UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Department of Information Engineering, Computer Science and Mathematics

PhD Program in ICT - System Engineering, Telecommunications and HW/SW platforms
XXXVIII cycle

MDP-Based Optimization and Risk-Aware Control of Edge Offloading and Adaptive Inference with Early Exiting for Connected Vehicles

SSD ING-INF/03

PhD Student
Simone Angelucci

Advisor
Dr. Roberto Valentini

Course Coordinator
Prof. Davide Di Ruscio

Co-Advisors
Prof. Fortunato Santucci
Dr. Claudia Rinaldi

A.Y. 2024-2025

Contents

Abstract

Intelligent Transportation Systems (ITS) require mobile nodes such as vehicles increasingly rely on computationally intensive perception pipelines, mostly based on Deep Neural Networks (DNNs), to support safety-critical functionalities such as object detection, collision avoidance, and cooperative driving. As vehicles operate in highly dynamic and resource-constrained environments, meeting stringent latency and accuracy requirements remains a significant challenge.

Edge Computing allows vehicles to offload complex tasks, such as the execution of DNNs, to servers at the edge of the network, thus reducing computing times and energy consumption at the mobile devices. Early Exiting (EE) is an emerging paradigm in deep learning that equips 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 use cases involving task execution in resource-constrained computing and communications environments for connected and automated vehicles (CAVs). In particular, we present a unified framework based on Markov Decision Processes (MDPs) for modeling, analyzing, and optimizing the interaction between early exiting, edge offloading and resource allocation in dynamic vehicular environments.

Specifically, this thesis presents two main contributions. The first contribution focuses on analyzing the performance of the integrated system, demonstrating the benefits of combining early exiting with edge computing. We formulate a control problem that jointly decides the computation location and the DNN exit points, and we obtain an optimal policy by solving a linear program that captures the trade-offs between latency, accuracy, and task discarded rate.

The second contribution extends the considered scenario to a more complex environment, including different mobile users and explicit radio and computational resources allocation. In this setting, we employ Distributional Reinforcement Learning (DistRL), which allows not only to learn adaptive control policies but also to quantify the uncertainty associated with policy outcomes. This approach provides a more robust framework for decision-making under uncertainty, enabling performance guarantees that account for the variability inherent in real-world ITS environments.

Keywords: Edge Computing, Early Exiting, Adaptive Inference, Markov Decision Processes, Distributional Reinforcement Learning

List of Figures

List of Tables

List of Publications

We indicate journal publications with **JX** and conference publications with **CX**. In particular, the following works have arisen from the research presented in this thesis:

1. **J1**: S. Angelucci, R. Valentini, M. Levorato, F. Santucci and C. F. Chiasserini, "Distributional RL for Task Offloading, Resource Allocation and Early Exit Selection at the Edge," TechRxiv. November 25, 2025, doi: 10.36227/techrxiv.176403069.98365132/v1. [SUBMITTED]

2. **C1**: S. Angelucci, R. Valentini, M. Levorato, F. Santucci and C. F. Chiasserini, "Edge Computing with Early Exiting for Adaptive Inference in Mobile Autonomous Systems," ICC 2024 - IEEE International Conference on Communications, Denver, CO, USA, 2024, pp. 2980-2985, doi: 10.1109/ICC51166.2024.10622411.

For completeness, we also list publications related to side research topics explored during the course of these three years.

1. **C2**: S. Angelucci, R. Valentini, P. Di Marco and F. Santucci, "Performance of Battery-free BackCom in Uplink NOMA Systems with Joint Detection," 2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2023, pp. 1250-1255, doi: 10.1109/CCWC57344.2023.10099336.

2. **C3**: S. Angelucci, C. Rinaldi, F. Franchi, F. Graziosi, "Comparison of ML Solutions for HRIR Individualization Design in Binaural Audio," In: Barolli, L. (eds) Advanced Information Networking and Applications. AINA 2023. Lecture Notes in Networks and Systems, vol 655. Springer, Cham. https://doi.org/10.1007/978-3-031-28694-0_25

Introduction

0.1 Context

Intelligent Transportation Systems (ITS) are reshaping the way mobility is conceived. They integrate advanced sensing, communication, and computing technologies into the transportation ecosystem to improve safety, efficiency, and coordination among vehicles and infrastructure. In the vision of next-generation mobility, ITS enable a transportation network that is automated, connected, and highly dynamic, where vehicles continuously exchange information with each other and with roadside infrastructure to enhance situational awareness, prevent accidents, and optimize traffic flows.

ITS comprise a broad set of hardware and software components that collectively acquire, exchange, and process the data required to improve mobility. An overview of the main technologies involved is provided in Fig. 1. At the hardware level, *Sensors & Cameras* are essential for perceiving the vehicle's surroundings and supporting local awareness. As modern transportation systems rely on ubiquitous connectivity, ITS also include a wide variety of *Internet of Things (IoT) devices* that enable vehicles, infrastructure, and roadside units to communicate seamlessly. The data gathered by these devices must be processed by intelligent software modules, most notably *Artificial Intelligence (AI)–based algorithms*, which extract high-level information, support perception, and enable decision making.

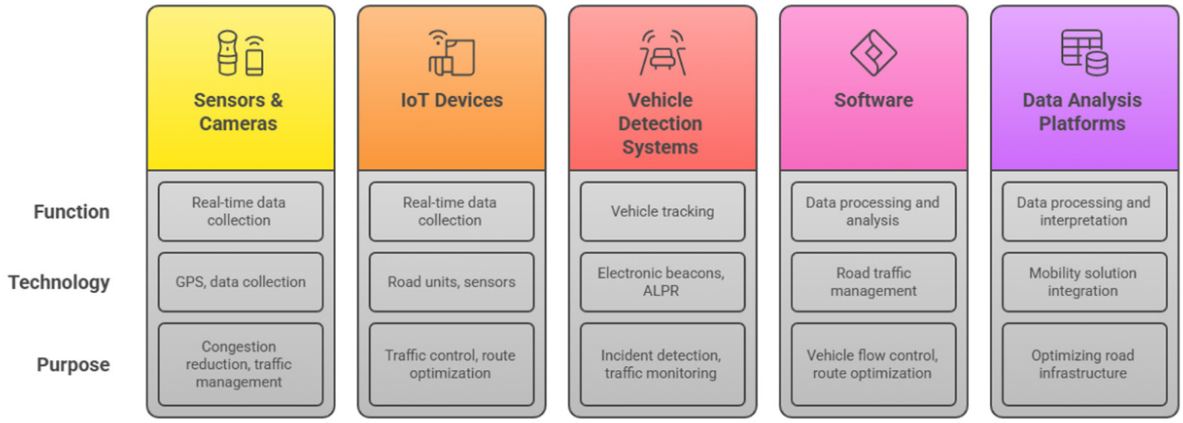


Figure 1: Main components of Intelligent Transportation Systems (ITS) [22].

Finally, although not explicitly highlighted in the figure, the entire ITS ecosystem critically relies on a robust *wireless communication infrastructure*. This infrastructure interconnects all components and supports the continuous flow of information required for cooperative, real-time, and

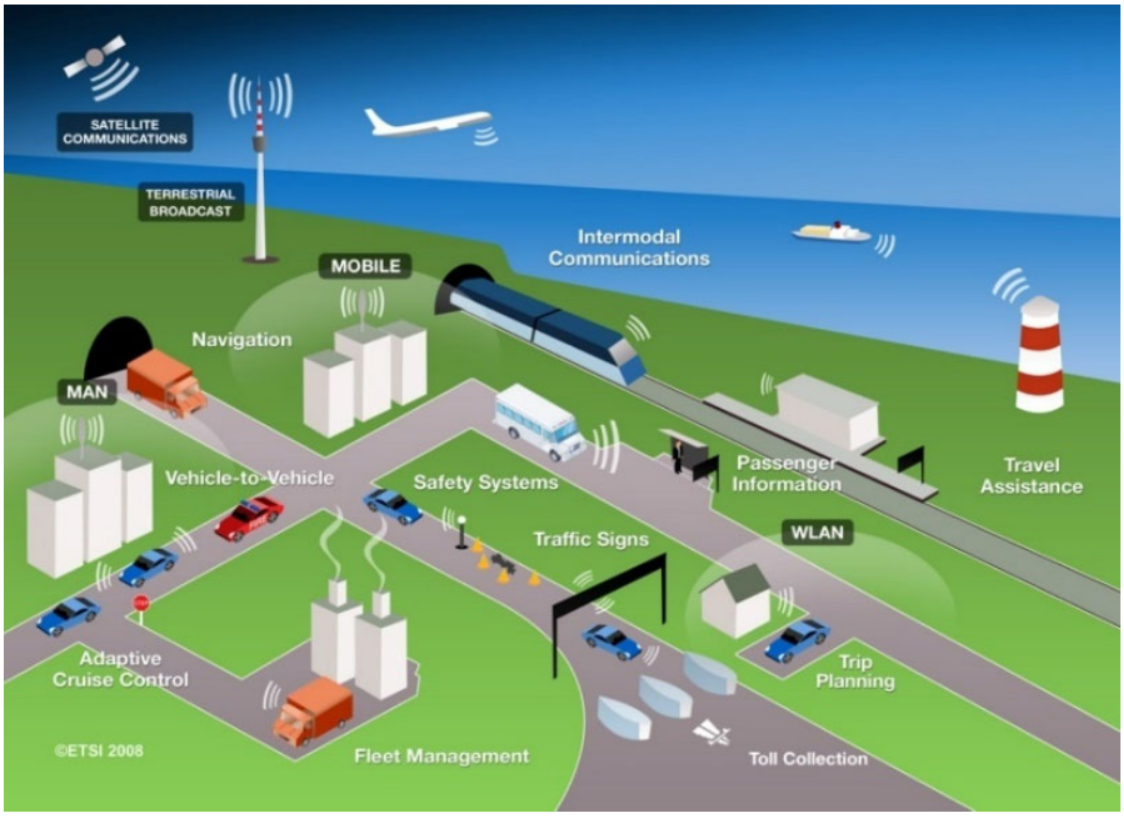


Figure 2: Cooperative-Intelligent Transportation System (C-ITS) [27].

safety-critical functionalities. In particular, it enables the development of *Cooperative ITS (C-ITS)*, where vehicles and infrastructure share their local observations to build a richer, collective situation awareness. Such awareness allows decisions to be made not only on the basis of individual sensor data, but by leveraging information exchanged within the network, ultimately improving safety and coordination across the transportation system. A typical scenario of cooperative information exchange relying on the wireless infrastructure is reported in Fig. 2.

