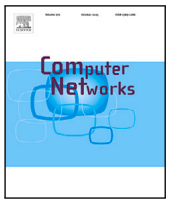
Distributional Reinforcement Learning for Task Offloading, Resource Allocation and Early Exit Selection at the Edge

Computer Networks 288 (2026) 112652

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Distributional Reinforcement Learning for task offloading, resource allocation and early exit selection at the edge

Simone Angelucci [a,b,*], Roberto Valentini [a,b], Marco Levorato [c], Fortunato Santucci [a,b], Carla Fabiana Chiasserini [d]

[a] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy
[b] Centre Ex-EMERGE, L'Aquila, Italy
[c] Donald Bren School of Information and Computer Science, University of California Irvine, Irvine, United States
[d] Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Italy

ARTICLE INFO

Keywords:
Early exiting
Edge computing
Mobile edge applications
Connected and automated vehicles
Distributional reinforcement learning
Markov decision processes

ABSTRACT

Early Exiting (EE) is an emerging paradigm in deep learning that equips Deep Neural Networks (DNNs) with intermediate classifiers, enabling a trade-off between inference accuracy and latency. In this work, we investigate the integration of EE mechanisms into edge computing architectures, focusing on a representative use case involving task execution in resource-constrained computing and communications environments for connected and automated vehicles (CAVs). We develop a detailed system model that captures the complex interplay among time-varying system components, including wireless channel coherence and the dynamic availability of computational and communication resources. Building on this model, we formulate a joint optimization problem encompassing task offloading, resource allocation, and early exit selection. We demonstrate how EE enhances system adaptability under stringent constraints, such as limited bandwidth, computing capacity, or delay requirements. To tackle the complexity of the proposed optimization, we adopt a novel solution approach based on the distributional Soft Actor-Critic (SAC) Deep Reinforcement Learning (DRL) algorithm, which quantifies the *uncertainty* of the learned policy. Simulation results confirm that integrating EE with edge computing significantly improves the trade-off between inference accuracy and latency, achieving up to 212% improvement in the average task completion ratio compared to edge computing systems without EE, under the considered simulation settings.

1. Introduction

Connected and Automated Vehicles (CAVs) are expected to be central components of future Intelligent Transportation Systems (ITS): this is clearly evidenced by the numerous application classes and use cases identified by key organizations such as the 3rd Generation Partnership Project (3GPP) and the 5G Automotive Association (5GAA) [1–4]. Many of these use cases rely heavily on Artificial Intelligence (AI), particularly for tasks related to perception and situational awareness.

A prominent example is the use of dynamic local maps [5], which store contextual information about the vehicle's surroundings. These maps support safety-critical applications by providing real-time data on nearby entities, such as pedestrians, vehicles, and obstacles. Populating these maps typically involves computationally demanding AI tasks such as object detection, recognition, and instance segmentation, which rely on Deep Neural Networks (DNNs). However, due to their resource and energy requirements, executing such models directly on CAVs is often impractical.

To address these limitations, edge computing has emerged as a promising solution that enables CAVs to off-load inference tasks to edge servers deployed within the wireless infrastructure. Although this reduces on-board computational load and latency, performance may still be hindered by factors such as network capacity variability and limited availability of edge resources.

To further optimize DNN execution, edge computing can be combined with adaptive inference strategies designed to reduce resource consumption and response time. One such approach is Early Exiting (EE) [6], which modifies the DNN's architecture by inserting intermediate exit points. These allow the model to produce a prediction and terminate execution early as soon as a certain confidence threshold is met. This mechanism not only helps reduce inference latency but also enables flexible trade-offs between accuracy and efficiency based on

* Corresponding author at: Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy.
E-mail address: simone.angelucci@graduate.univaq.it (S. Angelucci).

https://doi.org/10.1016/j.comnet.2026.112652
Received 20 April 2026; Received in revised form 24 July 2026; Accepted 3 August 2026
Available online 7 August 2026

system constraints. As such, EE offers a promising path for adapting AI workloads to dynamic environments while ensuring efficient use of computational resources.

While DNN inference offloading has been extensively studied, the integration of EE mechanisms into edge computing systems has only recently emerged as a research direction. This work belongs to this emerging area and focuses on the joint optimization of early-exit decisions with Task Offloading and Resource Allocation (TORA) decisions, including task offloading and radio and computational resource allocation. Although existing studies have demonstrated the potential of EE to reduce inference latency and computational cost, they typically optimize EE jointly with only a subset of the TORA decision variables or consider simplified system models. Therefore, the potential benefits arising from the interaction between early exit decisions and resource management strategies in realistic, resource constrained edge computing environments remain largely unexplored.

Motivated by these observations, this paper proposes a novel framework that integrates EE mechanisms into edge computing systems and explicitly incorporates them into TORA problem. We consider a CAV scenario with multiple mobile users belonging to heterogeneous service classes and executing time-sensitive DNN inference tasks. Both CAvs and the edge server are equipped with DNN models augmented with early exits. A centralized controller dynamically orchestrates task execution by jointly deciding task offloading, DNN exit point selection, radio resource allocation, and computational resource assignment. Although the considered system is instantiated in a CAV scenario, the focus of this work is not on modeling vehicular mobility or network topology dynamics. Rather, the CAV context is adopted as a representative application scenario characterized by stringent latency constraints, high task arrival rates, and safety-critical inference workloads. These features make CAV networks a particularly suitable testbed for studying EE-aware task offloading and resource allocation under tight performance requirements.

This paper extends our previous work in [7]. While [7] investigated EE in a simplified single-user edge computing scenario with no resource allocation problem, the present work significantly broadens the scope by considering a multi-user TORA framework with joint optimization of early exit selection, task offloading, and radio and computational resource allocation. Moreover, this work introduces a refined radio interface model and adopts a Distributional Reinforcement Learning (DistRL) approach, which were not considered in our previous study.

The main contributions of this work are summarized as follows:

- To the best of our knowledge, we are the first to explicitly formulate and study a TORA problem that jointly optimizes early exit selection, task offloading decisions, and radio and computational resource allocation, named as TORA-EE problem. Unlike existing works that consider EE only in conjunction with a subset of these decisions, our formulation integrates EE selection control variable within the TORA framework, enabling flexible optimization of edge intelligence systems under resource constraints.
- We propose a system model that captures key dynamics often oversimplified in the literature. In particular, we account for temporal correlations in system states through user-specific task queues, allowing current decisions to influence future system evolution. Furthermore, we adopt a refined radio interface model aligned with 5G specifications, incorporating temporally correlated wireless channels, adaptive modulation and coding schemes, and Block Error Rate (BLER) constraints. This modeling approach provides a more faithful representation of practical edge computing systems.
- To solve the resulting TORA-EE problem, we adopt a DistRL approach that models the full distribution of long-term returns rather than only their expected value. This enables the learning of risk-aware control policies that account for critical events such as queue overflows, latency violations, and task drops, by learning conservative policies that reduce performance variability. To the best of our knowledge, this is the first work to apply Distributional RL to resource-aware edge intelligence systems with early exiting mechanisms.
- Through extensive simulations, we demonstrate the benefits of integrating EE within the TORA framework. Results show that EE enables flexible adaptation of inference accuracy and execution latency in response to time-varying channel conditions, fluctuating resource availability, and heterogeneous service requirements, achieving performance gains of up to 212% in terms of successfully completed tasks compared to conventional edge computing architectures.

The remainder of this paper is structured as follows: Section 2 provides an overview of the literature and better clarifies our contributions, Section 3 illustrates the system modeling, Section 4 describes the algorithm we used for solving our problem, in Section 5 we show the results we have obtained, whereas in Section 6 we draw our conclusions and we sketch future directions.

2. Related works

The TORA problem has been extensively studied in the literature. Existing works differ primarily in the proposed solution strategies, which are often closely tied to the underlying modeling assumptions, such as task representation, task management policies, communication models, and resource characterization. Additional differences arise from the specific application scenarios considered and the optimization objectives targeted, which may include metrics such as latency, energy efficiency, reliability, or quality of service. In the following, we provide an overview of the literature according to the main classes of qualifying characteristics, so that our novel contributions can be later claimed.

Reference scenario. The scenarios usually analyzed vary across several dimensions. Some works focus on single-user or multi-user environments [8–11], while others differentiate between single-edge server deployments [8,9] and multi-edge server architectures [11,12]. In addition, several studies incorporate fog and cloud computing nodes into the system model [13–15], recognizing that offloaded computations may not always be executed locally or at the edge due to resource limitations. In the context of Vehicular Edge Computing (VEC), related work [16] explores alternative offloading strategies, such as leveraging neighboring vehicles either as execution nodes or as relays to edge servers, depending on resource availability and network conditions.

Task modeling. Regarding system modeling approaches, tasks are commonly characterized by a tuple of requirements and specifications. Typical task descriptions include parameters such as the maximum allowable execution time, the number of required computing cycles, and the data size to be transmitted to remote servers [9,10,15,17]. Some studies further incorporate attributes like task priority [12], as well as Random Access Memory (RAM) and Graphical Processing Unit (GPU) resource requirements [18].

Resource handling. When computing a task remotely, different resources are needed; in particular, we refer to radio and computational resources. A typical modeling choice is to have fixed total remote resources, or belonging to a predefined set of values, and the decisions relate to how to distribute the resources among the users [8–12,16,17]. Another common option is to assume that radio or computing resources are constant [9]. In such a case, no resource allocation occurs basically, since resources are already pre-allocated, hence the matter is whether execute the tasks locally or offload them. In addition, other studies model resource availability with queues [13], where each resource is represented by a queue representing the amount of work dedicated to that resource. Other common resources instead are Random Access Memory (RAM), storage [8,13,14], and energy, whose usage is usually minimized [9,10].

Radio interface. Task offloading typically relies on a wireless communication channel, whose characteristics are modeled in various ways throughout the literature. A widely adopted approach is to use Shannon's capacity formula to estimate achievable data rates, given the allocated bandwidth and instantaneous wireless channel gain [9,10,12,15–17,19]. In most of these works, channel gains are assumed to follow a Rayleigh fading model and to be temporally independent. In [15], the authors also account for transmission failures at the subcarrier level, especially in scenarios that employ orthogonal frequency division multiplexing (OFDM). In contrast, several studies [8,13,14,20] abstract away physical layer details by assuming fixed transmission times or modeling tasks as being directly enqueued at remote servers, thereby ignoring the wireless transmission dynamics.

Utility functions. The TORA problem also varies across studies in terms of the utility or cost functions targeted for optimization. Commonly considered objectives include maximizing the number of tasks successfully executed [14], minimizing task execution time or overall system latency [17], and reducing latency weighted by task priority [9] or model precision [11]. Other works focus on maximizing the probability of task admission at edge servers [8]. Furthermore, several studies adopt composite metrics, such as joint optimization of task throughput and latency [18], task throughput and energy consumption [9], or latency and energy consumption [15,16], to better reflect trade-offs in system performance.

Solution approaches. The choice of solution strategies for the TORA problem is typically guided by the underlying system modeling. For instance, when the system exhibits Markovian dynamics, Deep Reinforcement Learning (DRL) methods are commonly employed, as demonstrated in [9–11,15]. In contrast, when resource availability is modeled using queue dynamics, and stability is a concern, Lyapunov optimization is often adopted [12–14,18,19]. Alternatively, less commonly used approaches include hybrid methods that combine clustering techniques with convex optimization based on the Karush-Kuhn–Tucker (KKT) conditions [17], bio-inspired heuristics such as Grey Wolf Optimization [16], and submodular optimization frameworks [8].

Early exiting. Recently, EE mechanisms have been introduced as a means to dynamically adapt the execution depth of deep neural networks, enabling a flexible trade-off between inference accuracy and latency. Several works have explored the integration of EE within edge computing systems. For instance, [21,22], and [23] jointly consider early exit selection and model partitioning while optimizing different performance metrics. Other contributions extend the control space to include computational resource assignment [11,24], or additional model-level adaptations such as quantization [25]. Along this line, [26] addresses the joint allocation of edge computing resources and DNN splitting for a multisensor digital twin synchronized over a collaborative edge-cloud architecture, in which the exit point of each constituent task is not selected but determined stochastically by the acquired sensor data. Extending the resource dimension further, [27] formulates a joint communication and computation resource allocation problem for multiuser edge inference, additionally accounting for batch execution to maximize the number of tasks completed per scheduling interval. More recently, [28] jointly optimizes radio and computational resource allocation together with semantic extraction, data compression, and edge server selection for computation task offloading, modeling the radio access as an OFDMA system in which bandwidth allocation is governed by the number of assigned subcarriers.

2.1. Beyond the state of the art

Despite the extensive literature on TORA and the growing interest in early exiting for edge intelligence, several fundamental limitations remain.

First, with respect to problem formulation, the few works that consider EE in edge computing do not address the TORA problem in its entirety. Recent contributions jointly optimize EE with computational resource allocation [26], or with both radio and computational resource allocation [27,28]. However, none of these works formulates a unified TORA problem in which early exit selection is jointly optimized together with task execution and offloading decisions, radio resource allocation, and computational resource assignment. In contrast, this work explicitly incorporates EE into the TORA framework and formulates a unified optimization problem, referred to as TORA-EE, where all these decisions are optimized simultaneously. To the best of our knowledge, this is the first work to address this joint optimization problem under the proposed system model.

Second, concerning system modeling, many existing works rely on oversimplified representations of the wireless interface. A common assumption is that channel realizations are temporally uncorrelated, as in [9,10,12,15–17,19], despite the well-known temporal correlation exhibited by wireless channels. In this paper, we adopt a finite-state channel model that captures temporal channel dynamics, building upon the modeling approach introduced in [7]. Moreover, we refine the radio interface modeling by adopting a data rate formulation aligned with 3GPP specifications. Unlike the classical Shannon capacity expression, which provides an optimistic upper bound, the adopted model accounts for packet overhead and frequency-dependent attenuation. In addition, a BLER constraint is enforced through adaptive Modulation and Coding Scheme (MCS) selection, enabling reliable transmission under time-varying channel conditions.

