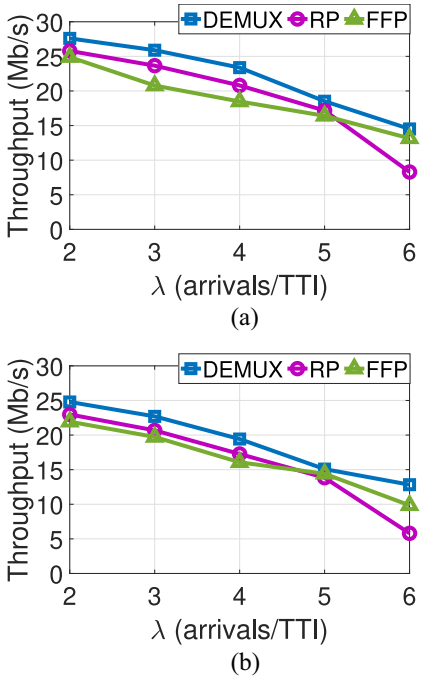


Fig. 11. Cell throughput comparison for 30 eMBB users under (a) CDL and (b) TDL channel models.

TABLE IV
COMPUTATION TIME OF DEMUX. RESULTS ARE AVERAGED OVER 10^5 TTIS

# of eMBB users	μ (ms)	$f \circ g$ (ms)	Total (ms)
10	0.276	0.088	0.364
30	0.269	0.092	0.361

even shorter time slot duration (250 and 125 μs), one could employ GPU-based implementation (e.g., the cuDNN platform from NVIDIA [37]) to accelerate forward propagation of neural network μ. Second, we find that the computation time of DEMUX does not increase as the number of eMBB users grows. The reasons are that: 1) forward propagation time of μ only depends on the numbers of layers and neurons per layer in its structure and 2) execution time of g and f is determined by the maximum number of eMBB users that can be scheduled per TTI (i.e., M^{RBG}) and is independent of the total number of users in the cell.

X. CONCLUSION

In this article, we investigated dynamic eMBB/URLLC multiplexing via a preemptive puncturing mechanism that was proposed in 3GPP standards body. We studied an important optimization problem under this mechanism on how to spread SCs preempted by each URLLC transmission across the eMBB users so that the adverse impact on eMBB is minimized. A major technical challenge is that this problem cannot be solved using traditional model-based optimization approaches. To address this challenge, we proposed DEMUX—a novel model-free DRL-based solution. Key contributions in the design of DEMUX include the following.

1) DEMUX is able to learn an optimal algorithm for URLLC preemption using deep function approximators without the need for a prior explicit problem formulation.
2) DEMUX is the first known design that employs the DRL method with a large and continuous action domain for enabling dynamic eMBB/URLLC multiplexing.
3) To achieve fast and stable convergence of learning, we proposed to augment the DDPG method by adapting the learning objective and removing additional target networks based on intrinsic properties of the URLLC preemption problem.
4) To ensure feasibility, we proposed a novel approach based on the KL divergence to converting an unconstrained output of a neural network into a feasible URLLC preemption solution.
5) For implementation, we built an experiment platform consisting of a PHY–MAC NR simulator and a DRL learning module with a scalable object-oriented design that can be used for validating the performance of DEMUX and other benchmark schemes.
6) Through extensive experiments using our platform, we showed that DEMUX significantly outperforms heuristic algorithms proposed in the 3GPP standards body and the literature (up to 130% performance gain in test network scenarios). Furthermore, we showed that DEMUX meets the timing requirement for real-time scheduling in NR.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their feedback.