Among the wireless infrastructures, an important enabling architecture is *Edge Computing*. By deploying computation and storage capabilities at the network edge, typically within Roadside Units (RSUs) or nearby infrastructure, edge computing brings processing resources closer to vehicles. This proximity significantly reduces communication latency and allows safety-critical tasks, such as perception, prediction, or decision making, to be executed in near real time. Within C-ITS, edge nodes also act as aggregation points for data coming from multiple vehicles, supporting cooperative perception and enhancing overall situation awareness. As a result, edge computing plays a central role in empowering intelligent, collaborative, and low-latency services in next-generation transportation systems.

0.2 Motivation

Safety-critical ITS applications demand stringent latency, reliability, and accuracy guarantees. Yet achieving these guarantees is difficult due to the following factors:

1. **Computational Burden:** State-of-the-art DNNs deliver high accuracy but require significant processing time, which may exceed the capabilities of embedded vehicular hardware, especially under concurrent workloads, or cannot cope with stringent latency requirements.

2. **Wireless Channel Variability:** Communication links in vehicular environments exhibit rapid fluctuations that can degrade offloading performance unpredictably. This variability fundamentally challenges classical optimization approaches that rely on average-case assumptions.

3. **Remote Resources Availability:** Access to edge or cloud servers is not always guaranteed. Remote nodes may experience contention, dynamic workload variations, or temporary unavailability due to mobility-induced handovers or network congestion. Such fluctuations can significantly impact both the expected inference delay and service reliability, making static offloading strategies ineffective.

Collectively, these factors underscore the need for adaptive, context-aware mechanisms capable of balancing local computation and remote offloading while providing formal performance guarantees under uncertainty.

Within this context, edge offloading allows vehicles to transfer computationally intensive tasks to nearby computing nodes, potentially reducing inference latency and expanding perception capabilities beyond the limits of onboard hardware. The performance of edge offloading, however, depends on network conditions, available bandwidth, server load, and mobility-induced handovers.

In parallel, *adaptive inference* techniques such as *Early Exiting* (EE) provide a complementary strategy to reduce computational cost. In a DNN with multiple exit points, inference can terminate at an intermediate layer if the prediction confidence is sufficiently high, thereby saving computation time and energy. EE enables a trade-off between latency and accuracy, and can be used to dynamically adjust computation based on both input complexity and available resources.

By combining these two mechanisms, vehicles can leverage a flexible inference pipeline that adapts to both local and remote computational capabilities. EE allows elasticity, while edge offloading provides scalable remote resources. Together, they enable dynamic allocation of computation in response to fluctuating workloads, network conditions, and environmental complexity. In particular, the tight coupling between adaptive DNN inference and edge computing represents a promising yet *poorly explored* direction.

Nevertheless, the benefits of such integration critically depend on the highly stochastic nature of the vehicular environment, where wireless channels fluctuate rapidly due to mobility, Doppler effects, and blockage phenomena, and where the availability of radio and computational resources is uncertain and time-varying. The resulting variability makes the coordination of EE and edge offloading a highly non-trivial decision-making problem.

Markov Decision Processes (MDPs), and MDP-based optimization frameworks, offer a natural and effective way to model such systems because they explicitly capture the stochastic evolution of the environment through states, probabilistic transitions, and reward structures. In particular, MDPs can represent temporal dependencies, channel variability, fluctuations in computational load, and the long-term consequences of both offloading and early-exit decisions. This modeling capability enables the design of adaptive policies that dynamically react to the current system state, maximizing performance and reliability even under partial information or noisy observations.

Importantly, in safety-critical applications, optimizing only the expected performance is insufficient. Traditional MDP solutions focus on mean outcomes, whereas robust ITS systems require risk-aware optimization techniques capable of shaping the entire return distribution.

These observations motivate a unified modeling and optimization framework that:

- integrates EE with edge offloading, enabling flexible inference pipelines that adapt their execution based on real-time conditions;
- jointly accounts for the stochastic nature of wireless channels, computational workloads, and resource availability;
- incorporates risk-aware decision-making criteria to ensure performance guarantees aligned with the stringent requirements of safety-critical ITS applications.

This thesis aims to address the observations by developing MDP-based optimization methods that explicitly model the joint behavior of EE and edge offloading in dynamic vehicular environments, ultimately enabling perception systems that are both fast and reliable under real-world uncertainties.

0.3 Contributions

Our work focuses on exploring the benefits of integrating the EE paradigm into edge computing systems. As a reference scenario, we consider CAVs that need to perform latency-sensitive inference tasks, which can either be executed locally or offloaded to an edge server.

We model the considered system through a MDP framework, which allows us to capture the stochastic nature of the underlying environment. In particular, wireless channel dynamics, the availability of radio and computational resources, and task generation are inherently non-deterministic phenomena that can be effectively approximated through Markovian dynamics.

In this dissertation, we discuss two of our works, whose main contributions are summarized as follows.

Our first work investigates the problem of selecting the optimal inference execution strategy in an edge computing scenario for CAVs. We develop an MDP framework to determine the best inference execution policy. We assume that the CAV runs a lightweight DNN without early exits, while the edge server can exploit EE mechanisms, enabling a more flexible and effective inference strategy. Based on this framework, we formulate an optimization problem aimed at maximizing inference accuracy while satisfying application-level constraints on inference latency and task dropping rate. Finally, we evaluate the resulting optimal policy under different system configurations and operating conditions, including variations in channel quality, network congestion, and edge server load. The results are compared with an optimal policy that does not employ early exits, showing that the proposed approach can achieve performance improvements of up to 11%.

Our second work, differently from the first one, addresses the joint optimization of inference execution and resource management in edge intelligence systems employing EE. We formulate a novel Task Offloading and Resource Allocation (TORA) problem that explicitly integrates early-exit selection with task offloading decisions and radio and computational resource allocation. We refer to this problem as TORA-EE. This unified formulation enables flexible optimization of edge

intelligence systems under resource constraints, extending prior works that consider early exits only in conjunction with a subset of these decisions.

We then introduce a system model that captures key dynamics often simplified in the literature. In particular, the radio interface is modeled using mechanisms aligned with 5G systems, including temporally correlated wireless channels, adaptive modulation and coding schemes, and Block Error Rate (BLER) constraints. To solve the resulting optimization problem, we adopt a Distributional Deep Reinforcement Learning (DistRL) approach that learns the full distribution of long-term returns rather than only their expected value. This enables the design of risk-aware control policies capable of handling critical events such as queue overflows, latency violations, and task drops, by learning conservative policies that reduce performance variability.

Finally, extensive simulation results demonstrate the benefits of integrating early exits within the TORA framework. In particular, early exiting allows the system to adapt inference accuracy and execution latency to time-varying channel conditions, resource availability, and heterogeneous service requirements, achieving improvements of up to 212% in terms of successfully completed tasks compared to conventional edge computing architectures.

0.4 Thesis Organization

The dissertation is structured into two main parts. Part I (Chapters 1–3) provides background, modeling foundations, and enabling technologies, while Part II (Chapters 4–5) presents the original research contributions. In particular:

- **Chapter 1** provides an overview of ITS and their latency-sensitive use cases. In particular, it reviews and compares representative use cases and performance requirements defined by major organizations, including 3GPP, 5GAA, and the EMERGE-NAVIGAZIONE project, with particular emphasis on scenarios characterized by stringent end-to-end latency constraints that motivate the solutions investigated in this thesis.

- **Chapter 2** presents the technological foundations of adaptive neural inference through EE, along with Distributional Reinforcement Learning (DistRL) as a modern paradigm for risk-aware sequential decision making. Their relevance to the investigated system is highlighted.

- **Chapter 3** introduces the stochastic models and decision frameworks employed throughout the thesis. It covers discrete-time Markov chains, Finite-State Markov chains (FSMC) models for vehicular wireless links, and classic results related to MDPs-based optimization.

- **Chapter 4** constitutes the first major contribution of the thesis. It develops an MDP-based optimization framework for real-time edge offloading and adaptive inference via EE. The model incorporates realistic system dynamics, such as task generation, server load and wireless channel evolution, based on FSMCs, and evaluates optimal policies under different constraints and system configurations. Performance improvement with respect to classic edge computing system is also highlighted.

- **Chapter 5** presents the second main contribution: a risk-aware control framework using DistRL for adaptive inference, task offloading and resource allocation, showing the interesting

interplay between early exit decisions and resource allocation. Comparative analyses highlight the improvements over expectation-based optimization. Finally, the chapter demonstrates how controlling the return distribution yields safer and more reliable vehicular perception pipelines under critical system conditions.