Furthermore, an aspect that is often overlooked particularly in DRL-based approaches is the temporal correlation among system states. Many works formulate the problem as an MDP with i.i.d. state components, such as task arrivals and available resources [9–11,15]. However, real edge systems exhibit strong temporal dependencies, where current decisions affect future system evolution. To capture these dynamics, we explicitly model user-specific task queues, introducing memory into the system and allowing the impact of suboptimal decisions to propagate over time (e.g., queue buildup and overflows). While queue-based models have appeared in prior work, they are typically addressed via Lyapunov optimization frameworks that focus on time-average performance and stability. In contrast, our approach adopts a reinforcement learning perspective that directly operates on such temporally correlated dynamics.

Finally, from an algorithmic standpoint, this work introduces a novel solution approach based on DistRL. Conventional Deep Reinforcement Learning (DRL) methods optimize only the expected long-term return, which may obscure rare but critical events such as excessive latency or queue overflows. DistRL, by modeling the full distribution of returns, enables a richer characterization of uncertainty and risk in long-term system performance. This property is particularly well suited to the TORA-EE problem, where avoiding tail events is as important as optimizing average metrics. To the best of our knowledge, this is the first work that applies DistRL to a resource-aware edge intelligence framework incorporating early exit mechanisms. We emphasize that the goal of this work is not to propose a new reinforcement learning algorithm nor to establish algorithmic superiority over existing DRL methods. Rather, our objective is to investigate the impact of adopting a DistRL paradigm for the TORA-EE problem, and to assess whether modeling the full return distribution provides tangible benefits over classical expectation-based approaches.

3. System model

We consider the system scenario depicted in Fig. 1, which consists of N_u CAVs that have to perform DNN-based inference tasks, and an edge server connected to a base station. The CAV can either perform the tasks locally or offload them to the edge server. We assume that both the edge server and the CAVs use dynamic neural models equipped with EEs. Also, time is assumed to be slotted, where the time variable t represents time interval $[tL, (t+1)L)$, with L the slot duration. For

Table 1
Symbols and related descriptions.

Symbol	Description
System variables	
N_u	Number of users
$q_n(t)$	Number of backlogged tasks for user n at time t
$\mathbf{q}_n(t)$	User n queue at time t
$Q_{\max}$	Maximum queue capacity
$r_n(t)$	User n wireless channel state
$f_D L$	Wireless channel correlation parameter
$W(t)$	Available radio resources
$C(t)$	Available remote computing resources
C_n	User n local computing resources
Task and application parameters	
$i_n(t)$	Number of incoming tasks for user n
$u_n(t)$	Number of removed tasks for user n
δ_n	User n task data size
τ_n	User n task deadline
$\boldsymbol{\delta}$	Users task data size vector
$\boldsymbol{\tau}$	Users task deadline vector
$\boldsymbol{\epsilon}$	Running applications
ζ_n	User n task generation parameter
$\boldsymbol{\zeta}$	Poisson parameters vector
$\mathbf{c}_\epsilon$	Task ϵ MOP vector
$c_{\epsilon,e}$	MOP needed for exit e for task ϵ
$\boldsymbol{a}_\epsilon$	Task ϵ accuracy vector
$a_{\epsilon,e}$	Accuracy associated to exit e for task ϵ
Execution-related variables	
$T_{\text{off},n}(t)$	Offloading time for user n at time t
$R_{\text{off},n}(t)$	Offloading data rate for user n at time t
$T_{\text{loc},n}(t)$	Local execution time for user n at time t
$T_{\text{rem},n}(t)$	Remote execution time for user n at time t
TORA-EE variables	
$o_n(t)$	Offloading decision for user n
$e_n(t)$	User n early exit decision
$\xi_n(t)$	Decision variable for radio resources
$\rho_n(t)$	Decision variable for computing resources
$w_n(t)$	Radio resources allocated to user n
$c_n(t)$	Remote computing resources allocated to user n
$A_n(e_n(t))$	Accuracy of early exit decision $e_n(t)$
$\boldsymbol{\nu}$	Objective function weights vector
DSAC-T parameters	
θ	Critic network parameter
ϕ	Actor network parameter
$\bar{\theta}$	Target critic network parameter
$\bar{\phi}$	Target actor network parameter
$Q_{\theta_i}(x,a)$	Mean learned by critic θ_i
$\sigma_{\theta_i}(x,a)$	Std. dev. learned by critic θ_i
π_ϕ	Policy learned by actor network ϕ
$Z^\pi(x,a)$	Random state–action return
$\mathcal{Z}^\pi(Z^\pi(x,a)\|x,a)$	Distribution of returns
Q_{lr}	Learning rate for critic network
π_{lr}	Learning rate for actor network
α	Temperature parameter
β_α	Temperature learning rate
γ	Discount factor
λ	Standard deviation weighting factor
η	Update rate
r_{neg}	Negative reward penalty
N_{ep}	Number of training episodes

compliance with the subframe duration in both 4G and 5G radio interfaces, we set L=1 ms.

All other system aspects, concerning task generation and handling as well as data transfer over the radio interface and the network model, are described below. Moreover, for improved readability, Table 1 provides an overview of the adopted notation and its corresponding definitions.

3.1. Task queuing and generation model

Each task generated by CAV n is described by the tuple $[\delta_n, \tau_n, \boldsymbol{c}_\epsilon, \boldsymbol{a}_\epsilon]$, where δ_n denotes the size of the data packet to be transmitted when

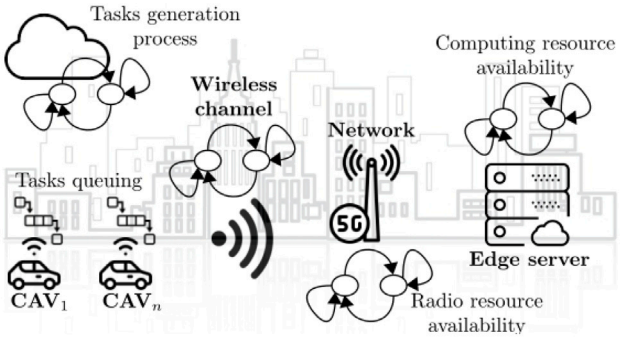


Fig. 1. Considered system scenario.

remote execution is selected, and τ_n is the task deadline, indicating that the task must be completed within τ_n time slots. The vector $\boldsymbol{c}_\epsilon = [c_{\epsilon,e}]$, with $e \in \{1, \dots, N_\epsilon\}$, specifies the number of Million Operations (MOP) required for executing the task when exit e of the DNN model required for task of application ϵ is selected. Finally, $\boldsymbol{a}_\epsilon = [a_{\epsilon,e}]$ is the corresponding accuracy vector, where $a_{\epsilon,e}$ denotes the accuracy achieved when selecting exit e, and $a_{\epsilon,1} < \cdots < a_{\epsilon,N_\epsilon}$. The accuracy values $a_{\epsilon,e}$ can be obtained through offline profiling of a DNN architecture modified with explicitly trained intermediate classifiers, as commonly adopted in early-exit neural networks [29–31]. Each exit produces predictions in the same output space as the final classifier, enabling accuracy evaluation at intermediate layers.

CAV n generates tasks according to a Poisson process with rate parameter ζ_n. Specifically, the probability that k new tasks arrive at CAV n over an interval of m time slots, with $m = 0, 1, \dots$, is given by

$$\Pr(i_n(t+mL) = k) = \frac{\exp(-\zeta_n mL)(\zeta_n mL)^k}{k!}, \tag{1}$$

with $k = 0, 1, \dots$.

Each CAV n stores tasks to be executed in a queue $\boldsymbol{q}_n(t) = [\tau_{1,n}(t), \tau_{2,n}(t), \dots, \mathbf{0}_n(t)] \in \mathbb{N}^{Q_{\max}}$ with finite capacity $Q_{\max}$, where $\tau_{i,n}(t)$ represents the remaining time slots for executing the task i and $\mathbf{0}_n(t)$ is the zero vector representing the empty slots of the queue. Tasks are executed sequentially according to a First-In First-Out (FIFO) manner, i.e., tasks are processed in the order in which they are generated. In particular, the task being in the first position of the queue, i.e. $i = 1$, is the one to be executed. Fig. 2 illustrates an example of the queue dynamics. Note that if a task i is generated at time slot t_0 at CAV n, then $\tau_{i,n}(t_0) = \tau_n$, $\tau_{i,n}(t_0+1) = \tau_n - 1$ and so on until the task is removed from the queue due to task execution or task expiration, which occurs when $\tau_{i,n}(t) \le 0$.

Let $q_n(t)$ denote the amount of backlogged tasks in the queue $\boldsymbol{q}_n(t)$ of the CAV n, at time slot t; the evolution of $q_n(t)$ is described by

$$q_n(t+1) = \min\{\max\{q_n(t) - u_n(t), 0\} + i_n(t), Q_{\max}\}, \tag{2}$$

where $u_n(t)$ represents the number of removed tasks at slot t and $i_n(t)$ is the number of incoming tasks at t. When the queue reaches its maximum capacity $Q_{\max}$, new incoming tasks are discarded.

3.2. Wireless channel model

We assume that the wireless link between the nth CAV and the base station (gNodeB) connected to the edge server is subject to small-scale fading, described by a complex channel gain $h_n(t) = h_{I,n}(t) + j h_{Q,n}(t)$, where $h_{I,n}(t)$, $h_{Q,n}(t)$ are independent, zero-mean Gaussian random processes. Under this assumption, the envelope of the channel gain, $r_n(t) = \sqrt{h_{I,n}^2(t) + h_{Q,n}^2(t)}$, follows the Rayleigh distribution. It is worth noting that the proposed framework is not limited to the Rayleigh distribution,

and can also accommodate different channel models depending on the propagation environment.

To enable a tractable representation of the fading process, we adopt a finite-state Markov chain (FSMC) model, where the continuous range of $r_n(t)$ is discretized into a finite number of intervals, each corresponding to a state of the Markov chain. The intervals are identified by a set of thresholds that can be obtained by imposing the steady state probabilities π_i, i=1, … , N_r, of the channel states, N_r being the desired number of states. Following [32], we assume a uniform steady state distribution, that is π_i=1/N_r, $\forall i$, and we determine the thresholds Γ_i and Γ_{i+1} by solving

$$\int_{\Gamma_i}^{\Gamma_{i+1}} f_r(r)dr = 1/N_r, \tag{3}$$

where $f_r(r)$ is the Rayleigh Probability Density Function (PDF).

Solving Eq. (3) yields

$$\Gamma_i = \sqrt{-2\sigma^2 \ln\left(1 - \frac{i}{N_r}\right)}. \tag{4}$$

Given the partitioning scheme in (4), we say that the chain is in state r_i if $r_n(t) \in [\Gamma_i, \Gamma_{i+1})$ and the discretized channel gain is given as the midpoint of the interval. Since the Rayleigh distribution has unbounded support, in practice we follow the procedure in [32] and truncate the fading range at a sufficiently large value M, such that the Rayleigh Cumulative Distribution Function (CDF) beyond M is negligible. The resulting state probabilities are subsequently renormalized.

The transition probability matrix $\mathbf{P_R}$=$(p_{R,ij})_{i,j=1}^{N_r}$ is derived based on a series expansion of the bivariate Rayleigh cumulative distribution function (CDF), as detailed in [32]. The temporal dynamics of the fading process are modeled using Clarke's isotropic scattering model [33], which characterizes the correlation between two consecutive fading samples as $J_0(2\pi f_D L)$, where J_0 denotes the Bessel's function of the first kind and order 0, f_D is the maximum Doppler frequency and L is the sampling interval, assumed here to be equal to the time slot duration.

The channel state plays a critical role in determining the performance of task offloading. In particular, we consider a system-level constraint on the Block Error Rate (BLER), defined as the average probability of unsuccessful transmission of a block of bits. To ensure that the BLER remains below a predefined threshold, CAvs dynamically adapt their transmission rate based on the observed Signal-to-Noise Ratio (SNR). The SNR, in turn, is a function of the instantaneous channel gain $r_n(t)$, given fixed transmission and noise powers.

To adapt the transmission rate, each CAV dynamically selects the Modulation and Coding Scheme (MCS) based on $r_n(t)$ [34]. Each MCS specifies a modulation order and a coding rate, which determines the level of redundancy introduced during transmission. These configurations are standardized by the 3GPP in its various releases. Empirical mappings between SNR and the optimal MCS selection are typically provided for a reference BLER threshold (e.g., BLER = 0.1). In our work, when considering BLER values different from the reference mapping, we still rely on the same SNR-MCS mapping, thus overestimating (or underestimating) the performance metrics. Indeed, higher target BLER values enable the use of more aggressive MCS levels, while lower target BLER values require more conservative MCS selections in order to satisfy stricter reliability constraints. The impact of this approximation is quantitatively assessed in Section 5.2. We refer to [35] for the SNR-MCS mapping at BLER = 0.1

Finally, the transmission rate also depends on the available radio resources, as explained in the next section.

3.3. Radio resource allocation

The base station connected to the edge server allocates communication resources based on Orthogonal Frequency Division Multiple Access (OFDMA). Radio resources are represented in terms of Physical Resource Blocks (PRBs), defined according to the 5G New Radio (NR) standard as contiguous groups of 12 subcarriers along the frequency axis in the OFDM plane over a time slot, whose duration depends on the chosen subcarrier spacing. Let $w(t)$ denote the total number of PRBs available for communication with the edge server at time t, and let $w_n(t) = \xi_n(t)w(t)$, $\xi_n(t) \in [0, 1]$, represent the fraction of PRBs allocated to CAV n. The availability of radio resources generally varies over time, depending on user demand and network conditions. To capture this variability, we model $w(t)$ as a discrete-time Markov chain taking values in the finite set $\{W_1, \dots, W_{N_w}\}$, where the transition probabilities can follow any arbitrary distribution to reflect realistic network dynamics. The proposed formulation is general and can accommodate both time-varying and static radio resource availability.