REFERENCES

[1] Qualcomm. (2018). *Making 5G NR a Commercial Reality*. [Online]. Available: https://www.qualcomm.com/media/documents/files/the-3gpp-release-15-5g-nr-design.pdf
[2] "Study on scenarios and requirements for next generation access technologies, version 15.0.0," 3GPP, Sophia Antipolis, France, Rep. TR 38.913, 2018. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996
[3] "Minimum requirements related to technical performance for IMT-2020 radio interface(s)," ITU, Geneva, Switzerland, Rep. M.2410-0, 2017. [Online]. Available: https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2410-2017-PDF-E.pdf
[4] "Study on physical layer enhancements for NR ultra-reliable and low latency case (URLLC), version 16.0.0," 3GPP, Sophia Antipolis, France, Rep. TR 38.824, 2019. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3498
[5] "Study on new radio access technology; radio interface protocol aspects, version 14.0.0," 3GPP, Sophia Antipolis, France, Rep. TR 38.804, 2017. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3070
[6] (Feb. 2017). *Final Report of 3GPP TSG RAN WG1 Meeting #88 Version 1.0.0, Athens, Greece*. [Online]. Available: https://www.3gpp.org/ftp/TSG_RAN/WG1_RL1/TSGR1_88/Report/
[7] C. P. Li, J. Jiang, W. Chen, T. Ji, and J. Smee, "5G ultra-reliable and low-latency systems design," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2017, pp. 1–5.
[8] (Nov/Dec. 2017). *Final Report of 3GPP TSG RAN WG1 Meeting #91 Version 1.0.0, Reno, NV, USA*. [Online]. Available: https://www.3gpp.org/ftp/tsg_ran/WG1_RL1/TSGR1_91/Report/
[9] *HiSilicon: Remaining Aspects on Pre-Emption Indication for DL Multiplexing of URLLC and eMBB*, document 3GPP TSG RAN WG1 Meeting #91, R1-1721452, Huawei, Reno, NV, USA, Nov./Dec. 2017.
[10] A. Anand, G. De Veciana, and S. Shakkottai, "Joint scheduling of URLLC and eMBB traffic in 5G wireless networks," in *Proc. IEEE INFOCOM*, Honolulu, HI, USA, Apr. 2018, pp. 1970–1978.

[11] A. K. Bairagi, M. S. Munir, M. Alsenwi, N. H. Tran, and C. S. Hong, "A matching based coexistence mechanism between eMBB and uRLLC in 5G wireless networks," in *Proc. ACM/SIGAPP Symp. Appl. Comput. (SAC)*, Limassol, Cyprus, Apr. 2019, pp. 2377–2384.

[12] J. Zhang, X. Xu, K. Zhang, B. Zhang, X. Tao, and P. Zhang, "Machine learning based flexible transmission time interval scheduling for eMBB and uRLLC coexistence scenario," *IEEE Access*, vol. 7, pp. 65811–65820, 2019.

[13] "Study on channel model for frequencies from 0.5 to 100 GHz, version 15.0.0," 3GPP, Sophia Antipolis, France, Rep. TR 38.901, 2018. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3173

[14] MathWorks. (2019). *5G Toolbox Documentation*. [Online]. Available: https://www.mathworks.com/help/5g/

[15] D. Tse, "Multiuser diversity in wireless networks: Smart scheduling, dumb antennas and epidemic communication," in *Proc. IMA Workshop Wireless Netw.*, 2001. [Online]. Available: https://web.stanford.edu/ dntse/papers/ima810.pdf

[16] S. B. Lee, S. Choudhury, A. Khoshnevis, S. Xu, and S. Lu, "Downlink MIMO with frequency-domain packet scheduling for 3GPP LTE," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 1269–1277.

[17] Y. Huang, S. Li, Y. T. Hou, and W. Lou, "GPF: A GPU-based design to achieve $\sim$100 μs scheduling for 5G NR," in *Proc. ACM MobiCom*, Oct./Nov. 2018, pp. 207–222.

[18] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in LTE cellular networks: Key design issues and a survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 678–700, 2nd Quart., 2013.

[19] A. Stolyar, "On the asymptotic optimality of the gradient scheduling algorithm for multi-user throughput allocation," *Oper. Res.*, vol. 53, pp. 12–25, Feb. 2005.

[20] *NR; Physical Channels and Modulation, Version 15.5.0*, 3GPP Standard TS 38.211, 2019. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3213

[21] *NR; Multiplexing and Channel Coding, Version 15.5.0*, 3GPP Standard TS 38.212, 2019. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3214

[22] *NR; Physical Layer Procedures for Control, Version 15.5.0*, 3GPP Standard TS 38.213, 2019. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3215

[23] *NR; Physical Layer Procedures for Data, Version 15.5.0*, 3GPP Standard TS 38.214, 2019. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3216

[24] T. Richardson and S. Kudekar, "Design of low-density parity check codes for 5G new radio," *IEEE Commun. Mag.*, vol. 56, no. 3, pp. 28–34, Mar. 2018.

[25] P. S. Rybin, "On the error-correcting capabilities of low-complexity decoded irregular LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun./Jul. 2014, pp. 3165–3169.

[26] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 387–395.

[27] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[28] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, 1951.