The thesis concludes by summarizing the main findings and outlining future directions for integrating risk-aware learning and adaptive edge intelligence in emerging vehicular networks.

Part I

Context & Background

Chapter 1

Intelligent Transportation Systems and related Latency-sensitive Use Cases

1.1 Introduction

The rapid evolution of vehicular technologies, combined with the increasing availability of powerful sensing, computation, and communication platforms, is driving the transition toward *Intelligent Transportation Systems* (ITS). ITS aim to improve road safety, traffic efficiency, and user experience by enabling vehicles, infrastructures, and other road users to cooperate and exchange information in real time. This vision relies on the integration of advanced perception, decision-making, and communication technologies capable of supporting highly dynamic and safety-critical operations.

Modern vehicles are increasingly equipped with heterogeneous sensors (e.g., cameras, LiDARs, radars), onboard computing units, and connectivity modules. However, the information available to a single vehicle remains inherently limited by its own sensing range and occlusions in the environment. To overcome these limitations, ITS promote a *cooperative* paradigm, in which vehicles and infrastructures share situational awareness and intentions, enabling a more accurate, predictive, and robust understanding of the surrounding environment.

Cooperative ITS applications include collision avoidance, coordinated lane merging, platooning, cooperative perception, remote driving, and many others. Most of these applications are characterized by stringent *latency*, *reliability*, and *availability* requirements, typically in the range of a few tens of milliseconds or even below. Achieving such performance imposes significant constraints on both communication protocols and onboard processing pipelines.

1.1.1 V2X Communication Paradigms

Vehicle-to-Everything (V2X) communication is the technological backbone of ITS. It encompasses several communication modes, depending on the involved entities:

- **Vehicle-to-Vehicle (V2V):** direct exchange of safety messages (e.g., position, speed, trajectory) to enable cooperative driving and collision avoidance.
- **Vehicle-to-Infrastructure (V2I):** communication with Roadside Units (RSUs) or traffic management systems to obtain traffic signal information, warnings, or cooperative guidance.

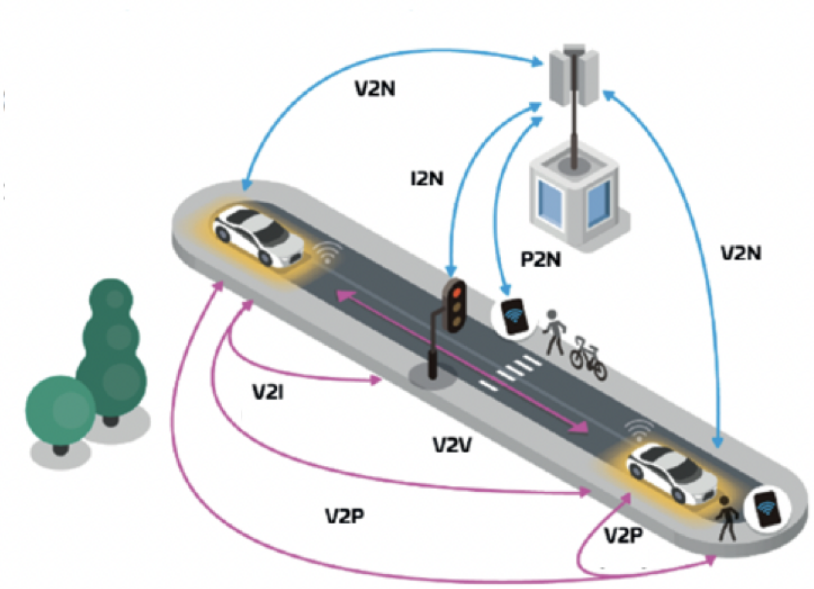


Figure 1.1: Example of V2X Communications [6].

- **Vehicle-to-Network (V2N):** communication with cloud-based or operator-managed services enabling large-scale traffic optimization, remote assistance, or high-level coordination.
- **Vehicle-to-Pedestrian (V2P):** communication with vulnerable road users (e.g., pedestrians, cyclists) equipped with portable devices.

A representative illustration showing the differences between such communication modes is reported in Fig. 1.1.

Each communication mode addresses different classes of ITS applications and is supported by distinct technologies. Historically, Dedicated Short-Range Communication (DSRC/IEEE 802.11p) was the first enabler of vehicular connectivity. More recently, the 3rd Generation Partnership Project (3GPP) introduced the Cellular V2X (C-V2X) family of technologies (LTE-V2X and NR-V2X), offering improved reliability, scalability, and the ability to complement direct communication with network-assisted services.

1.1.2 Scope of This Chapter

Many ITS applications are latency-critical: they require fast perception, decision-making, and communication to guarantee safety and correct operation. For instance, collision avoidance, platooning control, and cooperative perception often impose stringent end-to-end deadlines.

Meeting such constraints is challenging because the end-to-end latency is influenced by multiple components:

- sensing and data acquisition latency;
- onboard inference and decision-making latency;
- network access and transmission delays;

- remote inference or cloud/edge processing when involved.

This chapter reviews the principal classifications of ITS use cases. Over the past years, multiple organizations have proposed structured taxonomies of use cases for next-generation ITS. Among them, the 3GPP and the 5G Automotive Association (5GAA) have provided the most influential and comprehensive classifications, each accompanied by detailed performance and communication requirements.

In particular, we review the use cases defined by these two bodies, with a particular focus on scenarios that exhibit *stringent latency constraints*, those most relevant to the computational model investigated in this thesis, which combines EE with *edge computing* to enable low-latency, adaptive inference.

We then present the use-case scenarios developed within the EMERGE-NAVIGAZIONE project, which complement and extend the perspectives offered by 3GPP and 5GAA and further highlight application domains where low-latency operation is essential.

1.2 3GPP Use Cases

As part of its standardization efforts for C-V2X, 3GPP has developed an extensive catalogue of use cases for ITS. These use cases aim to support cooperative, connected, and automated mobility through a combination of direct (V2V, V2I, V2P) and network-assisted (V2N) communication modes. Building on the general ITS concepts introduced in Sec. 1.1, this section reviews the 3GPP use cases that are particularly relevant for latency-sensitive cooperative functions.

The initial V2X use case specifications were introduced in [1] for LTE-based V2X and and in [2] for enhanced NR-V2X services. In these documents, each use case is described following a common structure consisting of:

- **Description**, further articulated into:
  - *General*: overall purpose and context;
  - *Pre-conditions*: conditions required for operation;
  - *Service flows*: high-level interaction sequences;
  - *Post-conditions*: conditions to be met at completion;
- **Potential Requirements (PRs)**: functional and communication-related requirements needed to support the use case within the 3GPP system.

However, the use case framework was reorganized and consolidated in [3], which introduces a new grouping of ITS use cases and substitutes the earlier PRs with formalized *Service Requirements (SRs)*. In particular, use cases are grouped according to the following classification:

- **Vehicle Platooning**: vehicles form a coordinated group that travels with very small inter-vehicle gaps. This group imposes strict latency constraints to maintain safe and stable formation;
- **Advanced Driving**: semi- or fully-automated driving in which vehicles share sensor-derived information and planned trajectories to coordinate maneuvers. This group includes many

safety-critical operations with tight reaction times; Latency requirements vary depending on data richness and cooperative perception tasks.

- **Remote Driving**: remote or cloud-assisted vehicle control, especially relevant in constrained or hazardous environments. Requires ultra-low latency and highly reliable uplink/downlink communication;
- **Extended Sensors**: exchange of raw or processed sensor data among vehicles and infrastructure to enhance perception;
- **General**: communication principles and interworking requirements valid across all V2X scenarios.
- **Vehicle QoS Support**: mechanisms to notify applications about changes in expected network quality and to adapt behavior accordingly.

Tab. 1.1 presents the classification of V2X use cases into functional groups as defined in [2], illustrating which use cases belong to each group.

1.2.1 Latency-Critical Use Cases

In the following Tab. 1.2, a selection of 3GPP *scenarios*, which might include one or more use cases, characterized by the most stringent latency requirements is reported, as defined in [3]. Latency requirements for specific scenarios may vary depending on the Level of Automation (LoA) of the vehicles, which ranges from fully manual driving (LoA 0) to fully automated driving without human intervention (LoA 5). It is important to note that the latency values specified by 3GPP refer only to the communication segment and do not account for the end-to-end execution time of the entire scenario.

1.3 5GAA Use Cases

The 5GAA is a global, cross-industry consortium uniting stakeholders from the automotive, telecommunications, and technology sectors. Its mission is to design end-to-end solutions for future mobility by leveraging C-V2X technologies for safer,greener, and more efficient transportation systems. This section introduces the organizational structure of 5GAA, its use case methodology, the taxonomy adopted for grouping V2X services, and finally the subset of use cases most relevant for latency-sensitive ITS applications.