The assigned radio resources, together with the MCS, allows to evaluate the transmission rate by the CAVs. In particular, the achievable rate for offloading a task is given as [36]

$$\begin{aligned} R_{\text{off},n}(w_n(t), r_n(t)) = &\sum_{j=1}^{J} \upsilon_{\text{layer}}^{(j)} Q_m^{(j)} f^{(j)} R_{\max} \\ &\times \frac{12 w_n(t)}{T_s^{\mu}} (1 - \text{OH}^{(j)}), \end{aligned} \tag{5}$$

where J is the number of aggregated component carriers, $\upsilon_{\text{layer}}^{(j)}$ is the maximum number of supported layers which depends on uplink or downlink transmissions, $Q_m^{(j)}$ is the spectral efficiency given by the selected MCS, $f^{(j)}$ is a scaling factor, $R_{\max}$ = 948/1024, T_s^{μ} is the duration of the OFDM symbols, which is related to the adopted subcarrier spacing, or, equivalently, to the numerology μ and $\text{OH}^{(j)}$ is the overhead of the packet.

Given the transmission rate, we can determine the time required for offloading a task as

$$T_{\text{off},n}(t) = \frac{\delta_n}{R_{\text{off},n}(t)}. \tag{6}$$

It is important to note that task offloading may result in complete failure with a probability equal to the BLER. In such cases, retransmission is allowed in the next time slots, but the time spent for the offloading procedure is lost.

3.4. Computational resource allocation

The edge server shares its total amount of computational resources $c(t)$, which are expressed in terms of FLOPs per second (OPS). Similarly to radio resources assignment, we model the temporal variation of available computing resources as a Markov chain $c(t) \in \{C_0, \dots, C_{N_c}\}$ and the share of OPS reserved to CAV n for remote execution is given as $c_n(t) = \rho_n(t)c(t)$, where $\rho_n(t) \in [0, 1]$ is the allocation ratio. In contrast, each CAV is equipped with a fixed local computational capability denoted by C_n.

Given the available computational resources, the local execution time at CAV n can be defined as

$$T_{\text{loc},n}(t) = \frac{c_{\epsilon,e}}{C_n}. \tag{7}$$

Conversely, the remote execution time also accounts for the offloading delay, and is expressed as

$$T_{\text{rem},n}(t) = T_{\text{off},n}(t) + \frac{c_{\epsilon,e}}{c_n(t)}, \tag{8}$$

where $T_{\text{off},n}(t)$ denotes the time required to transmit the task to the edge server as defined in Eq. (6).

Finally, the overall system state evolution is described by the state vector $\boldsymbol{x}(t)$, which aggregates all the described system components and is defined as

$$\boldsymbol{x}(t) = [\boldsymbol{q}_1(t), \dots, \boldsymbol{q}_{N_u}(t), \boldsymbol{r}(t), w(t), c(t)], \tag{9}$$

where $\boldsymbol{r}(t) = [r_n(t)]$ is the vector collecting the channel states of the CAVs.

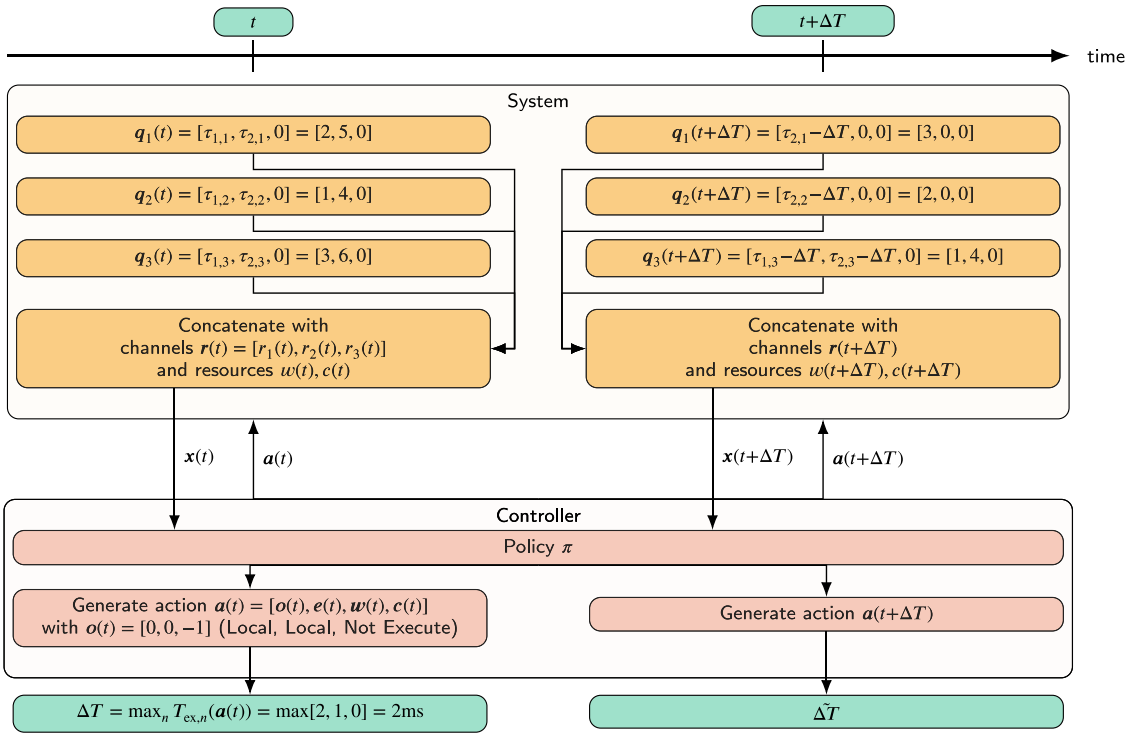


Fig. 2. Example of queue evolution for 3 users. Each queue is a fixed-length vector of remaining time slots, denoted by $\tau_{i,n}$, with i the queue position and n representing the user. The controller observes the system state $\boldsymbol{x}(t)$ and outputs the action $\boldsymbol{a}(t)$. In this example, the controller decides to execute locally the tasks of the users 1 and 2, while for user 3 the system prefers not to execute, hence postponing the execution to another time. The next state is evaluated after ΔT time slots, determined by the largest execution time. Tasks deadlines are decreased accordingly and executed tasks are removed from the queue. Only one task per user can be executed per time slot, and the execution strategy always refers to the first task in the queue.

4. Task offloading resource allocation and early exit selection problem

Our goal is to derive the optimal task execution strategy that maximizes accuracy and minimizes execution time, while preventing queue overflows. This must be accomplished under stringent system constraints, namely that each task must be executed within its application-defined deadline and that queue capacities must not be exceeded. To this end, we jointly control three key system decisions: *(i)* task execution decisions, *(ii)* allocation of radio and computational resources, and *(iii)* the selection of early exits.

We first formulate the optimization problem and discuss its complexity. Then, we illustrate the solution approach: as already mentioned, it consists in a novel variant of DRL, that is able to quantify the uncertainty of the learned control policy.

4.1. Problem formulation

We denote by $o_n(t) \in \{-1, 0, 1\}$ the execution decision for user n at time slot t. Specifically, $o_n(t) = 0$ indicates that the task at the head of the queue is executed locally, $o_n(t) = 1$ indicates remote execution via offloading, and $o_n(t) = -1$ denotes that no execution decision is taken at time slot t (idle action). When $o_n(t) = -1$, the task remains in the queue, its execution is deferred to subsequent time slots and, for consistency, the execution time variable $T_{\text{ex},n}(\boldsymbol{a}(t))$ is set to 0. Importantly, the task's remaining time to deadline $\tau_{n,i}(t)$ continues to decrease at each time slot, so prolonged idling increases the risk of deadline expiration. The example in Fig. 2 illustrates this behavior. Moreover, let $e_n(t) \in \{1, \dots, N_{n,e}\}$ be the selected exit of the deep model used by user n, and let $w_n(t)$ and $c_n(t)$ represent the amounts of radio and computational resources allocated to user n, respectively. We define the system execution strategy at time t as $\boldsymbol{a}(t) = [\boldsymbol{o}(t), \boldsymbol{e}(t), \boldsymbol{w}(t), \boldsymbol{c}(t)]$, where $\boldsymbol{o}(t)$, $\boldsymbol{e}(t)$, $\boldsymbol{w}(t)$ and $\boldsymbol{c}(t)$ are the vectors collecting offloading decisions, early exit selections, and resource allocations for all users.

In order to determine a policy $\boldsymbol{a}(t)$ that balances model accuracy and execution latency, while ensuring reliable system operation by preventing queue overflows, we first define the instantaneous utility function $\psi(t)$ we want to maximize as

$$\psi(t) = \sum_{n}^{N_u} \nu_1 \frac{A_n(e_n(t))}{A_{\max,n}} + \nu_2 \left(1 - \frac{T_{\text{ex},n}(\boldsymbol{a}(t))}{\tau_n}\right), \tag{10}$$

where $A_n(e_n(t))$ is a function denoting the accuracy achieved by user n when selecting the model exit $e_n(t)$ as part of the execution strategy $\boldsymbol{a}(t)$. The function $T_{\text{ex},n}(\boldsymbol{a}(t))$ represents the corresponding execution time for user n under action $\boldsymbol{a}(t)$. The parameters $\boldsymbol{\nu} = [\nu_1, \nu_2]$, satisfying $\nu_1 + \nu_2 = 1$ and $\nu_1, \nu_2 \geq 0$, are used to weigh the relative importance of accuracy and latency in the optimization objective. Then, we formulate the following TORA-EE optimization problem:

$$\underset{\boldsymbol{a}(t)}{\text{argmax}} \quad \lim_{T\to\infty} \frac{1}{T} \sum_{t}^{T} \psi(t) \tag{11}$$

$$\text{s.t.} \quad T_{\text{ex},n}(\boldsymbol{a}(t)) \leq \tau_{1,n}(t), \forall n, t \tag{12}$$

$$q_n(t) + i_n(t) \leq Q_{\max}, \forall n, t \tag{13}$$

$$\sum_{n} w_n(t) \leq w(t), \forall t \tag{14}$$

$$\sum_{n} c_n(t) \leq c(t), \forall t \tag{15}$$

$$\xi_n(t) \in [0, 1], \forall n, t \tag{16}$$

$$\rho_n(t) \in [0, 1], \forall n, t \tag{17}$$

$$o_n(t) \in \{-1, 0, 1\}, \forall n, t \tag{18}$$

$$e_n(t) \in \{1, \dots, N_{n,e}\}, \forall n, t \tag{19}$$

where the constraint in Eq. (12) ensures that the task of user n is executed within the remaining deadline $\tau_{1,n}(t)$, the constraint in Eq. (13) guarantees that no new incoming tasks are discarded whereas the constraints in Eqs. (14)–(15) ensure that the allocation of radio and remote computational resources does not exceed their limits. Finally, the constraints in Eqs. (16)–(19) limit the search space of the decision variables.

The decision variables are inherently coupled through the task execution process. In particular, the offloading decision $o_n(t)$ determines whether a task is executed locally, offloaded to the edge server, or postponed to a future time slot. The selected exit point $e_n(t)$ determines both the inference accuracy and the computational workload associated with the task, thereby affecting its execution time. Furthermore, the radio resource allocation $w_n(t)$ influences the achievable transmission rate and the communication delay experienced by offloaded tasks, while the computational resource allocation $c_n(t)$ affects the processing delay at the edge server. These latency components jointly determine the overall execution time $T_{\text{ex},n}(\boldsymbol{a}(t))$, which in turn impacts queue evolution, deadline satisfaction, and the possibility of task drops. Therefore, offloading decisions, exit-point selection, and resource allocation decisions are jointly optimized within the proposed TORA-EE framework.

4.2. A novel solution based on deep distributional reinforcement learning

The TORA-EE problem, defined in Eqs. (10)–(19), falls into the class of Mixed-Integer Nonlinear Programming (MINLP) problems, which are known to be NP-hard. This classification arises from the hybrid nature of the decision variables: while the resource allocation variables can be assumed as continuous, the task execution and early exit selection variables are discrete. Furthermore, the execution time expressions introduce nonlinearity due to their dependence on the allocated resources, then contributing to the problem's complexity.

Moreover, due to the complexity of the considered scenario, we could not adopt the solution approach proposed in [7], where a significantly simpler problem was addressed by leveraging a standard procedure for mapping a MDP into a Linear Program (LP). While the Markov property still holds in our system model, the high dimensionality of both the state and action spaces renders the construction and manipulation of transition probability matrices computationally infeasible. To effectively exploit the underlying Markovian structure, we therefore resort to DRL approach, which can scale to large state–action spaces without requiring explicit knowledge of the transition dynamics.

A recent advancement in the field of DRL is DistRL [37], which shifts the focus from learning the expected return (i.e., the value function) to learning the entire probability distribution of returns. While the value function captures only the average system behavior under a given policy π, distributional methods provide a richer representation of the outcome space. By modeling the distribution of returns, DistRL enables the design of risk-sensitive policies, which can explicitly consider higher-order moments such as the standard deviation. This is particularly advantageous in systems where minimizing standard deviation or avoiding worst-case outcomes is as relevant as optimizing average performance.

The main difference with traditional RL algorithms is that for every state x there is the need to learn the probability distribution of the possible cumulative rewards that can be obtained from that state, following a policy π, rather than the scalar value representing the expectation of it. Also, different policies lead to different distribution returns.

In order to characterize the DistRL algorithms, we first define the state–action return as in [38]

$$Z^{\pi}(x_t, a_t) = r_t + \gamma G_t, \tag{20}$$

where r_t is the instantaneous reward obtained at time t when in state x_t and taking action a_t, $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = r_{t+1} + \gamma G_{t+1}$ is the discounted cumulative reward function, and γ is the discount factor. Note that the state–action return is a random variable. We are then interested in learning the probability density function $\mathcal{Z}^{\pi}(Z^{\pi}(x_t, a_t)|x_t, a_t)$ of the state–action return $Z^{\pi}(x_t, a_t)$, given a certain policy π. The distributional variant of the Bellman operator can be rewritten instead as

$$\mathcal{T}_D^{\pi} Z(x_t, a_t) \overset{D}{=} r + \gamma Z(x_{t+1}, a_{t+1}), \tag{21}$$

where $\overset{D}{=}$ means that the terms at both sides of the equation have to follow the same probability distribution.