[29] M. Xie, J. Hu, S. Guo, and A. Y. Zomaya, "Distributed segment-based anomaly detection with Kullback–Leibler divergence in wireless sensor networks," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 1, pp. 101–110, Jan. 2017.

[30] Y. Bu, S. Zou, Y. Liang, and V. V. Veeravalli, "Estimation of KL divergence: Optimal minimax rate," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 2648–2674, Apr. 2018.

[31] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016. [Online]. Available: arXiv:1603.04467.

[32] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015.

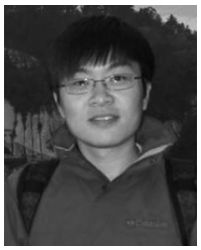
[34] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Trans. Wireless Commun.*, vol. 18, no. 1, pp. 310–323, Jan. 2019.

[35] T. He, N. Zhao, and H. Yin, "Integrated networking, caching and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[36] J. Zhu, Y. Song, D. Jiang, and H. Song, "A new deep-Q-learning-based transmission scheduling mechanism for the cognitive Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2375–2385, Aug. 2018.

[37] NVIDIA. (2019). *cuDNN Developer Guide V7.6.3*. [Online]. Available: https://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/index.html

[38] S. Ross, *A First Course in Probability, Chapter 1*, 8th ed. London, U.K.: Pearson, 2009.

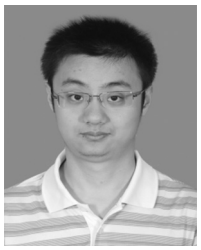


Yan Huang (Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with Virginia Tech, Blacksburg, VA, USA.

His research interests are GPU-based real-time optimizations for wireless networks and machine learning for communications.

Shaoran Li (Student Member, IEEE) received the B.S. degree from Southeast University, Nanjing, China, in 2014, and the M.S. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2017. He is currently pursuing the Ph.D. degree with Virginia Tech, Blacksburg, VA, USA.

His research interests include algorithm design and implementation for wireless networks.



Chengzhang Li (Student Member, IEEE) received the B.S. degree in electronics engineering from Tsinghua University, Beijing, China, in 2017. He is currently pursuing the Ph.D. degree with the Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA.

His current research interests are modeling, analysis, and algorithm design for wireless networks, with a focus on age of information and latency research.

Y. Thomas Hou (Fellow, IEEE) received the Ph.D. degree from the NYU Tandon School of Engineering, New York, NY, USA, in 1998.

In 2002, he joined Virginia Tech, Blacksburg, VA, USA, where he is a Bradley Distinguished Professor of electrical and computer engineering. From 1997 to 2002, he was a member of research staff with the Fujitsu Laboratories of America, Sunnyvale, CA, USA. He has over 300 papers published in IEEE/ACM journals and conferences. He has authored/coauthored two graduate textbooks: *Applied Optimization Methods for Wireless Networks* (Cambridge University Press, 2014) and *Cognitive Radio Communications and Networks: Principles and Practices* (Academic Press/Elsevier, 2009). He holds five U.S. patents. His current research focuses on developing innovative solutions to complex science and engineering problems arising from wireless and mobile networks and wireless security.

Prof. Hou received eight best paper awards from IEEE and ACM. He was/is on the editorial boards of a number of IEEE and ACM transactions and journals. He served as the Steering Committee Chair for IEEE INFOCOM Conference. He was also a Distinguished Lecturer of the IEEE Communications Society. He was a member of the IEEE Communications Society Board of Governors. He was named an IEEE Fellow for contributions to modeling and optimization of wireless networks.

Wenjing Lou (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Florida, Gainesville, FL, USA.

He is the W. C. English Endowed Professor of computer science with Virginia Tech, Blacksburg, VA, USA. Her research interests cover many topics in the cybersecurity field, with her current research interest focusing on blockchain, privacy protection in machine learning systems, and security and privacy problems in the Internet-of-Things systems.

Prof. Lou received the Virginia Tech Alumni Award for Research Excellence in 2018, which is the highest university level faculty research award. She received the INFOCOM Test-of-Time Paper Award in 2020. She is a Highly Cited Researcher by the Web of Science Group. She is a TPC Chair for IEEE INFOCOM 2020 and ACM WiSec 2020. She is the Steering Committee Chair of IEEE CNS conference series and the Steering committee Member of IEEE INFOCOM Conference and the IEEE TRANSACTIONS ON MOBILE COMPUTING. She served as a Program Director with the U.S. National Science Foundation from 2014 to 2017.