To coordinate its technical activities, 5GAA operates through seven Working Groups (WGs):

- **WG1:** Use Cases and Technical Requirements
- **WG2:** System Architecture and Solution Development
- **WG3:** Evaluation, Testbeds and Pilots
- **WG4:** Standards and Spectrum
- **WG5:** Business Models and Go-To-Market Strategies

3GPP Use Cases	
Use case group	**Use case name**
Platooning	eV2X support for Vehicle Platooning Information exchange within platoon Automated Cooperative Driving for Short distance Grouping Information sharing for limited automated platooning Information sharing for full automated platooning Changing Driving-Mode
Advanced Driving	Cooperative Collision Avoidance (CoCA) Information sharing for limited automated driving Information sharing for full automated driving Emergency Trajectory Alignment Intersection Safety Information Provisioning for Urban Driving Cooperative lane change (CLC) of automated vehicles 3D video composition for V2X scenario
Remote Driving	eV2X support for Remote Driving Teleoperated Support (TeSo)
Extended Sensor	Automotive: Sensor and State Map Sharing Collective Perception of Environment Video data sharing for automated Driving
General	Communication between vehicles of different 3GPP Radio Access Technologies (RATs) Multi-PLMN environment Use case on Multi-RAT Use case out of 5G coverage Dynamic Ride Sharing Tethering via Vehicle Proposal for secure software update for electronic control unit
Vehicle QoS Support	QoS aspect of vehicles platooning QoS aspects of advanced driving QoS aspects of remote driving QoS Aspect for extended sensor Different QoS estimation for different V2X applications

Table 1.1: Classification of 3GPP use cases into functional groups [2].

- **WG6:** Regulatory and Public Affairs
- **WG7:** Security and Privacy

Among these, **WG1** plays a central role in defining the envisioned V2X services and their technical requirements.

Over the years, WG1 has produced multiple volumes of use cases and three roadmaps. To ensure consistency across these contributions, 5GAA adopts a structured methodology for describing use cases, explicitly linking operational scenarios to technical and performance requirements. Rather than focusing solely on communication aspects, this methodology captures the entire execution of a use case, from the involved actors and their interactions to the resulting service-level constraints.

Each 5GAA use case is defined through a common template [5] that includes a high-level de-

3GPP Latency-Critical Scenarios			
Scenario	**Description**	**Latency (comm.)**	**Category**
Cooperative Driving	Exchange of vehicle status, position, speed, and planned trajectories in a high-automation platoon to maintain safe inter-vehicle distances and coordinated maneuvers	10-25 ms	Platooning
Information Sharing	Periodic and event-driven reporting between vehicles and RSU for platoon coordination, including speed adjustments, joining/leaving platoon, and leader-follower updates	20 ms	Platooning
Cooperative Collision Avoidance	Real-time exchange of position, speed, acceleration, and predicted trajectories among nearby vehicles to prevent imminent collisions	10 ms	Advanced Driving
Cooperative Lane Change	Coordination of lane-change maneuvers using vehicle-to-vehicle messaging to ensure safe gaps and collision-free transitions	10-25 ms	Advanced Driving
Emergency Trajectory Alignment	Rapid alignment of vehicle trajectories during emergency maneuvers, sharing high-frequency motion and intent data to avoid collisions	3 ms	Advanced Driving
Sensor Information Sharing	Sharing of raw or processed sensor data (e.g., LiDAR, radar) among vehicles and infrastructure to enhance situational awareness and cooperative perception	3-100 ms	Extended Sensors
Video Sharing	Exchange of camera or 3D video streams among vehicles and infrastructure to support cooperative perception, traffic monitoring, or automated driving functions	10-50 ms	Extended Sensors
Control/Information Exchange	Real-time transmission of control commands and status updates between remote operators or cloud servers and the vehicle, enabling teleoperation or assisted driving in hazardous conditions	5 ms	Remote Driving

Table 1.2: 3GPP scenarios with the most stringent latency requirements [3].

scription of the scenario and its objectives; the identification of involved actors (e.g., vehicles, infrastructure, vulnerable road users, remote operators) and their respective roles; the definition of pre-conditions, main and alternative event flows, and post-conditions; and a set of *Service Level Requirements (SLRs)* specifying quantitative performance targets such as latency, reliability, data rate, and update frequency. In addition, information requirements describe the type and granularity of data exchanged among actors, ranging from kinematic states to sensor or perception data.

Similarly to 3GPP, 5GAA organizes its use cases into seven categories, each addressing different aspects of ITS applications:

- **Safety:** Use cases that enhance the safety of vehicles and drivers, including emergency braking, intersection management, assisted collision warning, and lane change support. These apply both to human-driven and autonomous vehicles, sometimes requiring standardization or regulatory alignment to ensure consistent operation across Original Equipment Manufacturers (OEMs);
- **Vehicle Operations Management:** Use cases that provide operational and management value to vehicle manufacturers, such as sensor monitoring, ECU software updates, and remote support. These typically improve maintenance efficiency and fleet management;
- **Convenience:** Use cases that enhance driver or passenger comfort, such as infotainment, cooperative navigation, and autonomous smart parking. While not safety-critical, they provide significant user value and may be monetized directly;
- **Autonomous Driving:** Use cases relevant for high-level automated or self-driving vehicles (LoA 4–5), including full vehicle control, teleoperation (potentially with AR support), dynamic map updates, and cooperative safety features;
- **Platooning:** Use cases related to coordinated vehicle groups, covering platoon formation, position management, distance control, platoon dissolution, and steady-state operation. These benefit transport companies, road operators, and society through improved efficiency and reduced emissions;
- **Traffic Efficiency:** Use cases aimed at optimizing traffic flow and reducing environmental impact, such as Green traffic-Light Optimized Speed Advisory (GLOSA), congestion information, and smart routing. These serve infrastructure providers and can generate commercial or public value;
- **Society and Community:** Use cases that provide public or societal value, including emergency vehicle priority, crash reporting, patient monitoring, and other services for authorities or emergency responders. These are relevant for OEMs offering features to public or private organizations.

These categories cover a wide range of ITS functionalities, from safety-critical exchanges to comfort services and traffic management. Tab. 1.7, reported at the end of the chapter, shows the different use cases grouped by respective categories.

1.3.1 Latency-Critical Use Cases

Tab. 1.3 summarizes the 5GAA use cases, grouped by scenarios, most relevant to our latency-focused analysis, providing their description, latency requirements, and category. In contrast to 3GPP, the latency requirements specified by 5GAA refer to the execution of the entire use case rather than only the communication segment.

Many of the scenarios listed in Tab. 1.3, particularly those involving cooperative driving and shared perception, exhibit very stringent latency requirements that make them strong candidates

5GAA Latency-Critical Scenarios			
Scenario	**Description**	**Latency (e2e)**	**Category**
Coordinated Platoon Driving	Coordinated management of vehicles in a platoon, maintaining safe inter-vehicle distances and synchronized acceleration/braking.	10-100 ms	Platooning
Warning & Collision Avoidance	Exchange of critical information between vehicles and infrastructure to prevent accidents and alert drivers of hazards. Includes all warning and collision avoidance use cases.	100 ms	Safety
Cooperative Maneuvers / Emergency Driving	Exchange of trajectories and intentions among automated vehicles to ensure safety during complex maneuvers or emergency situations.	10-80 ms	Autonomous Driving
Sensor Sharing & Cooperative Perception	Sharing of advanced sensor data (LiDAR, radar, video) among vehicles and infrastructure to enhance environmental awareness and safety.	10-50 ms	Autonomous Driving
Tele-Operated Driving	Real-time remote control of vehicles over the network, including assistance for complex maneuvers or in hazardous environments.	20-100 ms	Autonomous Driving
Cooperative Traffic & Parking	Coordination between vehicles and infrastructure to optimize traffic flow, manage parking, and reduce waiting times.	50-100 ms	Traffic Efficiency / Convenience
High-Bandwidth In-Vehicle Applications	Transmission of high-definition or interactive content within the vehicle, requiring very low latency for a smooth user experience.	20 ms	Convenience
Fleet Monitoring & Vehicle Management	Remote monitoring, software updates, and operational management for fleets or individual vehicles, improving maintenance efficiency and reliability.	20-100 ms	Vehicle Operations Management
Emergency Vehicle Priority & Societal Services	Information exchange to support emergency vehicles, public alerts, or community services.	100 ms	Society and Community

Table 1.3: 5GAA scenarios with the most stringent latency requirements [5, 7, 8].

for distributed processing and edge-computing solutions. Use cases such as *Coordinated Platoon Driving*, advanced sensor sharing, and cooperative maneuvers demand reaction times on the order of a few tens of milliseconds, highlighting the need for highly optimized communication and processing pipelines. Similarly, applications such as tele-operated driving and safety-related services further emphasize the importance of low-latency architectures, emphasizing the central role of computation-adaptive models in ensuring consistent performance in dynamic and potentially congested environments.

1.4 EMERGE Use Cases

The EMERGE (Light Commercial Vehicles & Emerging Technologies for dual-use in "everyday operations" and "emergency") research program was approved by the Inter-ministry Committee for Economic Planning (CIPE) of Italy with act n.70/2017 within the RESTART initiative, aimed at leveraging research and technology to foster economic and social development in L'Aquila and Abruzzo after the 2009 earthquake. A primary objective of EMERGE is to develop and validate advanced applications and services for "smart mobility", with a focus on light commercial vehicles for dual-use scenarios, i.e., last-mile delivery and emergency operations.

In the EMERGE-NAVIGAZIONE project, use cases are defined according to two reference scenarios:

- **Everyday operations:** routine mobility and logistics activities.
- **Emergency operations:** situations requiring urgent response, e.g., disaster or accident scenarios.

EMERGE proposes its own classification of V2X use cases. In particular, it groups use cases with the following categories:

- **Cooperative Safety**;
- **Cooperative Driving and Perception**;
- **Convenience**;
- **Remote Driving**.

Fig. 1.2 shows the EMERGE classification scheme along with a comparison with others categorizations, including 3GPP and 5GAA.