To solve our optimization problem, we adopted a distributional variant of the Soft Actor-Critic (SAC) deep reinforcement learning algorithm, specifically the Distributional SAC with Three Refinements (DSAC-T) [38]. DSAC-T extends the original DSAC algorithm [39] and incorporates several enhancements to improve learning stability and convergence speed. Similar to standard actor–critic architectures, DSAC-T employs separate DNNs to approximate the return distribution and the optimal policy, referred to as the critic and actor networks, respectively. To ensure stable training, DSAC-T uses target networks, which are delayed copies of the critic and actor networks updated more slowly. In this framework, both the return distribution and the policy are modeled as Gaussian distributions. Consequently, each of the critic and actor networks outputs two parameters, the mean and the standard deviation, representing the respective distributions. Formally, the critic network of parameters θ outputs the return distribution $\mathcal{Z}_{\theta}(\cdot|x, a)$ in terms of average of the distribution $Q_{\theta}(x, a)$ and standard deviation $\sigma_{\theta}(x, a)$, while the actor network of parameters ϕ outputs the mean and the standard deviation of the policy π_{ϕ}. The target critic and actor networks have parameters $\bar{\theta}$ and $\bar{\phi}$, respectively, which are updated according to the rule

$$\bar{\theta} \leftarrow \eta\theta + (1 - \eta)\bar{\theta}, \tag{22}$$

and

$$\bar{\phi} \leftarrow \eta\phi + (1 - \eta)\bar{\phi}, \tag{23}$$

where η is the update rate.

The DSAC-T is also a *maximum entropy* RL algorithm, meaning that the reward function encompasses a term representing the entropy of the policy π. Then, the entropy-augmented discounted cumulative reward function, when following a policy π being in state x_t, is expressed as

$$G^{\pi}(x_t) = \sum_{k=0}^{\infty} \gamma^k [r_{t+k+1} + \alpha \mathcal{H}(\pi(\cdot|x_{t+k+1}))], \tag{24}$$

where

$$\mathcal{H}(\pi(\cdot|x_t)) = \mathbb{E}\Big[-\log \pi(a_t|x_t)\Big] \tag{25}$$

is the entropy of the policy and α is the temperature parameter updated according to

$$\alpha \leftarrow \alpha - \beta_{\alpha} \mathbb{E}[-\log \pi(a_t|x_t) - \bar{\mathcal{H}}], \tag{26}$$

where β_{α} is the learning rate and $\bar{\mathcal{H}}$ is a target entropy. The maximum entropy-augmented action-state return can be then rewritten with the use of the distributional Bellman operator

$$\mathcal{T}_D^{\pi} Z(x, a) \overset{D}{=} r + \gamma(Z(x', a') - \alpha \log \pi(a'|x')), \tag{27}$$

where, starting from now on, we omit the temporal dependency of the involved quantities for notational simplicity and with x', a' we indicate the state and the action in the next time step, respectively. The distribution of $\mathcal{T}_D^{\pi} Z(x, a)$ can be denoted as $\mathcal{T}_D^{\pi} \mathcal{Z}(\cdot|x, a)$.

In order to learn the probability distribution of the state action return, the critic network update rule is the minimization of

$$J_{\mathcal{Z}}(\theta) = \mathbb{E}\Big[D_{\text{KL}}(\mathcal{T}_D^{\pi_{\bar{\phi}}} \mathcal{Z}_{\bar{\theta}}(\cdot|x, a), \mathcal{Z}_{\theta}(\cdot|x, a))\Big], \tag{28}$$

where $D_{\text{KL}}(\cdot)$ is the Kullback–Leibler divergence, which represents the distance between two probability distributions, $\mathcal{Z}_{\bar{\theta}}(\cdot|x, a)$ and $\mathcal{Z}_{\theta}(\cdot|x, a)$

are the reward distribution learned by the critic networks and $\pi_{\tilde{\phi}}$ is the policy learned by the target actor network. Actually, the DSAC-T relies on twin value distribution learning, meaning that there are two independently trained critic networks of parameters $\theta_i, i = \{1, 2\}$, where the one associated with the smaller mean of the value distribution is used for constructing the gradients of the actor and the critic. Accordingly, Eq. (28) is then rewritten in a sample-based form, for the critic network i, as

$$J_{\hat{Z}}(\theta_i) = \omega_i \mathbb{E}\left[\frac{(y_z - Q_{\theta_i}(x,a))^2}{2\sigma^2_{\theta_i}(x,a)} + \log(\sqrt{2\pi}\sigma_{\theta_i}(x,a))\right], \tag{29}$$

where

$$y_z = r + \gamma(Z(x',a') - \alpha \log \pi_{\tilde{\phi}}(a'|x')) \tag{30}$$

is the target value for the return, with $Z(x',a') \sim \mathcal{Z}_{\tilde{\theta}_i}(\cdot|x',a')$, and $\omega_i = \mathbb{E}_\pi[\sigma^2_{\theta_i}(x,a)]$ serves as a scaling factor to prevent gradient explosion in the update of $J_{\hat{Z}}(\theta_i)$ when $\sigma_{\theta_i}(x,a) \to 0$. This scaling parameter is dynamically updated according to the following rule

$$\omega_i \leftarrow \eta \mathbb{E}\left[\sigma^2_{\theta_i}(x,a)\right] + (1-\eta)\omega_i, \tag{31}$$

where η is the update rate.

The actor update rule is the maximization of

$$J_\pi(\phi) = \mathbb{E}\left[\min_{i=1,2} Q_{\theta_i}(x,a) - \alpha \log(\pi_\phi(a|x)) - \lambda\sigma_{\theta_i}(x,a)\right], \tag{32}$$

where the term $\lambda\sigma_{\theta_i}(x,a)$ has the purpose of diminishing the randomness of the return.

In our case, we denote with $\mathcal{X}$ the set of system states, where the system state $\boldsymbol{x}(t)$ at time instant t is given by Eq. (9), whereas we denote with $\mathcal{A}$ the set of possible actions, where the action $\boldsymbol{a}(t)$ undertaken at time instant t consists of the already mentioned offloading decisions $\boldsymbol{o}(t)$, remote resource allocation $\boldsymbol{c}(t)$ and $\boldsymbol{w}(t)$ and the early exit selection $\boldsymbol{e}(t)$ for all the N_u users in the system.

In the proposed RL formulation, queue overflow and execution deadline constraints in (12)–(13) are modeled as soft constraints through the reward function rather than as hard optimization constraints. Specifically, whenever a task violates its execution deadline or is dropped due to queue overflow, a negative reward term r_{neg} is applied, determining the new utility function $\tilde{\psi}(t)$ defined as:

$$\tilde{\psi}(t) = \psi(t) r_{\text{pos}} + r_{\text{neg}}(N_{\text{late}}(t) + N_{\text{drop}}(t)), \tag{33}$$

where $N_{\text{late}}(t)$ denotes the number of tasks that violate their execution deadline, including both tasks whose estimated execution time exceeds the remaining deadline and tasks whose deadline expires while waiting in the queue, and $N_{\text{drop}}(t)$ is the number of dropped tasks whenever $q_n(t)+i_n(t) > Q_{\max}$. The parameter r_{pos} is a positive scaling factor applied to balance the magnitude between reward and penalty terms. Then, the reward at each time step is computed according to Eq. (33). In practice, the magnitude of r_{neg} is chosen sufficiently large to make constraint violations suboptimal under the learned policy. This approach preserves the unconstrained MDP structure while strongly discouraging infeasible behaviors.

Details on how action feasibility is enforced are provided in Section 5.1, where the training procedure is described. Specifically, the actions generated by the agent are mapped onto the feasible action set $\tilde{\mathcal{A}}(t)$ before being applied to the environment, while deadline violations and queue overflows are handled through the penalty term in the utility function. Therefore, the TORA-EE problem, formulated in Eqs. (11)–(18), is addressed in this work with the following unconstrained reformulation (see Fig. 3).

$$\underset{\boldsymbol{a}(t)\in\tilde{\mathcal{A}}(t)}{\text{argmax}} \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \tilde{\psi}(t). \tag{34}$$

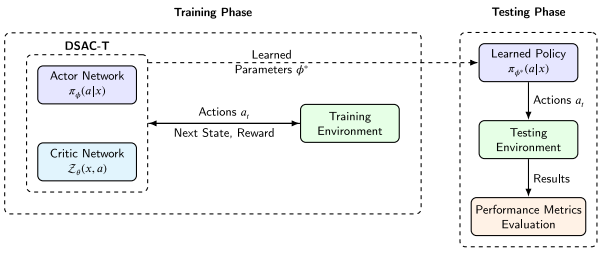


Fig. 3. Workflow diagram: During the training phase, the DSAC-T algorithm interacts with the environment and updates its parameters for learning the value distribution while optimizing the policy loss function. After the training phase, the learned policy is tested with the environment for assessing its performance.

5. Performance results

In this section, we discuss the performance of the analyzed system by showing the results obtained with the policy learned by the DSAC-T algorithm. We first detail the training process and the performance evaluation setting. Then, we present the results obtained with various system scenarios and under different training settings. We conclude the section by showing also the convergence of the learning process.

5.1. Training and performance assessment details

We trained the DSAC-T algorithm by randomly generating the system parameters. The considered parameter ranges were selected to cover heterogeneous but realistic wireless and edge computing scenarios rather than a single deployment configuration. Note that the parameters are not intended to represent a single consistent physical configuration but rather a distribution of plausible conditions to improve the generalization capability of the DSAC-T algorithm.

It is assumed that the users can perform two different tasks, hence, the running application ϵ was sampled from the interval $\{1, 2\}$, each requiring a different neural network architecture. The first neural network, equipped with 3 early exits, including the final exit, requires to perform the following $\boldsymbol{c}_1 = [22.622, 25.955, 59.506]$ MOP per exit, where each exit has associated the following accuracy $\boldsymbol{a_1} = [56.37, 78.04, 85.95]$, as in [40]. The second neural network has the computational requirement $\boldsymbol{c_2} = [0.711, 0.752, 0.794]$ MOP and the related accuracy $\boldsymbol{a_2} = [38.97, 51.93, 93.91]$.

The deadlines τ_n of the tasks were sampled from the interval $[50, 100]$ ms, the Poisson parameter ζ_n for the generation of the tasks from the interval $[0.001, 0.01]$ task/ms and the task data size δ_n from $[10\,000, 100\,000]$ bit, to capture heterogeneous service requirements.

The tasks could be executed locally or remotely where the local computing capability was sampled from $[0.002, 0.005]$ Tera OPerations per Seconds (TOPS), the maximum remote computing capability from the interval $[0.009, 0.02]$ TOPS while the maximum available number of PRBs from the interval $[150, 300]$ PRBs, which corresponds to medium-to-large bandwidth allocations typical of modern cellular systems depending on the adopted numerology. The normalized Doppler frequency $f_D L$ in $[0.1, 1]$, covering mobility conditions from low to moderately high vehicular speeds across different carrier frequencies, while the BLER was selected in $[0.01, 0.2]$, reflecting practical link adaptation operating regions commonly targeted in wireless systems.

The number of users N_u was fixed to 3, the number of remote computing resources states $N_c = 1$ and the number of radio resources states $N_w = 1$, meaning that we set constant remote computational and radio resources over time, but the approach can be easily extended to different resources states, This choice allows us to isolate and highlight the impact of early exit decisions, queue dynamics, and learning-based control without introducing additional variability due to fluctuating

Table 2
Values of hyperparameters, unless otherwise stated.

Hyperparameter	Value
Q_{lr}	0.0001
π_{lr}	0.0001
β_a	0.0003
γ	0.99
η	0.005
r_{neg}	−100
r_{pos}	100
λ	0

radio resources. Nevertheless, the proposed model and solution framework naturally extend to scenarios with time-varying radio resources. The number of channel states is set to $N_r = 15$. Regarding the selection of the MCS according to the channel gain and to the BLER constraint, we relied on the approach described in Section 3.2. The maximum capacity of the queues $Q_{\max}$ was instead fixed to 10.

The hyperparameters of the DSAC-T algorithm are listed in Table 2, unless otherwise stated. The number of training iterations was fixed to $3e6$, with a number of episodes $N_{\text{ep}} = 12$, implying episodes of $2.5e5$ iterations. In each episode, the system parameters including task generation rates, running applications, maximum amount of resources, ..., were randomly sampled from the previously described intervals. Moreover, every 200 iterations the system state was restarted to a system state with empty queues.

Since the DSAC-T actor network produces only continuous outputs, discrete decisions, such as offloading choices and early-exit selections, are obtained by rounding the corresponding continuous components to the nearest admissible integer values. Investigating dedicated hybrid-action distributional RL algorithms constitutes an interesting direction for future work.

In order to deal with unfeasible actions, we decided to map them onto a feasible space, without giving any negative reward for discouraging unfeasible actions [41]. In particular, for radio and computational resource allocation, the actor network outputs unconstrained non-negative continuous values for each user. To enforce the capacity constraints in Eqs. (14)–(15), we adopt a proportional normalization mechanism. Specifically, let $\rho(t) = [\rho_1(t), \dots, \rho_{N_u}(t)]$ denote the raw actor outputs for computational allocation. The executed allocation is obtained as

$$\tilde{\rho}_n(t) = \frac{\rho_n(t)}{\sum_{k=1}^{N_u} \rho_k(t)}, \tag{35}$$

which guarantees by construction that constraint in Eq. (15) is satisfied. This transformation preserves the relative proportions among users decided by the policy, while strictly enforcing resource feasibility at every time step. Whenever the normalization denominator is equal to zero, the resource allocation variables are set to zero for all users. An analogous normalization is applied to radio resource allocation.