<table>
<tr><th>EMERGE</th><th>3GPP</th><th>ETSI</th><th>5GAA</th><th>5GPPP</th></tr>
<tr><td rowspan="2">Cooperative Safety</td><td rowspan="2">Advanced Driving</td><td rowspan="2">Cooperative road safety</td><td>Safety</td><td rowspan="2">Traffic Safety</td></tr>
<tr><td>Society and Community</td></tr>
<tr><td rowspan="2">Cooperative Driving and Perception</td><td rowspan="2">Extended Sensors</td><td rowspan="2">Cooperative Traffic Efficiency</td><td rowspan="2">Traffic Efficiency and Environmental friendliness</td><td>Cooperative Sensing</td></tr>
<tr><td>Traffic Efficiency</td></tr>
<tr><td>Remote Driving</td><td>Remote Driving</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td rowspan="2">Convenience</td><td rowspan="2">-</td><td>Cooperative local services</td><td rowspan="2">Convenience</td><td rowspan="2">Infotainment Services</td></tr>
<tr><td>Global Internet services</td></tr>
</table>

Figure 1.2: Classification methodologies for V2X use cases [23].

The use cases identified by the EMERGE-NAVIGAZIONE project are reported in Tab 1.4.

1.4.1 Latency-Critical Use Cases

Among the defined EMERGE use cases, those relevant to low-latency ITS applications are summarized in Tab. 1.5.

EMERGE-NAVIGAZIONE Use Cases		
EMERGE Scenario	**Category**	**Use Case**
Everyday Operations	Cooperative Driving and Perception	Electronic Horizon Construction
	Cooperative Driving and Perception	AI Techniques for Efficient Traffic Management
	Cooperative Safety	Assisted Left Turn
	Cooperative Safety	Assisted Intersection Crossing
	Cooperative Safety	Detection and Notification of Vulnerable Road Users (VRU)
Emergency Operations	Cooperative Driving and Perception	Detection and Monitoring of Critical Events
	Convenience	Extended Connectivity in Case of Emergency
	Cooperative Safety	Dynamic Emergency Corridor

Table 1.4: Classification of EMERGE use cases into functional groups.

EMERGE-NAVIGAZIONE Latency-Critical Use Cases			
Use case	Description	Latency (e2e)	Category
Electronic Horizon Construction	Vehicles exchange state information (position, speed, etc.) among neighboring vehicles and with infrastructure to inform each other and the infrastructure of their competence region.	200 ms (V2V)	Cooperative Driving and Perception

Table 1.5: EMERGE-NAVIGAZIONE latency-critical use cases.

1.5 Summary Across Organizations

Tab. 1.6 provides a summary of selected use cases from 3GPP, 5GAA, and EMERGE, highlighting latency-sensitive scenarios and their corresponding V2X communication type, organization, and category. Use cases belonging to the same functional class are color-coded for ease of comparison. As can be seen, latency requirements are heterogeneous among use cases, depending on the criticality of the scenario. For instance, use cases that require remote control of mobile devices, like Tele-Operated Driving, have more stringent latency requirements with respect to use cases where the objective is to update local information on specific geographic areas, such as the Electronic Horizon Construction use case, even though this information have to be provided in a timely manner. We note that the framework developed in this thesis is suited for use cases that require to process information with Deep Neural Networks (DNNs), as could happen in the Sensor Sharing & Cooperative Perception, Electronic Horizon Construction, Warning & Collision Avoidance use cases, consequently covering different ranges of latency requirements.

Latency-Critical V2X Scenarios Across Organizations				
Scenario	**Latency**	**Latency type**	**Category**	**Organization**
Emergency Trajectory Alignment	3 ms	Communication-only	Advanced Driving	3GPP
Cooperative Collision Avoidance	10 ms	Communication-only	Advanced Driving	3GPP
Cooperative Driving	10-25 ms	Communication-only	Platooning	3GPP
Cooperative Lane Change	10-25 ms	Communication-only	Advanced Driving	3GPP
Information Sharing	20 ms	Communication-only	Platooning	3GPP
Sensor Sharing & Cooperative Perception	10-50 ms	End-to-end	Autonomous Driving	5GAA
Tele-Operated Driving	20-100 ms	End-to-end	Autonomous Driving	5GAA
Coordinated Platoon Driving	10-100 ms	End-to-end	Platooning	5GAA
Warning & Collision Avoidance	100 ms	End-to-end	Safety	5GAA
Electronic Horizon Construction	200 ms	End-to-end	Cooperative Driving / Traffic Efficiency	EMERGE-NAVIGAZIONE

Table 1.6: Color-coded summary of latency-critical V2X scenarios across 3GPP, 5GAA, and EMERGE-NAVIGAZIONE. Use cases are first grouped by latency type (communication-only, then end-to-end) and sorted by latency criticality (dark = more stringent, light = relaxed).

1.5.1 Illustrative Example: Electronic Horizon Construction

To provide a concrete illustration of how latency requirements may manifest in cooperative ITS applications, we briefly describe the *Electronic Horizon Construction* concept as defined in the EMERGE-NAVIGAZIONE project. This example is introduced solely to contextualize the previously surveyed standards and is *not* implemented or analyzed as a specific use case in this thesis. The analytical framework developed in the following chapters remains independent of any particular scenario.

The aim of the Electronic Horizon is to supply vehicles and infrastructure with timely and coherent information about surrounding traffic conditions. An illustrative scenario is shown in Fig. 1.3, in which multiple vehicles approach a roundabout. In such contexts, drivers or onboard perception systems may experience occlusions caused by other vehicles or environmental elements, whereas a dynamically constructed horizon containing nearby vehicle states can improve situational awareness and decision making.

Constructing such a horizon involves managing both the amount and the quality of exchanged data, which directly influence communication load and processing requirements. Fundamental information typically includes vehicle states (e.g., position and heading), possibly combined with richer sensor data. As expected, higher data volumes allow for a more precise representation of the local environment, but require more bandwidth and computing resources. Data granularity may also adapt to network and traffic conditions: for instance, congested networks may require transmitting

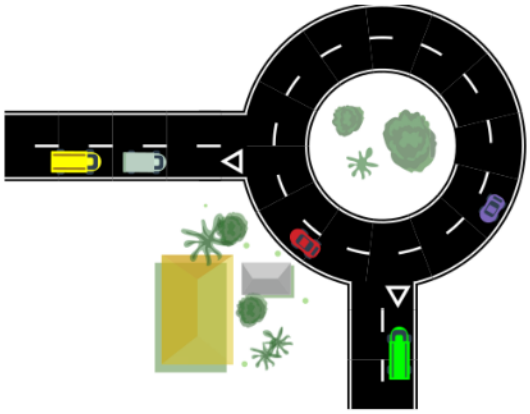


Figure 1.3: Electronic Horizon Construction: example scenario.

reduced or pre-processed information, trading accuracy for stability, while the spatial extent of the horizon can vary with traffic density or application needs.

Overview of Existing Approaches

Although the Electronic Horizon is not central to the analytical contributions of this thesis, it is useful to mention two recent implementations that exemplify how cooperative perception can be structured: the *Local Dynamic Map* (LDM), standardized by ETSI, and the *Electronic Horizon* (eHorizon), defined by the ADASIS forum.

Local Dynamic Maps LDMs were introduced in the SAFESPOT project [9] to aggregate static and dynamic data into a geo-referenced representation accessible to cooperative applications. Information is organized into four layers according to its dynamicity (Fig. 1.4), and standardized in ETSI TR 102 863 [27]. The architecture (Fig. 1.5) comprises a *Data Store* and an *LDM Management* module responsible for updates, queries, and security procedures.

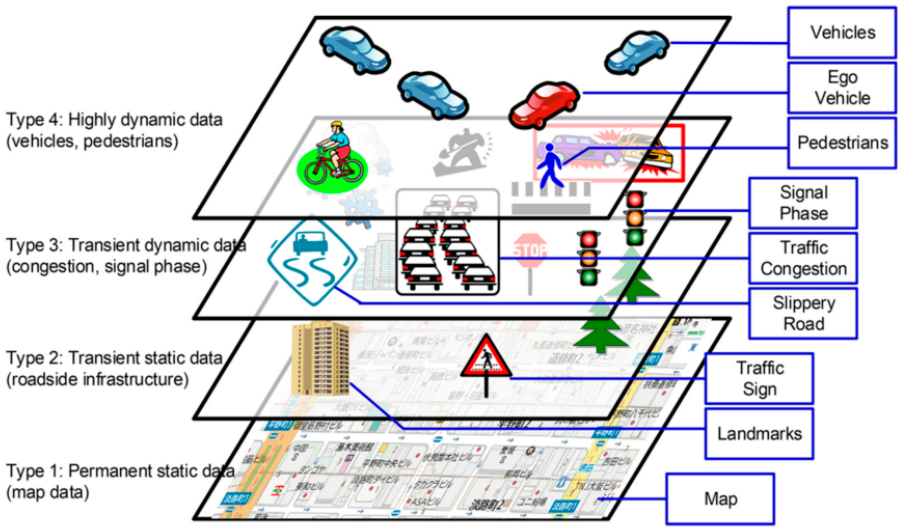


Figure 1.4: LDM data layers [9].

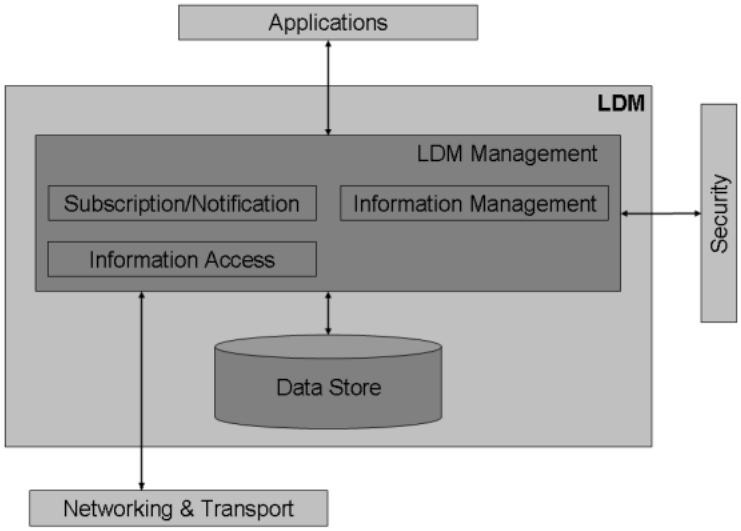


Figure 1.5: LDM Architecture [27].

Recent implementations, such as the Interoperable LDM (iLDM) [31], combine data from heterogeneous sources, including OSM, SUMO, KITTI, and onboard sensors, through a graph database, allowing efficient storage and fast retrieval of dynamic information. A concrete test application, referred to as the Vehicle Discovery Service (VDS), is shown in Fig. 1.6. The goal of this application is to identify all vehicles within a surrounding area of an ego vehicle. The positions of other vehicles and dynamic objects perceived by the ego vehicle are stored in the iLDM and queried in real time to retrieve those within a specific region of interest.

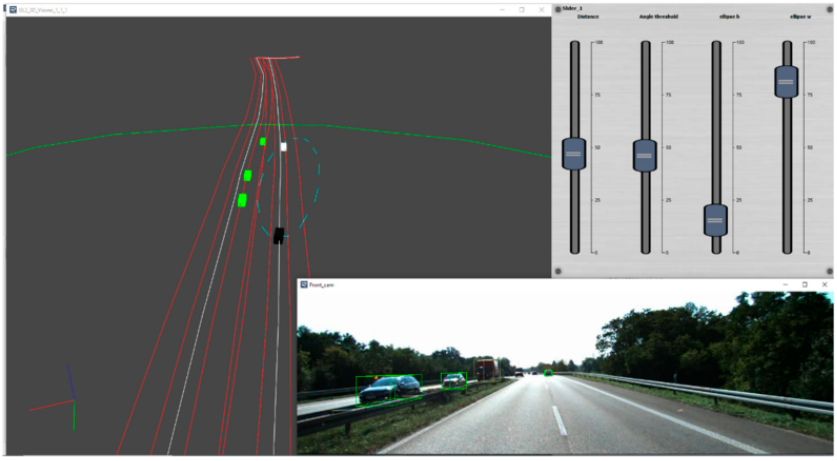


Figure 1.6: iLDM test use case: Vehicle Discovery Service [31].

Electronic Horizon The ADASIS eHorizon provides a complementary approach, focusing on road-segment and trajectory information extending beyond the vehicle's immediate field of view [55]. The architecture, reported in Fig. 1.7, consists of an *ADAS Horizon Provider*, which extracts and encodes relevant map data, and an *ADAS Horizon Reconstructor*, which rebuilds the eHorizon for consumption by ADAS applications.

An example of an eHorizon provider implementation is the *Information Manager* prototype

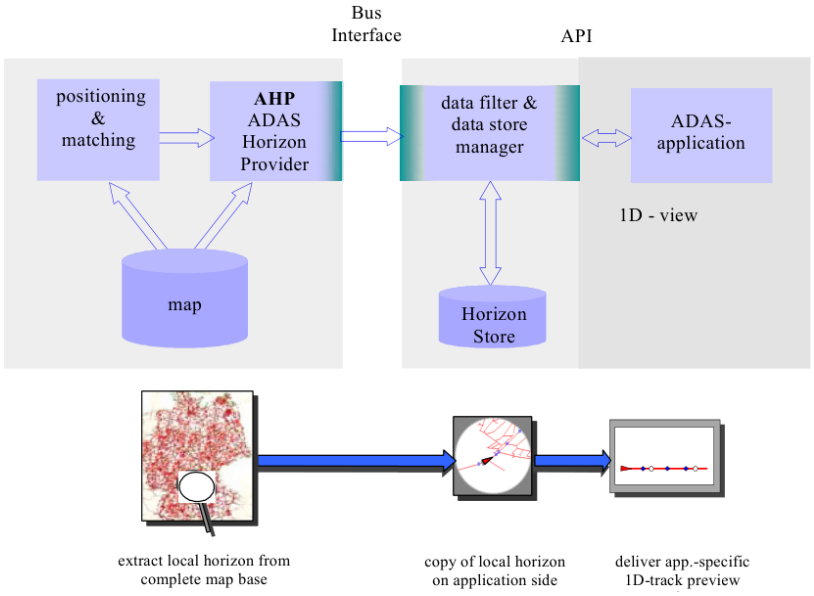


Figure 1.7: ADASIS Functional Architecture [55].

by Ford Research [55], which combines map, sensor, and route-prediction modules to produce an adaptive horizon for ADAS functions (Fig. 1.8).

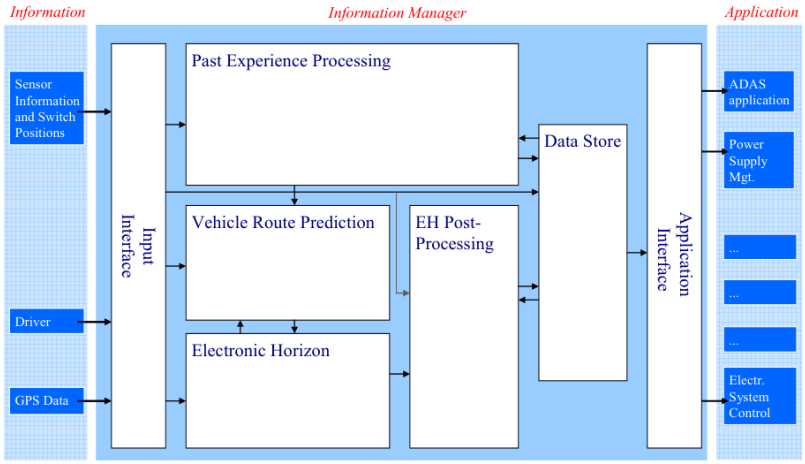
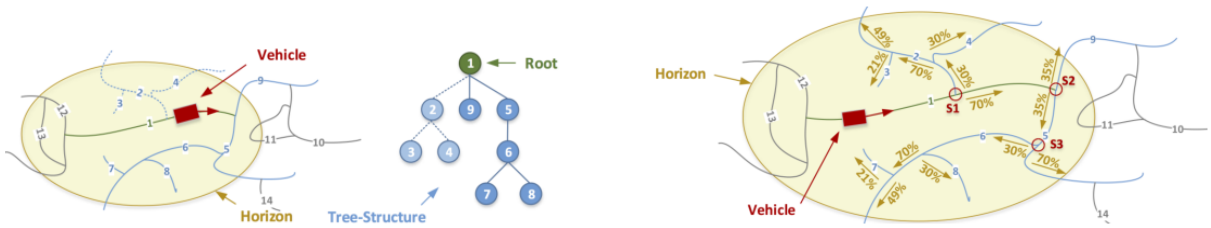


Figure 1.8: Information Manager architecture [55].

The eHorizon is usually represented as a probabilistic tree of road segments ahead of the vehicle [16]–[17] (Figs. 1.9a–1.9b), with the *Most Probable Path* (MPP) estimated from contextual and historical data. Depending on road type and available computational resources, the horizon can be compact (urban contexts) or extensive (highways), and remote implementations can adjust their output based on bandwidth availability.

Overall, the Electronic Horizon example shows how a generic ITS use case can be concretely realized starting from a high-level functional description. Existing implementations reveal a recurring architectural approach in which heterogeneous information sources, including onboard sensors, digital maps, and cooperative messages, are integrated and processed. In such systems, design choices are strongly influenced by latency constraints, data volume, and computational requirements, which in turn affect data aggregation strategies, processing distribution, and the level of



(a) Tree associated to the eHorizon.

(b) Transition probabilities at intersections.

Figure 1.9: Representation of the eHorizon structure and associated transition probabilities [16].

information granularity. Although the specific solutions may differ across implementations, these examples illustrate how standardized ITS use cases can be effectively mapped onto practical system architectures suitable for real-world deployment.

1.6 Conclusions

In this chapter, we have presented a comprehensive overview of reference use cases for cooperative ITS, focusing on latency-critical applications. We reviewed the principal classifications and requirements defined by 3GPP, 5GAA, and the EMERGE-NAVIGAZIONE project, highlighting scenarios where vehicles and infrastructure must exchange information in real time to support safety, traffic efficiency, and cooperative perception.

Among these use cases, *Electronic Horizon Construction* emerged as a paradigmatic example of a latency-sensitive, computation-intensive application. Constructing and maintaining an accurate electronic horizon requires the continuous collection, transmission, and processing of vehicle states and sensor data. The fidelity of the resulting map directly affects cooperative driving performance, especially in scenarios with occlusions or dense traffic, such as roundabouts or urban intersections.

These considerations motivate the integration of EE and edge computing paradigms. Together, these methods can enable real-time processing even under constrained network conditions or heavy computational loads, directly addressing the latency and reliability requirements identified across EMERGE-NAVIGAZIONE, 3GPP, and 5GAA use cases.