We conducted performance assessment of the trained networks by testing the learned policies. As regard the testing parameters, we denote with $\boldsymbol{\zeta} = [\zeta_1, \dots, \zeta_{N_u}]$ the vector containing the Poisson parameters for the generation of the tasks of the users, with $\boldsymbol{\tau}$ the vector containing the deadlines of the tasks, with $\boldsymbol{\delta}$ the vector containing the data size of the tasks expressed in bits and with $\boldsymbol{\epsilon}$ the chosen applications.

For each tested model, average metrics per user were derived, consisting of average completion ratio, defined as the ratio between the number of executed tasks N_{ex} and the number of generated tasks N_{gen}, the average normalized accuracy, where the normalization occurs with respect to the maximum achievable accuracy $A_{\max}$, the average execution time T and the cumulative reward R obtained by cumulating all the rewards obtained during the test performance.

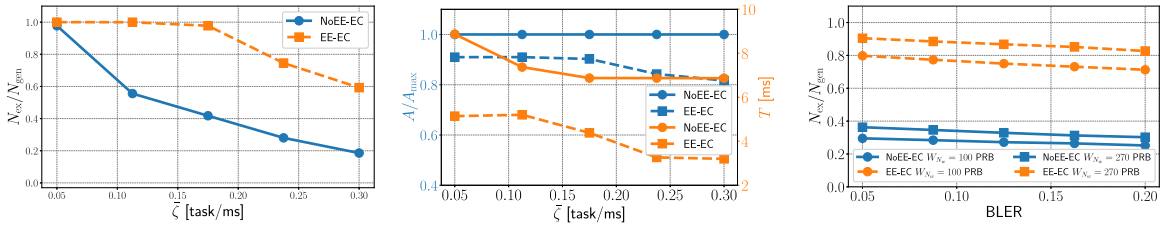
We first show the benefits of relying on EE with respect to a classic EC system which does not rely on EE. In particular, we indicate with EE-EC the system relying on EE, while with NoEE-EC the classic EC system. We then analyze the impact of the reward function design and the penalty terms on the learned policies by comparing the DSAC-T approach against both SAC and PPO. SAC is widely adopted in task offloading and resource allocation problems and therefore provides a suitable reference for evaluating the benefits of learning the full return distribution. Similarly, PPO represents one of the most widely used policy-gradient reinforcement learning algorithms and serves as an additional benchmark for assessing the effectiveness of the proposed approach. While SAC shares a similar actor-critic architecture with DSAC-T, it only learns the expected return rather than the full return distribution. PPO, on the other hand, relies on a clipped policy optimization objective and follows a different training paradigm. In this set of experiments, we set $\lambda = 0$ in order to isolate and analyze the effects of EE and the reward design without the influence of the risk-aware term. The risk-aware behavior of the DSAC-T is instead specifically investigated in Section 5.4, where λ is varied and its impact on the learned policy is analyzed. Fig. 3 illustrates the overall workflow of the proposed methodology.

5.2. EE vs. NoEE

In Fig. 4(a) we compare the average completion ratio of the 3 users as a function of the task generation parameter $\boldsymbol{\zeta}$. The horizontal axis represents the average of the Poisson parameters of the 3 users, denoted by $\bar{\zeta}$. Results show that as $\bar{\zeta}$ increases, the completion ratio decreases. This behavior is primarily due to the fact that, under more aggressive task generation, the system becomes unable to process all the incoming tasks within the required deadlines. However, it can be observed that the EE-EC system successfully executes a larger number of tasks compared to the conventional EC system, showing an increase of up to 212% of executed tasks. This advantage stems from the EE mechanism, which enables the system to dynamically adjust computation times by selectively choosing earlier exits in the DNNs. As illustrated in the next figure, this adaptation helps reduce task execution latency, allowing queues to be emptied more quickly and mitigating the risk of overflow.

The impact of EE on the average execution times and the average accuracy is detailed in Fig. 4(b). As the average task generation rate $\bar{\zeta}$ increases, the EE-EC system dynamically adjusts the trade-off between accuracy and execution time. Specifically, to avoid queue overflows under higher load conditions, the EE-EC system increasingly selects earlier exits in the DNN models. While this strategy leads to a reduction in accuracy, it enables faster task processing and helps maintain system responsiveness despite the growing number of incoming tasks. In contrast, the NoEE-EC system is unable to adapt task accuracy, then leading to the flat accuracy curve: this limitation results in the execution of a smaller number of tasks, as previously shown, primarily due to longer computational times. As depicted in the figure, the NoEE-EC system exhibits consistently larger execution times when compared to the EE-EC system. Additionally, both systems demonstrate a decreasing trend in average execution time as the task generation rate increases. For the EE-EC system, this reduction is largely attributed to the more frequent selection of earlier exits in the DNN models. In the case of the NoEE-EC system, the decrease in execution time is instead a consequence of increased reliance on task offloading to the edge server, which offers significantly greater computational capacity than the individual CAVs.

In Fig. 4(c), we compare the performance of the two systems in terms of completion ratio as a function of the target BLER. The completion ratio is further evaluated under two different radio resource availability conditions, specifically for $W_{N_w} \in 100, 270$ PRB. As expected, the completion ratio generally decreases with increasing BLER, since larger BLER values correspond to a larger number of transmission failures during the offloading process. Moreover, both systems benefit from a larger availability of radio resources, which helps to mitigate the negative impact of unreliable transmissions by reducing the time required for successful offloading. Notably, the EE-EC system consistently outperforms the conventional NoEE-EC system across all



(a) Average completion ratio versus ζ. $\tau = [50, 50, 50]$ ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.005$ TOPS, $W_{N_w} = 270$ PRB, $C_{N_c} = 0.03$ TOPS, $f_D L = 0.65$, BLER = 0.05.

(b) Average execution time and average accuracy versus ζ. $\tau = [50, 50, 50]$ ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.005$ TOPS, $W_{N_w} = 270$ PRB, $C_{N_c} = 0.03$ TOPS, $f_D L = 0.65$, BLER = 0.05.

(c) Average completion ratio versus BLER. $\zeta = [0.2, 0.2, 0.2]$ task/slot, $\tau = [50, 50, 50]$ ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.005$ TOPS, $C_{N_c} = 0.03$ TOPS, $f_D L = 0.65$.

Fig. 4. Performance comparison between NoEE-EC and EE-EC architectures.

Table 3
Average relative rate error (%) induced by using the SNR–MCS mapping calibrated at $\text{BLER}_{\text{ref}} = 0.1$, for different target BLER values and slope parameters β.

β/BLER	0.05	0.0875	0.125	0.1625	0.2	0.5
1.3	−3.73	0	0	13.94	16.50	50.46
1.5	−3.73	0	0	12.60	15.41	48.85
1.8	−3.73	0	0	6.42	14.22	45.50

configurations. Furthermore, the results indicate that the EE-EC system exhibits greater sensitivity to increased radio resource availability, leveraging it more effectively to improve task completion.

To quantify the bias introduced by using a reference SNR–MCS mapping calibrated at BLER = 0.1, we evaluate the induced average rate error when the target BLER differs from the reference value. Table 3 reports the average relative rate error for different BLER targets and slope parameters β. The detailed derivation, and a deeper discussion, are reported in Appendix. In the BLER operating region of practical interest (0.05–0.2), the induced rate error remains moderate and does not alter the qualitative behavior of the proposed optimization framework.

5.3. Reward function

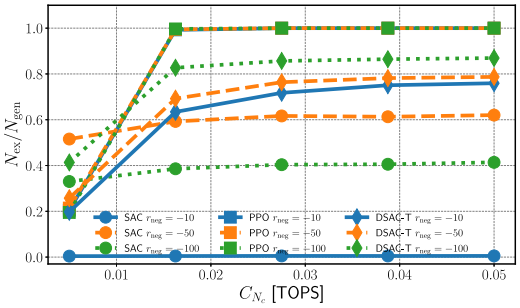
In this section, we analyze the influence of the reward function on system performance. Specifically, we investigate the effect of the penalty term r_{neg} applied when a task is discarded, as well as the roles of the weighting factors ν_1 and ν_2, which correspond to accuracy and execution time, respectively. Additionally, we compare the performance of the DSAC-T algorithm against the SAC and PPO baselines. For both benchmark algorithms, we relied on the Stable-Baselines3 implementation. To ensure a fair comparison, all agents were trained and evaluated under the same environment settings. Moreover, whenever applicable, the training hyperparameters of the baseline methods were aligned with those adopted for DSAC-T.

5.3.1. Penalty

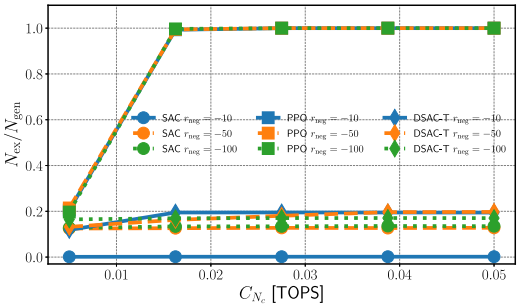
In Fig. 5(a) we report the performance comparison of three EE-MEC policies learned with different values of r_{neg}, namely $r_{\text{neg}} = -10, -50, -100$. In particular, we present the task completion ratio as a function of the total computational resources available at the edge server. The results indicate that the number of executed tasks increases with greater resource availability, primarily due to reduced inference times. The influence of the penalty term r_{neg} on the learned policies becomes especially pronounced at lower values of C_{N_c}, where longer inference times are observed. Specifically, policies trained with smaller penalty values tend to place less emphasis on avoiding task discards, a tendency that becomes more evident as the penalty decreases. When taking r_{neg} constant, the DSAC-T generally performs better than the classic SAC. Notably, the highest completion ratio is achieved by the policy trained with more severe r_{neg}, whereas the lowest one corresponds to the policy trained with $r_{\text{neg}} = -10$ with all algorithms. In particular, when $r_{\text{neg}} = -10$, the SAC algorithm shows a significant degradation in performance, yielding a task completion ratio that is close to zero. This could be because the low value of r_{neg} does not sufficiently penalize discarded tasks, leading the SAC agent to learn a policy that does not prioritize task completion. We also compare the DSAC-T algorithm with the PPO algorithm. The results show that PPO achieves lower completion ratios than DSAC-T when the available remote computational capacity is limited. However, as C_{N_c} increases, PPO becomes the best performing algorithm, achieving the highest task completion ratio among all the considered approaches. Remarkably, PPO is able to execute the largest number of tasks for all the considered values of r_{neg} when sufficient remote computational resources are available, whereas the performance of SAC and DSAC-T remains more dependent on the selected penalty value.

However, when using the DSAC-T algorithm, overly severe penalties may lead to suboptimal policies, as illustrated in Fig. 5(b), which presents the same performance comparison as Fig. 5(a), but under a lower local computational capacity C_n. Importantly, beyond a certain threshold of C_{N_c}, the completion ratio for the policy trained with $r_{\text{neg}} = -100$ no longer improves and begins to fall behind the performance of other policies. This behavior arises because the policy trained with the most severe penalty tends to favor conservative strategies, such as local execution, in an attempt to avoid the uncertainty associated with task offloading and dynamic resource allocation. In contrast, policies trained with less punitive values of r_{neg} are more inclined to exploit remote computing resources, thereby achieving faster execution and higher accuracy, even at the cost of a larger number of discarded tasks. This aligns with the design of the reward function, which prioritizes execution speed and precision. Conversely, the SAC algorithm shows the same trends reported in Fig. 5(a), achieving the lowest performance with $r_{\text{neg}} = -10$, and the highest one with $r_{\text{neg}} = -50$. Interestingly, the PPO algorithm exhibits a behavior that is largely consistent with that observed in Fig. 5(a). In particular, the task completion ratio steadily increases as the remote computational capacity C_{N_c} grows, regardless of the value of r_{neg}. As in the previous scenario, PPO achieves the highest completion ratios for sufficiently large values of C_{N_c}, outperforming both SAC and DSAC-T. This behavior suggests that PPO learns a policy that relies more extensively on remote execution, allowing it to better exploit the additional computational resources available at the edge server and achieve higher task completion ratios under constrained local computing conditions.

To further investigate the behavior of the considered algorithms, Fig. 6 compares the task completion ratio achieved by DSAC-T and PPO as a function of the local computational capacity C_n for two different values of remote computational capacity, namely $C_{N_c} = \{0.005, 0.01\}$ TOPS, while fixing the penalty term to $r_{\text{neg}} = -10$. In this analysis, a limited amount of radio resources is also considered, with $W_{N_w} = 100$ PRBs, in order to assess the behavior of the learned policies when both



(a) Average completion ratio versus C_{N_c}. $\zeta = [0.1, 0.1, 0.1]$ task/ms, $\tau = [50, 50, 50]$ ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.003$ TOPS, $W_{N_w} = 270$ PRB, $f_D L = 0.65$, BLER $= 0.2$.



(b) Average completion ratio versus C_{N_c}. $\zeta = [0.1, 0.1, 0.1]$ task/ms, $\tau = [50, 50, 50]$ ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.001$ TOPS, $W_{N_w} = 270$ PRB, $f_D L = 0.65$, BLER $= 0.2$.

Fig. 5. Performance comparison between DSAC-T, SAC and PPO: the effect of penalties.