5GAA Use Cases	
Use case group	**Use case name**
Safety	Cross-Traffic Left-Turn Assist Intersection Movement Assist Emergency Brake Warning Traffic Jam Warning and Route Information Hazardous Location Warning Lane Change Warning Vulnerable Road User Forward Collision Warning Control Lost Warning Emergency Vehicle Warning Speed Limit Warning Do Not Pass Warning Abnormal Vehicle Warning Signal Violation Warning Traffic Sign In Car Road Work Warning
Vehicle Operations Management	Software Update / Software Update of Reconfigurable Radio System Vehicle Health Monitoring
Convenience	In-Vehicle Entertainment (IVE) – High-Definition Content Delivery, On-line Gaming and VR Automated Vehicle Marshalling – Joint Authentication and Proof of Localisation Automated Vehicle Marshalling (Wake Up) Awareness Confirmation Cooperative Curbside Management Cooperative Lateral Parking Obstructed View Assist Vehicle Decision Assist Decentralised Parking-Lot Management
Autonomous Driving	High Definition Sensor Sharing See-Through for Passing Automated Driving Automated Intersection Crossing Autonomous Vehicle Disengagement Report Cooperative Lane Merge Cooperative Manoeuvres of Autonomous Vehicles for Emergency Situations Data sharing of dynamic objects Non-analysed Sensor Signal Sharing Automated Valet Parking (AVP) Infrastructure-Based Tele-Operated Driving Remote Automated Driving Cancellation Tele-Operated Driving Tele-Operated Driving Support Tele-Operated Driving for Automated Parking
Platooning	Platoon Management Group Formation Distance Control Vehicles Platooning in Steady State
Traffic Efficiency	Speed Harmonisation Traffic Light Optimised Speed Advisory Continuous Traffic Flow via Green Lights Coordination Bus Lane Sharing Request Bus Lane Sharing Revocation Yielding Right-of-Way to VRUs Group Start
Society and Community	Vulnerable Road User Protection Emergency Vehicle Approaching Traffic Light Priority Accident Report Patient Transport Monitoring

Table 1.7: Classification of 5GAA use cases into functional groups [5, 7, 8].

Chapter 2

Recent Advances in Deep Learning

2.1 Introduction

In the previous chapter, we presented the application context of this thesis, focusing on the latency-sensitive use cases for ITS. Deep Neural Networks (DNNs) offer a promising approach to process the rich, heterogeneous data needed for most of the ITS use cases. Such models can integrate vehicle sensor data and historical traffic information to predict future vehicle states and road conditions along the most probable paths. However, deploying these networks in real-time ITS scenarios introduces computational and latency challenges, particularly in edge computing environments where resources are limited and decisions must be made promptly.

This chapter provides the theoretical background necessary to understand the two main components of our approach: Early Exiting (EE) and *Distributional Reinforcement Learning* (DistRL). EE equips neural networks with auxiliary classifiers, allowing inference to terminate at intermediate layers when predictions are sufficiently confident, thus reducing computation and latency. DistRL is then employed to control these early-exit decisions, optimizing the trade-off between inference speed, accuracy, and reliability under dynamic and uncertain conditions.

The first part of the chapter introduces DNNs and the supervised classification setting, followed by a discussion of EE mechanisms. We review foundational contributions, analyze design principles behind modern EE architectures, and discuss their advantages and limitations.

The second part introduces Reinforcement Learning (RL) and its distributional extension. While classical RL focuses on maximizing the expected cumulative reward, this expectation-based view ignores environmental variability and uncertainty. DistRL addresses this by modeling the full probability distribution of future returns, allowing *risk-sensitive* decision-making.

2.2 Early Exiting

2.2.1 Deep Neural Networks and the Supervised Classification Problem

DNNs have become the dominant approach for solving supervised learning tasks such as image classification, speech recognition, and natural language understanding. A DNN is a parametric function $f_\theta : \mathbb{R}^d \to \mathbb{R}^{|\mathcal{C}|}$ composed of multiple layers of linear and nonlinear transformations, where θ collectively denotes all learnable parameters. Given an input sample $\mathbf{x} \in \mathbb{R}^d$, the network produces a vector of logits $\mathbf{z} = f_\theta(\mathbf{x})$, which is typically converted into class probabilities through the softmax

function:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}), \qquad \hat{y}_c = \frac{\exp(z_c)}{\sum_{c' \in \mathcal{C}} \exp(z_{c'})}, \tag{2.1}$$

where $\mathcal{C}$ denotes the set of possible labels.

In supervised classification, the goal is to learn parameters θ such that the predicted distribution $\hat{\mathbf{y}}$ matches the ground-truth one-hot label $\mathbf{y}$ for every training sample. This is usually achieved by minimizing the cross-entropy loss:

$$\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{c \in \mathcal{C}} y_c \log(\hat{y}_c). \tag{2.2}$$

Optimization is performed on a dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ using stochastic gradient descent (SGD) or one of its variants.

Although DNNs attain state-of-the-art accuracy in many benchmarks, their high computational cost poses significant challenges. Modern architectures such as ResNet, DenseNet, and Vision Transformers consist of dozens or even hundreds of layers, and must process every input through the full sequence of computations to obtain a prediction. This results in high inference latency and energy consumption, which is undesirable for real-time or resource-constrained applications such as mobile devices, embedded systems, or edge computing scenarios.

2.2.2 Early Exiting Rationale

With EE DNNs are modified in such a way that early classifiers are spread through the network architecture [48]. During the inference process, the input is sequentially evaluated by such intermediate classifiers and the execution is terminated if sufficient confidence is reached before the final exit point. An example of DNNs modified with 2 early classifiers is reported in Fig. 2.1.

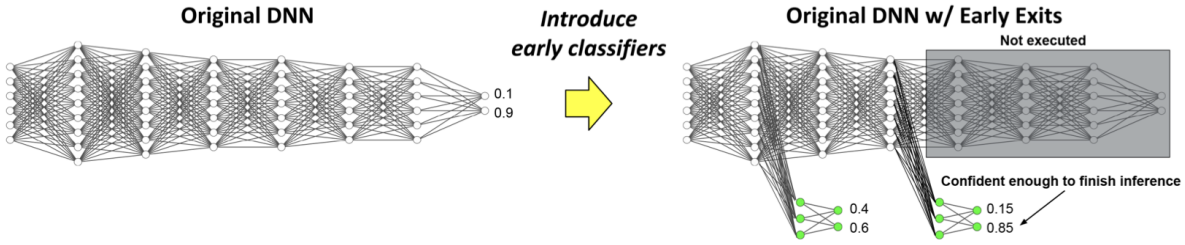


Figure 2.1: Example of DNN modified with EE [48].

EE originated for reducing DNN computational complexity without the need of designing lightweight models, i.e. simpler models with less parameters whose focus is to reach similar accuracy performance compared to large models, or relying on techniques such as model pruning, where parameters of the DNNs are removed, or knowledge distillation, where a smaller model is instructed by a larger model. Indeed, EE can save computational times when the inference process is stopped at one of the early classifiers, being able in turn to save computational resources.

The rationale behind EE is that the typical complexity of a DNN is not required for *all* the inputs provided to the network, hence wasting computational resources in such cases. Conversely, with EE the inference process is specifically tailored to the input provided to the network, adapting at runtime the required computational resources.

Below, we review three representative approaches that incorporate EE mechanisms into DNNs to reduce inference cost. The three works considered here illustrate the evolution of EE design principles: (i) **BranchyNet**, which augments standard architectures with auxiliary classifiers and relies on manually tuned entropy thresholds; (ii) **MSDNet**, which introduces multi-scale feature representations and dense connectivity to improve early classifier performance, yet still depends on handcrafted confidence criteria; and (iii) **EENet**, which learns both exit utility functions and confidence thresholds directly from data, eliminating manual tuning and achieving superior accuracy–latency trade-offs.

2.2.3 BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks [62]

BranchyNet modifies well-established architectures such as LeNet, AlexNet, and ResNet by adding several exit branches and additional convolutional layers, yielding Branchy-LeNet (B-LeNet), Branchy-AlexNet (B-AlexNet), and Branchy-ResNet (B-ResNet), respectively. An example of the B-AlexNet architecture is shown in Fig. 2.2.

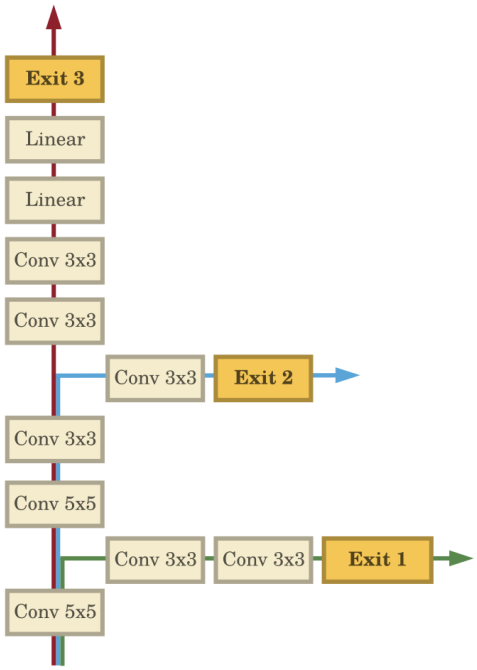


Figure 2.2: B-ALexNet Architecture [62].