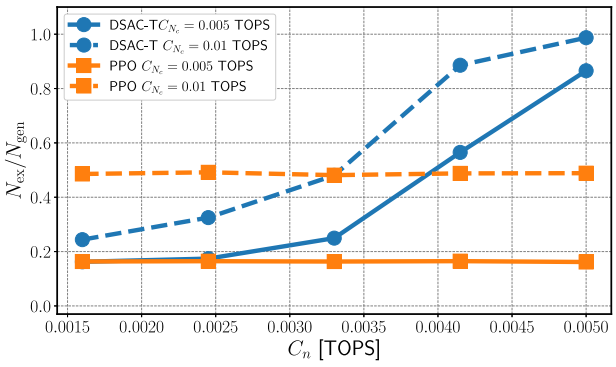


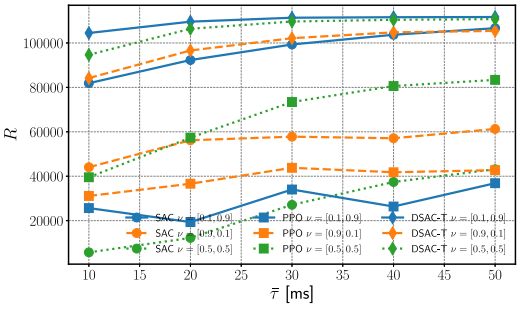
Fig. 6. DSAC-T vs. PPO: Average completion ratio versus C_n. $\zeta = [0.1, 0.1, 0.1]$ task/ms, $\tau = [50, 50, 50]$ ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $W_{N_w} = 100$ PRB, $f_D L = 0.65$, BLER $= 0.2$.

computational and communication resources are constrained. When $C_{N_c} = 0.005$ TOPS, the two algorithms achieve comparable performance for low values of C_n. However, as the local computational capacity increases, DSAC-T increasingly outperforms PPO, exhibiting a clear improvement in task completion ratio. A different trend can be observed when $C_{N_c} = 0.01$ TOPS. In this case, PPO achieves higher performance than DSAC-T for low values of C_n, suggesting a more effective exploitation of the additional remote computational resources under highly resource-constrained local conditions. Nevertheless, as C_n increases, DSAC-T again becomes the best-performing algorithm, benefiting from the greater availability of local computational resources. A notable observation is that PPO appears largely insensitive to variations in the local computational capacity. Indeed, its performance remains nearly constant across the entire range of C_n values considered. In contrast, DSAC-T consistently benefits from increases in local computational resources, achieving progressively higher task completion ratios as C_n grows. This behavior suggests that PPO relies more heavily on remote execution regardless of the available local resources, whereas DSAC-T is able to adapt its execution strategy and more effectively exploit improvements in local computational capacity. As a final remark, since PPO and DSAC-T rely on fundamentally different learning paradigms, the resulting execution strategies may differ significantly even when trained under the same environment settings. A deeper analysis of the relationship between the training dynamics and the learned policies is left for future work.

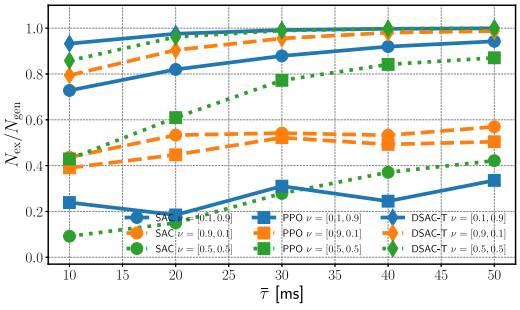
5.3.2. Weights

The weights of the reward function shown in Eq. (10) have to be chosen according to the desired system requirements. In Fig. 7(a) we report the cumulative reward as a function of the average tasks deadlines $\bar{\tau}$ for three different reward weights configurations, namely $\nu = [0.9, 0.1]$, $\nu = [0.5, 0.5]$ and $\nu = [0.1, 0.9]$. The first configuration favors higher accuracy rather than faster execution times, whereas the latter one has an opposite behavior. The middle configuration equally balances the two metrics. For both SAC and DSAC-T, the results show that the models trained with $\nu = [0.1, 0.9]$ achieve the largest cumulative rewards over the considered range of deadlines. However, the two algorithms exhibit different behaviors for the remaining reward configurations, $\nu = [0.5, 0.5]$ and $\nu = [0.9, 0.1]$, highlighting a different sensitivity to the trade-off between execution speed and accuracy. A different trend is observed for PPO. In this case, the highest cumulative rewards are obtained with the balanced configuration $\nu = [0.5, 0.5]$, whereas the configuration $\nu = [0.1, 0.9]$, which places greater emphasis on execution speed, yields the lowest performance. Moreover, the PPO policy trained with $\nu = [0.1, 0.9]$ exhibits larger fluctuations in cumulative reward as the average task deadline varies, resulting in a less regular trend than those observed for SAC and DSAC-T. This behavior suggests a higher sensitivity of PPO to changes in the operating conditions when execution speed is strongly prioritized in the reward function. These results suggest that PPO benefits from a more balanced optimization objective, while SAC and DSAC-T achieve better performance when the reward function prioritizes execution speed over accuracy.

Insights into the cumulative rewards shown in Fig. 7(a) can be better understood by examining the corresponding average completion ratios reported in Fig. 7(b). For both SAC and DSAC-T, the highest completion ratios are achieved by the policies trained with $\nu = [0.1, 0.9]$, consistently with the cumulative rewards observed in Fig. 7(a). Since this reward configuration places greater emphasis on execution speed, the learned policies tend to complete a larger number of tasks within their deadlines. In contrast, the configuration $\nu = [0.9, 0.1]$, which prioritizes accuracy over execution speed, generally leads to lower completion ratios due to the longer execution times associated with more accurate processing strategies. The differences in performance between SAC and DSAC-T are particularly evident for $\nu = [0.9, 0.1]$ and $\nu = [0.5, 0.5]$, whereas the two algorithms achieve comparable completion ratios when trained with $\nu = [0.1, 0.9]$. A different behavior is observed for PPO. In this case, the policy trained with the balanced reward configuration $\nu = [0.5, 0.5]$ achieves the highest completion ratio, outperforming the configurations that place a stronger emphasis on either execution speed or accuracy. This result is consistent with the cumulative rewards reported in Fig. 7(a), suggesting that PPO benefits from a more balanced trade-off between the two optimization objectives.



(a) Cumulative reward versus τ. $\zeta = [0.1, 0.1, 0.1]$ task/ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.005$ TOPS, $C_{N_c} = 0.01$ TOPS, $W_{N_w} = 270$ PRB, $f_D L = 0.65$, BLER $= 0.1$.



(b) Average completion ratio versus τ. $\zeta = [0.1, 0.1, 0.1]$ task/ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.005$ TOPS, $C_{N_c} = 0.01$ TOPS, $W_{N_w} = 270$ PRB, $f_D L = 0.65$, BLER $= 0.1$.

Fig. 7. Performance comparison between DSAC-T, SAC and PPO: the effect of reward weights.

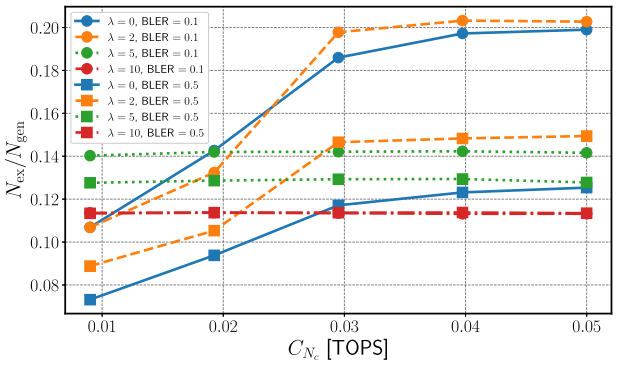


Fig. 8. Risk-sensitivity: Average completion ratio versus C_{N_c}. $\zeta = [0.1, 0.1, 0.1]$ task/ms, $\tau = [30, 30, 30]$ ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.001$ TOPS, $W_{N_w} = 50$ PRB, $f_D L = 0.65$.

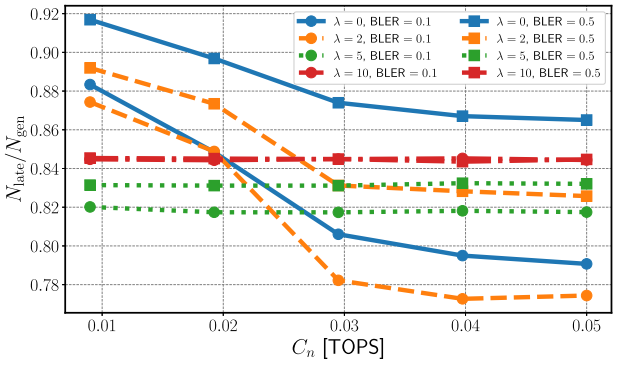


Fig. 9. Risk-sensitivity: Average expired tasks ratio versus C_{N_c}. $\zeta = [0.1, 0.1, 0.1]$ task/ms, $\tau = [30, 30, 30]$ ms, $\delta = [1, 1, 1]$e4 bit, $\epsilon = [1, 1, 1]$, $C_n = 0.001$ TOPS, $W_{N_w} = 50$ PRB, $f_D L = 0.65$.

5.4. Risk-sensitive policy

In Fig. 8 we illustrate the benefits of employing a distributional RL approach. In particular, we compare multiple policies trained with different values of the standard deviation weighting factor λ, evaluating their average completion ratio as a function of the available remote computational resources C_{N_c} under varying BLER conditions. All policies are tested in a constrained scenario characterized by limited local computational capacity ($C_n = 0.001$ TOPS) and radio resources ($W_{N_w} = 50$ PRBs).

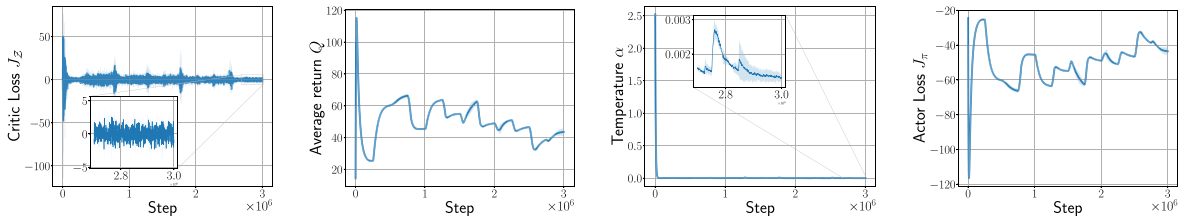
In general, the worst performance is observed for policies trained with higher values of λ, specifically $\lambda = 5$ and $\lambda = 10$, across all BLER levels. This outcome is expected, as larger λ values lead to more risk-averse policies: those policies prioritize safer execution strategies, typically favoring local execution to avoid the uncertainties associated with task offloading. However, this conservative behavior results in suboptimal performance, since the policy does not adapt to the current system parameters. Despite the overall performance degradation, these risk-sensitive policies exhibit interesting behaviors. As already shown in Fig. 4(c), the completion ratio generally decreases with increasing BLER. However, it can now be observed from Fig. 8 that performance remains almost unaffected by the BLER level for $\lambda = 10$: indeed, the curves for BLER $= 0.1$ and BLER $= 0.5$ completely overlap, indicating that the policy consistently chooses local execution regardless of channel conditions. This confirms the tendency of high-λ policies to avoid offloading.

Further evidence of this risk-averse behavior can be found in the insensitivity of performance for $\lambda = 5$ and $\lambda = 10$ to variations in C_{N_c}, thereby evidencing that these policies also avoid to resort on available remote computing resources. Conversely, for $\lambda = 0$ and $\lambda = 2$, we observe a more adaptive behavior. Specifically, at low BLER the policy with $\lambda = 2$ eventually outperforms the one with $\lambda = 0$ as C_{N_c} increases. Under high BLER, the $\lambda = 2$ policy performs better, as it accounts for the increased risk of offloading failures.

To further assess the impact of the risk-aware behavior, Fig. 9 reports the corresponding average expired task ratio for the same scenario considered in Fig. 8. The expired task ratio is defined as the ratio between the number of tasks that violate their execution deadline (including tasks whose deadline expires while waiting in the queue) and the total number of generated tasks. The observed trends are consistent with the completion ratio results discussed in Fig. 8. In particular, the policy trained with $\lambda = 2$ consistently achieves the lowest expired task ratio over the considered range of remote computational resources, outperforming both the risk-neutral policy ($\lambda = 0$) and the more conservative policies ($\lambda = 5$ and $\lambda = 10$).

These results provide additional evidence that incorporating the return variance into the optimization objective enables the learning of policies that better balance performance and reliability, simultaneously increasing the task completion ratio and reducing deadline violations. Conversely, excessively large values of λ lead to overly conservative behaviors. In particular, the policies trained with $\lambda = 5$ and $\lambda = 10$ tend to favor local execution in order to avoid the uncertainty associated with task offloading. While this strategy reduces the exposure to communication-related risks, it also limits the exploitation of remote computational resources, ultimately resulting in a larger number of expired tasks.

The mismatch reported in Table 3 in the data rate evaluation for BLER $= 0.5$ is large, as such value lies far from the reference BLER target of 0.1. However, this case is used solely to highlight the structural



(a) $J_{\mathcal{Z}}$ versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(b) Q versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(c) α versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(d) J_π versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(e) σ versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(f) $J_{\mathcal{Z}}$ versus training steps and λ. $\gamma = 0.99$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.5, 0.5]$.

(g) Q versus training steps and λ. $\gamma = 0.99$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.5, 0.5]$.

(h) σ versus training steps and λ. $\gamma = 0.99$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.5, 0.5]$.

Fig. 10. Training performance.

behavior of the learned policy under extreme network unreliability, rather than to provide quantitative performance claims.

5.5. Training performance

Finally, we report the training performance when varying the DSAC-T hyperparameters. In particular, we first discuss the general training performance and then we explore the impact of the standard deviation weighting factor λ. For clarity, all reported critic-related metrics are shown for the first critic network, the second critic exhibited similar behavior.

In Fig. 10(a) we report the behavior of the loss function $J_{\mathcal{Z}}$ related to the learned reward distribution, as a function of the number of training steps. It can be observed that $J_{\mathcal{Z}}$ decreases as the training steps increases, except for some spikes that occur every time that a new episode is started. Spikes indeed happen every $2.5e5$ iterations, which is the interval of iterations in which we change the episode parameters. The reason behind these spikes is that by changing the episode parameters, the learning algorithm has first to change the learned policy. Then, also the reward distribution changes according to the new policy and the new system parameters, causing a mismatch between the reward distribution learned until then and the actual reward distribution.

In Fig. 10(b) we report the average of the mean value Q of the reward distribution. Looking at the plot, we can observe big spikes during the first iterations. Indeed, the entropy of the policy, which regulates the relevance of the exploration in the learning process, is the most relevant factor in the entropy-augmented return in Eq. (24) during the first phase. In principle, during the first iterations of the algorithm, less probable actions should be preferred in order to favor exploration. However, at a later stage more importance should be given to policy exploitation, otherwise the learning process might be too slow. As regard the general behavior of Q, we can observe changes of slope along with iterations. Again, as for $J_{\mathcal{Z}}$, these changes happen when there is a change of the episode parameters. A specific setting of episode parameters has a certain maximum reward that can be achieved, that changes from setting to setting. Also, the policy has to be adapted according to the specific setting.

As a further step in our investigation, we show in Fig. 10(c) how the entropy weighting factor is updated during the training process. As already mentioned, during the training process α is updated according to β_α in order to regulate the balance exploration–exploitation. Hence, as the number of iterations increases, α decreases.

In Fig. 10(d) we discuss the behavior of the actor network loss function J_π. As one can observe, the behavior of J_π is similar to Q, except for the opposite sign, since the two quantities are related by Eq. (32). We recall indeed that the objective of the critic network is to maximize the entropy-augmented return function. The choice of reporting J_π with negative values is to show how this function decreases, whereas the mean of the reward increases, as the training process progresses. Due to the similarity between Q and J_π from now on we will show only the results related to the mean of the reward function, when varying the other algorithm hyperparameters.

In Fig. 10(e) we report the average of the standard deviation σ of the reward distribution. We recall indeed that the main characteristic of DistRL is to learn the reward probability distribution rather than only the expected value given by the value (or the Q) function. Similarly to the results shown in the previous figures, we can observe how the slope of the standard deviation varies during the training process for the same reasons we mentioned before.

In Fig. 10(f) we report $J_{\mathcal{Z}}$ as a function of the standard deviation weighting factor λ. Also in this case, we can see that the critic loss function converges as the training process progresses, with spikes happening in correspondence of the change of the episode parameters. Furthermore, we can observe that we get good convergence of the algorithm for all the three values of λ.

A more remarkable effect of λ can be observed by looking at the performance of the average of the reward distribution Q, shown in Fig. 10(g). Results suggest that the performance might benefit from a larger value of λ, provided that it is not too large. Indeed, we can see that the lowest mean of the reward is reached for $\lambda = 10$. Larger values of λ might induce the algorithm to learn too conservative policies in order to keep low the standard deviation of the reward, thus giving less relevance to the mean.

Finally, we show in Fig. 10(h) the related effects of λ on the standard deviation of the reward distribution. As one would expect, in most cases a larger value of λ is associated with a lower values of average standard deviation.

6. Conclusions

In this work, we addressed the challenging problem of task offloading and resource allocation in edge computing systems. In particular, we explored the integration of early exiting mechanisms, demonstrating their potential to dynamically adapt system performance based on task requirements, network conditions, and the availability of remote computational resources, providing an increase of average task completion ratio of up to 212% with respect to classic edge computing systems, under the considered system configuration. To effectively interact with this complex environment, we employed a novel variant of DRL, capable of learning adaptive policies while also quantifying the associated uncertainty. This approach enabled more informed decision-making in highly dynamic scenarios, being able to develop risk-sensitive policies. Our preliminary findings suggest promising directions for understanding and potentially balancing the trade-off between high-reward policies and other policies characterized by low outcome standard deviation. Future work will further explore this trade-off to assess whether it is possible to simultaneously optimize for both performance and robustness. As part of our ongoing research activities, future work will focus on the deployment of EE-enabled AI applications on a real edge-computing testbed available at our institution. This will allow us to validate the proposed TORA-EE framework under realistic operating conditions and to assess practical aspects such as scalability, deployment overhead, and controller complexity. Such experimental activities could be naturally integrated within the context of ongoing research initiatives involving our university, such as the P-CAR project [42], which is developing advanced facilities for the testing and validation of Connected and Automated Vehicle services and edge-enabled intelligent transportation systems.

CRediT authorship contribution statement

Simone Angelucci: Writing – review & editing, Writing – original draft, Validation, Methodology. **Roberto Valentini:** Writing – review & editing, Validation, Supervision. **Marco Levorato:** Writing – review & editing, Visualization, Supervision, Resources, Conceptualization. **Fortunato Santucci:** Writing – review & editing, Visualization, Supervision, Funding acquisition. **Carla Fabiana Chiasserini:** Writing – review & editing, Visualization, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was partly funded by the project "Centre of Excellence on Connected, Geo-Localized and Cyber-secure Vehicles (EX-Emerge)" – Italian Government (Grant Number : CIPE Resolution 70/2017).

Appendix. Error analysis of fixed SNR–MCS mapping across BLER

Using the same SNR-MCS mapping for BLER values different from a reference BLER value $\mathrm{BLER}_{\mathrm{ref}}$ (0.1 in our case) introduces a mismatch that may bias the estimated data rate and, consequently, the offloading performance.

To quantify this effect, we analytically estimate the error induced by this approximation. We first evaluate the SNR shift required to move from $\mathrm{BLER}_{\mathrm{ref}}$ to a generic target $\mathrm{BLER}_{\mathrm{tgt}}$ in order to rely on the same MCS.

For each MCS m, we approximate the BLER curve in the waterfall region by a logistic function

$$\mathrm{BLER}_m(\gamma) = \frac{1}{1+\exp\big(\beta(\gamma-\gamma_m)\big)}, \tag{A.1}$$

where, for notational convenience, we reuse the symbol γ to denote the SNR, γ_m is the SNR offset for MCS m, and β is a parameter capturing the slope of the BLER curve. We assume that all MCSs share the same slope parameter β, while differing only in their SNR offsets γ_m. Under this assumption, the BLER curves form a family of horizontally shifted replicas of a common function.

By inverting Eq. (A.1), the SNR required to achieve a generic target BLER for MCS m is

$$\gamma = \gamma_m + \frac{1}{\beta}\ln\left(\frac{1-\mathrm{BLER}}{\mathrm{BLER}}\right). \tag{A.2}$$

Evaluating this expression at $\mathrm{BLER}_{\mathrm{ref}}$ and at a different target $\mathrm{BLER}_{\mathrm{tgt}}$, the corresponding SNR thresholds satisfy

$$\gamma_m^{(\mathrm{tgt})} = \gamma_m^{(\mathrm{ref})} + \Delta\gamma, \tag{A.3}$$

where

$$\Delta\gamma = \frac{1}{\beta}\ln\left(\frac{\mathrm{BLER}_{\mathrm{ref}}(1-\mathrm{BLER}_{\mathrm{tgt}})}{\mathrm{BLER}_{\mathrm{tgt}}(1-\mathrm{BLER}_{\mathrm{ref}})}\right). \tag{A.4}$$

Importantly, $\Delta\gamma$ does not depend on the MCS index m, due to the equal-slope assumption. Therefore, we assume that changing the BLER target induces a rigid translation of all SNR thresholds by the same quantity $\Delta\gamma$, and the entire SNR–MCS mapping is uniformly shifted along the SNR axis. As a result, the MCS selected under $\mathrm{BLER}_{\mathrm{tgt}}$ at a given SNR γ is equivalently obtained by evaluating the reference mapping at $\gamma+\Delta\gamma$. This property enables a tractable characterization of the rate mismatch induced by reusing an SNR–MCS mapping calibrated at $\mathrm{BLER}_{\mathrm{ref}}$.

It is worth emphasizing that the uniform shift property strictly holds under the equal-slope logistic approximation adopted for analytical tractability. Since the SNR–MCS mapping reported in [35] may exhibit MCS-dependent slope variations, directly shifting the reference thresholds would not necessarily yield an exact mapping for a different BLER target. Therefore, rather than redefining the mapping, we quantify the bias introduced by reusing the reference table, which allows us to assess the robustness of the reported performance results.

There is an error every time that the SNR shift induces a change of MCS. We are then interested in evaluating the mismatch probability conditioned to a particular channel state r_i. Assuming that the instantaneous SNR within channel state r_i is uniformly distributed in the interval $[\gamma_i, \gamma_{i+1}]$, i.e., $\gamma \sim \mathcal{U}(\gamma_i, \gamma_{i+1})$, the mismatch probability conditioned on state i is given by

$$p_{\mathrm{mis}|r_i} = \mathbb{P}\left(\gamma+\Delta\gamma \notin [\gamma_i, \gamma_{i+1}]\right). \tag{A.5}$$

Under the uniform assumption, this probability reduces to

$$p_{\mathrm{mis}|r_i} = \min\left(1, \frac{|\Delta\gamma|}{\gamma_{i+1}-\gamma_i}\right). \tag{A.6}$$

The average relative rate error is then computed as

$$\bar{\Delta R} = \sum_{i=1}^{N_r} \pi_i\, p_{\mathrm{mis}|r_i}\, \Delta R_i, \tag{A.7}$$

where π_i denotes the steady-state probability of being in channel state r_i and ΔR_i is the data rate mismatch associated with channel state i. In our model, consistent with the adopted finite-state wireless channel model, we assume $\pi_i = 1/N_r$. Note that since the only MCS-dependent quantity in Eq. (5) is the spectral efficiency, the relative error reduces to the ratio between the corresponding spectral efficiencies, taken from the reference mapping in [35], and it can be defined as follows:

$$\Delta R_i = \frac{R_i^{(\mathrm{tgt})} - R_i^{(\mathrm{ref})}}{R_i^{(\mathrm{ref})}} = \frac{Q_i^{(\mathrm{tgt})} - Q_i^{(\mathrm{ref})}}{Q_i^{(\mathrm{ref})}}, \tag{A.8}$$

where, for simplicity of notation, with $R_i^{(\mathrm{tgt})}$ and $R_i^{(\mathrm{ref})}$ we indicate the offloading rate when being in channel state i and using the derived target BLER mapping or the reference one, respectively.

Table A.1
Transmission decision error probability (%) induced by using the SNR–MCS mapping calibrated at $\text{BLER}_{\text{ref}} = 0.1$, for different target BLER values and slope parameters β.

β/BLER	0.05	0.0875	0.125	0.1625	0.2	0.5
1.3	5.13	0	0	0	4.33	6.25
1.5	4.45	0	0	0	3.75	6.25
1.8	3.7	0	0	0	3.13	6.25

In order to evaluate $Q_i^{(\text{tgt})}$, we first approximate the instantaneous SNR by its midpoint value within each interval, obtaining $\bar{\gamma}_i$. Then, we exploit the SNR shift $\Delta\gamma$ to compute the shifted SNR

$$\bar{\gamma}_i^{\text{shift}} = \bar{\gamma}_i - \Delta\gamma. \tag{A.9}$$

Let, for notational convenience, reuse the symbol $\{\tau_m\}$ to denote the ordered set of SNR decision thresholds of the reference mapping, such that MCS m is selected when

$$\tau_{m-1} \le \bar{\gamma}_i < \tau_m. \tag{A.10}$$

The MCS consistent with BLER_{tgt} is then obtained as the smallest index $m^\star$ satisfying

$$\bar{\gamma}_i^{\text{shift}} < \tau_{m^\star}. \tag{A.11}$$

This procedure is equivalent to evaluating the reference SNR-MCS mapping at the shifted SNR value, in accordance with the uniform translation property discussed previously. An MCS mismatch occurs whenever $m^\star \neq m$.

To evaluate the impact of using an SNR–MCS mapping calibrated at $\text{BLER}_{\text{ref}} = 0.1$ for different BLER targets, we computed the average relative rate error as in Eq. (A.7). Table 3 reports the average rate error for $\text{BLER} \in [0.05, 0.2]$ and $\text{BLER} = 0.5$, for different values of the BLER slope parameter $\beta \in \{1.3, 1.5, 1.8\}$. The error can be positive or negative, meaning that a negative error implies a higher transmitting rate with respect to the possible one, while a positive error implies that high transmission rates could have been achieved if relying on the correct SNR–MCS mapping. This is consistent with the intuition that when considering more conservative BLER constraints, lower data rates are achieved when considering same SNR values. The opposite can be said instead when relaxing the BLER constraint.

In general, results reported in Table 3 show that for BLER values close to the reference value (e.g., 0.0875 and 0.1250), the average error is 0% for all considered β. In general, for $\text{BLER} \in [0.05, 0.2]$, the mismatch induces an average rate error between approximately 0.3% and 15%, depending on β. For extreme BLER values (e.g., 0.5), the error increases significantly (up to 50%), as expected due to the large SNR shift required.

Table A.1 instead reports the probability of incorrect transmission decisions. When tightening the BLER constraint, some channel states that were previously feasible for transmission may become unfavorable. Conversely, when relaxing the BLER constraint, previously unfavorable channel states may become feasible for transmission, since lower SNR values are sufficient to sustain the transmission. The probability of incorrect transmission decisions is evaluated similarly to the average relative error:

$$P_{\text{tx\_err}} = \sum_{i=1}^{N_r} \pi_i \, p_{\text{mis}|r_i} \, \mathbf{1}_{\left\{(R_i^{(\text{tgt})}>0) \neq (R_i^{(\text{ref})}>0)\right\}}, \tag{A.12}$$

where the indicator function is equal to one whenever the transmission decisions differ each other.

As a concluding remark, the main performance trends and comparative conclusions of this work are drawn within the practically relevant BLER range $[0.05, 0.2]$, where the induced bias remains moderate and does not alter the qualitative behavior of the proposed TORA-EE framework. This analysis confirms that, while the reuse of a fixed SNR–MCS mapping introduces a controlled bias, the reported results remain robust in realistic operating regimes.

Data availability

No data was used for the research described in the article.

References

[1] 3rd Generation Partnership Project (3GPP), Service Requirements for V2X Services , Technical Specification (TS) 22.185, 2016, URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2989. Version 14.0.0.

[2] 5G Automotive Association (5GAA), C-V2X Use Cases: Methodology, Examples and Service Level Requirements, White Paper, 2019, URL: https://5gaa.org/content/uploads/2025/12/c-v2x-use-cases-and-service-level-requirements-vol-i.pdf.

[3] 5G Automotive Association (5GAA), C-V2X Use Cases Volume II: Examples and Service Level Requirements, White Paper, 2020, URL: https://5gaa.org/content/uploads/2020/10/5GAA_White-Paper_C-V2X-Use-Cases-Volume-II.pdf.

[4] E. Cinque, F. Valentini, A. Persia, S. Chiocchio, F. Santucci, M. Pratesi, V2X communication technologies and service requirements for connected and autonomous driving, in: Proc. of AEIT AUTOMOTIVE, Turin, Italy, 2020, http://dx.doi.org/10.23919/AEITAUTOMOTIVE50086.2020.9307388.

[5] L. Andreone, R. Brignolo, S. Damiani, G. Sommariva, G. Vivo, M. Stefano, SAFESPOT Final Report, Technical Report, 2010, Version 1.0.

[6] Y. Matsubara, M. Levorato, F. Restuccia, Split computing and early exiting for deep learning applications: Survey and research challenges, ACM Comput. Surv. 55 (5) (2022) arXiv:2103.04505.

[7] S. Angelucci, R. Valentini, M. Levorato, F. Santucci, C.F. Chiasserini, Edge computing with early exiting for adaptive inference in mobile autonomous systems, in: Proc. of IEEE ICC, Denver, Colorado, USA, 2024, http://dx.doi.org/10.1109/ICC51166.2024.10622411.

[8] A. Ben-Ameur, A. Araldo, T. Chahed, Multiple resource allocation in multi-tenant edge computing via sub-modular optimization, in: Proc. of IEEE ICC, Rome, Italy, 2023, http://dx.doi.org/10.1109/ICC45041.2023.10279675.

[9] S. Birhanu Engidayehu, T. Mahboob, M. Young Chung, Deep reinforcement learning-based task offloading and resource allocation in MEC-enabled wireless networks, in: Proc. of APCC, Jeju Island, South Korea, 2022, http://dx.doi.org/10.1109/APCC55198.2022.9943689.

[10] Z. Cheng, M. Min, Z. Gao, L. Huang, Joint task offloading and resource allocation for mobile edge computing in ultra-dense network, in: Proc. of IEEE GLOBECOM, Taipei, Taiwan, 2020, http://dx.doi.org/10.1109/GLOBECOM42002.2020.9322099.

[11] X. Zhang, Y. Teng, N. Wang, B. Sun, G. Hu, Accelerating deep neural network tasks through edge-device adaptive inference, in: Proc. of IEEE PIMRC, Toronto, Canada, 2023, http://dx.doi.org/10.1109/PIMRC56721.2023.10293996.

[12] W. Fan, Z. Chen, Z. Hao, F. Wu, Y. Liu, Joint task offloading and resource allocation for quality-aware edge-assisted machine learning task inference, IEEE Trans. Veh. Technol. 72 (5) (2023) 6739–6752, http://dx.doi.org/10.1109/TVT.2023.3235520.

[13] P. Lai, Q. He, X. Xia, F. Chen, M. Abdelrazek, J. Grundy, J. Hosking, Y. Yang, Dynamic user allocation in stochastic mobile edge computing systems, IEEE Trans. Serv. Comput. 15 (5) (2022) 2699–2712, http://dx.doi.org/10.1109/SERVICES55459.2022.00030.

[14] L. Li, Q. Guan, L. Jin, M. Guo, Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queueing system, IEEE Access 7 (2019) 9912–9925, http://dx.doi.org/10.1109/ACCESS.2019.2891130.

[15] M. Hao, D. Ye, S. Wang, B. Tan, R. Yu, URLLC resource slicing and scheduling in 5G vehicular edge computing, in: IEEE VTC2021-Spring, [Online], 2021, http://dx.doi.org/10.1109/VTC2021-Spring51267.2021.9448805.

[16] W. Zhang, K. Tuo, Research on offloading strategy for mobile edge computing based on improved Grey Wolf optimization algorithm, MDPI Electron. 12 (11) (2023) http://dx.doi.org/10.3390/electronics12112533.

[17] J. Liu, Y. Wang, W. Zhang, K. Tian, A novel offloading and resource allocation scheme for time-critical tasks in heterogeneous internet of vehicles, in: Proc. of INOCON, Bangalore, India, 2023, http://dx.doi.org/10.1109/INOCON57975.2023.10101035.

[18] J. Jang, K. Tulkinbekov, D.-H. Kim, Task offloading of deep learning services for autonomous driving in mobile edge computing, MDPI Electron. 12 (2023) http://dx.doi.org/10.3390/electronics12153223.

[19] J. Feng, Q. Pei, F.R. Yu, X. Chu, J. Du, L. Zhu, Dynamic network slicing and resource allocation in mobile edge computing systems, IEEE Trans. Veh. Technol. 69 (7) (2020) 7863–7878, http://dx.doi.org/10.1109/TVT.2020.2992607.

[20] Q. Liu, T. Han, VirtualEdge: Multi-domain resource orchestration and virtualization in cellular edge computing, in: Proc. of IEEE ICDCS, Dallas, Texas, USA, 2019, http://dx.doi.org/10.1109/ICDCS.2019.00108.

[21] C. Li, J. Wang, K. Jiang, C. Xiong, S. Wan, MEOCI: Model partitioning and early-exit point selection joint optimization for collaborative inference in vehicular edge computing, IEEE Trans. Parallel Distrib. Syst. 37 (3) (2026) 666–679, http://dx.doi.org/10.1109/TPDS.2026.3652171.

[22] J. Guan, Q. Zhang, I. Murturi, P.K. Donta, S. Dustdar, S. Wang, Collaborative inference in DNN-based satellite systems with dynamic task streams, in: ICC 2024 - IEEE International Conference on Communications, 2024, pp. 3803–3808, http://dx.doi.org/10.1109/ICC51166.2024.10622590.
[23] Z. Liu, J. Song, C. Qiu, X. Wang, X. Chen, Q. He, H. Sheng, Hastening stream offloading of inference via multi-exit DNNs in mobile edge computing, IEEE Trans. Mob. Comput. 23 (1) (2024) 535–548, http://dx.doi.org/10.1109/TMC.2022.3218724.
[24] F. Dong, H. Wang, D. Shen, Z. Huang, Q. He, J. Zhang, L. Wen, T. Zhang, Multi-exit DNN inference acceleration based on multi-dimensional optimization for edge intelligence, IEEE Trans. Mob. Comput. 22 (9) (2023) 5389–5405, http://dx.doi.org/10.1109/TMC.2022.3172402.
[25] T. Niu, Y. Teng, Z. Han, P. Zou, An adaptive device-edge co-inference framework based on soft actor-critic, in: 2022 IEEE Wireless Communications and Networking Conference, WCNC, 2022, pp. 2571–2576, http://dx.doi.org/10.1109/WCNC51071.2022.9771550.
[26] J.-W. Kim, H.-S. Lee, Early exiting-aware joint resource allocation and DNN splitting for multisensor digital twin in edge–cloud collaborative system, IEEE Internet Things J. 11 (22) (2024) 36933–36949, http://dx.doi.org/10.1109/JIOT.2024.3439852.
[27] Z. Liu, Q. Lan, K. Huang, Resource allocation for batched multiuser edge inference with early exiting, in: ICC 2023 - IEEE International Conference on Communications, 2023, pp. 3614–3620, http://dx.doi.org/10.1109/ICC45041.2023.10278846.
[28] Y. Zheng, T. Zhang, X. Mu, Y. Liu, R. Huang, Joint semantic transmission and resource allocation for intelligent computation task offloading in MEC systems, IEEE Trans. Wirel. Commun. 24 (10) (2025) 8756–8770, http://dx.doi.org/10.1109/TWC.2025.3568870.
[29] S. Teerapittayanon, B. McDanel, H.T. Kung, BranchyNet: Fast inference via early exiting from deep neural networks, 2017, arXiv:1709.01686.
[30] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, K.Q. Weinberger, Multi-scale dense networks for resource efficient image classification, 2018, arXiv:1703.09844.
[31] F. Ilhan, K.-H. Chow, S. Hu, T. Huang, S. Tekin, W. Wei, Y. Wu, M. Lee, R. Kompella, H. Latapie, G. Liu, L. Liu, Adaptive deep neural network inference optimization with eenet, 2023, arXiv:2301.07099, URL: https://arxiv.org/abs/2301.07099.
[32] C. Tan, N. Beaulieu, On first-order Markov modeling for the Rayleigh fading channel, IEEE Trans. Commun. 48 (12) (2000) 2032–2040, http://dx.doi.org/10.1109/26.891214.
[33] R.H. Clarke, A statistical theory of mobile-radio reception, Bell Syst. Tech. J. 47 (6) (1968) 957–1000, http://dx.doi.org/10.1002/j.1538-7305.1968.tb00069.x.
[34] A. Goldsmith, S.-G. Chua, Adaptive coded modulation for fading channels, IEEE Trans. Commun. 46 (5) (1998) 595–602, http://dx.doi.org/10.1109/26.668727.
[35] A.K. Thyagarajan, P. Balasubramanian, V. D, K. M, SNR-CQI mapping for 5G downlink network, in: Proc. of IEEE APWiMob, Bandung, Indonesia, 2021, http://dx.doi.org/10.1109/APWiMob51111.2021.9435258.
[36] 3rd Generation Partnership Project (3GPP), User Equipment (UE) Radio Access Capabilities, Technical Specification (TS) 38.306, 2021, URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3193. Version 16.3.0.
[37] M.G. Bellemare, W. Dabney, R. Munos, A distributional perspective on reinforcement learning, in: Proc. of PMLR International Conference on Machine Learning, Sydney, Australia, 2017, URL: https://proceedings.mlr.press/v70/bellemare17a.html.
[38] J. Duan, W. Wang, L. Xiao, J. Gao, S.E. Li, C. Liu, Y.-Q. Zhang, B. Cheng, K. Li, Distributional soft actor-critic with three refinements, IEEE Trans. Pattern Anal. Mach. Intell. 47 (5) (2025) 3935–3946, http://dx.doi.org/10.1109/TPAMI.2025.3537087.
[39] J. Duan, Y. Guan, S.E. Li, Y. Ren, Q. Sun, B. Cheng, Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors, IEEE Trans. Neural Netw. Learn. Syst. 33 (11) (2022) 6584–6598, http://dx.doi.org/10.1109/TNNLS.2021.3082568.
[40] C. Singhal, Y. Wu, F. Malandrino, M. Levorato, C.F. Chiasserini, Resource-aware deployment of dynamic DNNs over multi-tiered interconnected systems, in: Proc. of IEEE INFOCOM, Vancouver, Canada, 2024, http://dx.doi.org/10.1109/INFOCOM52122.2024.10621218.
[41] M. Theile, L. Dirnberger, R. Trumpp, M. Caccamo, A.L. Sangiovanni-Vincentelli, Action mapping for reinforcement learning in continuous environments with constraints, 2024, URL: https://arxiv.org/abs/2412.04327, arXiv:2412.04327.
[42] Radiolabs Consortium, P-CAR: PNT center for automated road-transport, 2021, https://www.radiolabs.it/en/progetti/p-car-2/. (Accessed 23 June 2026).

Simone Angelucci is a Ph.D. Student in ICT at University of L'Aquila. He received his B.S. in Information Engineering and his M.S. in Telecommunications Engineering both at University of L'Aquila. His research interests deal mainly with Edge Computing Systems enhanced by Early Exiting. His research also contributed to Passive Backscattering systems with Non Orthogonal Multiple Access and to the field of Binaural Audio.

Roberto Valentini received the M.Sc. degree in telecommunications engineering and the Ph.D. degree in Information and Communication Technology (ICT) from the University of L'Aquila, Italy, in 2014 and 2018, respectively. He is currently an assistant professor in telecommunications at the University of L'Aquila and a member of the Center of Excellence Ex-EMERGE. His research includes efficient communication paradigms for the Internet of Things, backscattering communications, passive radio-frequency identification systems, battery-powered and battery-free wireless sensor networks.

Marco Levorato joined the Computer Science department at UCIrvine in August 2013. Between 2010 and 2012, He was a post-doctoral researcher with a joint affiliation at Stanford and the University of Southern California working with prof. Andrea Goldsmith and prof. Urbashi Mitra. From January to August 2013, he was an Access post-doctoral affiliate at the Access center, Royal Institute of Technology, Stockholm. He is a member of the ACM, IEEE and IEEE Comsoc society. His research interests are focused on next-generation wireless networks, signal processing, cyber–physical systems, smart city and smart energy systems. He has co-authored over 75 technical articles on these topics, including the paper that has received the best paper award at IEEE GLOBECOM (2012). He completed the PhD in Electrical Engineering at the University of Padova, Italy, in 2009. He obtained the B.S. and M.S. in Electrical Engineering summa cum laude at the University of Ferrara, Italy in 2005 and 2003, respectively. In 2016, he received the UC Hellman Foundation Award for his research on Smart City IoT infrastructures.

Fortunato Santucci (SM IEEE) is a Professor of Telecommunications at University of L'Aquila. He has collaborated with universities worldwide and authored about 300 publications. His research focuses on energy-efficient communications for IoT, vehicular communications, network security, satellite systems, and communication-control co-design for industrial automation. He served as Deputy Director of the Centre of Excellence of Ex-EMERGE on connected vehicles (2019–2025) and is currently Vice-President of the Radiolabs consortium. He has held academic leadership roles, served on IEEE committees, and was an editor for IEEE Transactions on Communications (2000–2013).

Carla Fabiana Chiasserini (Fellow, IEEE) is a Professor at Politecnico di Torino, Italy, and an Affiliated Professor with Chalmers University of Technology, Sweden.