Training is performed by jointly minimizing the weighted sum of the softmax cross-entropy loss across all exit points [62]:

$$L_{\text{branchynet}}(\hat{\mathbf{y}}, \mathbf{y}; \theta) = \sum_{n=1}^{N} w_n L(\hat{\mathbf{y}}_{exit_n}, \mathbf{y}; \theta), \tag{2.3}$$

where N is the total number of exit points and

$$L(\hat{\mathbf{y}}_{exit_n}, \mathbf{y}; \theta) = -\frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} y_c \log \hat{y}_{exit_{n,c}}, \tag{2.4}$$

with

$$\hat{\mathbf{y}}_{exit_n} = \text{softmax}(\mathbf{z}_{exit_n}) = \frac{\exp(\mathbf{z}_{exit_n})}{\sum_{c \in \mathcal{C}} \exp(z_{exit_{n,c}})}, \tag{2.5}$$

and $\mathbf{z}_{\text{exit}_n} = f_{\text{exit}_n}(\mathbf{x}; \theta)$. Here $\mathcal{C}$ is the set of classes, $\mathbf{y}$ is the one-hot ground truth vector, and $\mathbf{x}$ is an input sample.

After the training phase, the inference is performed according the procedure described in Fig. 2.3. In particular, at each exit, the entropy of the prediction is evaluated and it is compared with an exit point-dependent threshold value T_n. If the entropy is below such threshold, meaning that the network is confident with the prediction, the most probable label is returned, otherwise the network keeps going with the execution of the network.

1: **procedure** BRANCHYNETFASTINFERENCE($\boldsymbol{x}, \boldsymbol{T}$)
2: **for** $n = 1..N$ **do**
3: $\boldsymbol{z} = f_{\text{exit}_n}(\boldsymbol{x})$
4: $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{z})$
5: $e \leftarrow \text{entropy}(\hat{\boldsymbol{y}})$
6: **if** $e < T_n$ **then**
7: **return** $\arg\max \hat{\boldsymbol{y}}$
8: **return** $\arg\max \hat{\boldsymbol{y}}$

Figure 2.3: BranchyNet Early Exiting Procedure [62].

Clearly, the thresholds T_n play a key-role for the performance of the network, since they allow to balance execution time with accuracy. If the values of such thresholds are too small, the network might be forced to execute all the branches, thus without bringing benefits in terms of early exits. However, larger values of such thresholds could imply poor performance in terms of accuracy. In Fig. 2.4 the performance comparison between the original DNNs and the modified ones is reported. Performance are expressed in terms of average runtime per sample versus accuracy. Each point of the BranchyNets results corresponds to different values of the vector $\mathbf{T}$ containing the exit points thresholds T_n, obtained by sweeping through $\mathbf{T}$. Results show how the execution time is traded-off with the accuracy since a smaller execution time is always associated with a decrease of the classification accuracy.

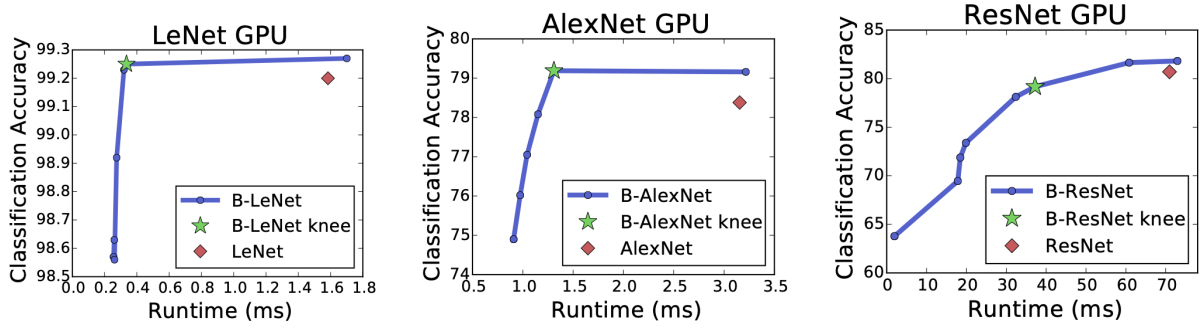


Figure 2.4: BranchyNet results are reported for all the three modified DNNs. Different values for the BranchyNets are due to different values of the thresholds T_n [62].

2.2.4 Multi-Scale Dense Networks for Resource Efficient Image Classification (MSDNET) [37]

MSDNet addresses two problems that authors of BranchyNet did not consider:

- **Lack of coarse-level features:** Early layers of a DNN typically capture fine-grained features, while coarse-level representations emerge only in deeper layers. Simply inserting early classifiers into shallow layers results in suboptimal accuracy because these classifiers rely solely on fine-level features. To overcome this limitation, MSDNet introduces *multi-scale* representations that provide early classifiers with access to both fine- and coarse-level features.

- **Interference from early classifiers:** The authors note that inserting early classifiers can degrade the performance of the final classifier, as early-layer features become optimized for short-term objectives rather than for deep feature extraction. Fig. 2.5 reports the relative accuracy of the final classifier—defined as the ratio between the accuracy of the modified network and the original model without early exits—for various architectures and early-exit placements. The results highlight that DenseNet is comparatively robust to the presence of early classifiers, motivating the use of *dense connectivity* in MSDNet. In DenseNet, each layer receives feature maps from all earlier layers. Even if early classifiers push some shallow features toward easier, short-term objectives, the deeper layers can still rely on many alternative feature maps that remain unaffected. As a result, dense connectivity prevents early exits from compromising the development of deep features and helps the final classifier maintain high accuracy.

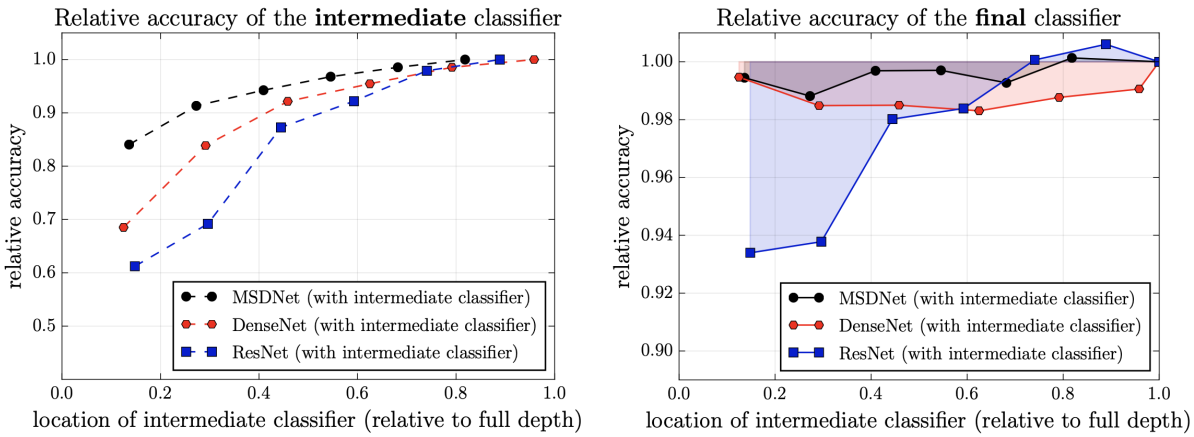


Figure 2.5: Comparison of relative accuracies of models with injected early exits [37].

The proposed architecture is reported in Fig. 2.6. In particular, MSDNet employs multi-scale feature representations across all layers, combining horizontal convolutions to preserve high-resolution information and diagonal strided convolutions to generate progressively coarser features. Dense connectivity ensures that each feature map at layer ℓ and scale s concatenates transformed features from the same scale and, when applicable, the adjacent finer scale. Classifiers are attached to selected layers and operate on the coarsest scale, leveraging dense connections to access all previously computed coarse features. As in BrachyNet, training is performed using a weighted

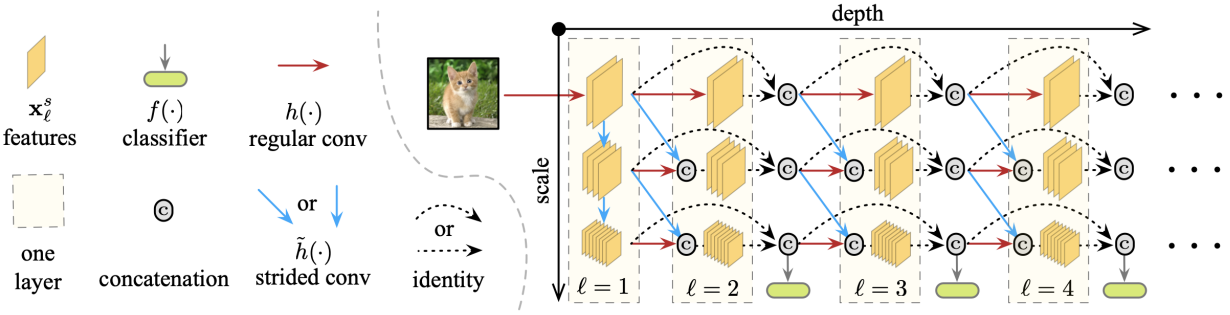


Figure 2.6: MSDNet Architecture [37].

cumulative cross-entropy loss that jointly optimizes all classifiers and enables the network to produce reliable predictions at multiple exit points.

Below we report the performance of the MSDNet architecture evaluated in the *anytime prediction* setting, where each test example is associated with a *computational budget*, which represents the maximum amount of computation (e.g., number of MUL-ADD) that can be spent before a prediction must be produced. The budget may vary across examples and is generally not known in advance. Performance is compared with other models for different values of computation budget and for different datasets. Across ImageNet and CIFAR-100 datasets, MSDNet consistently outperforms baselines including FractalNets, deeply supervised networks (ResNetMC, DenseNetMC), and ensembles of ResNets or DenseNets of varying or identical depths. Notably, MSDNet achieves 4–8% higher accuracy than ensembles of ResNets/DenseNets at intermediate computational budgets. Its advantage comes from early production of low-resolution, coarse features that are more informative for classification than the high-resolution early-layer features in standard networks.

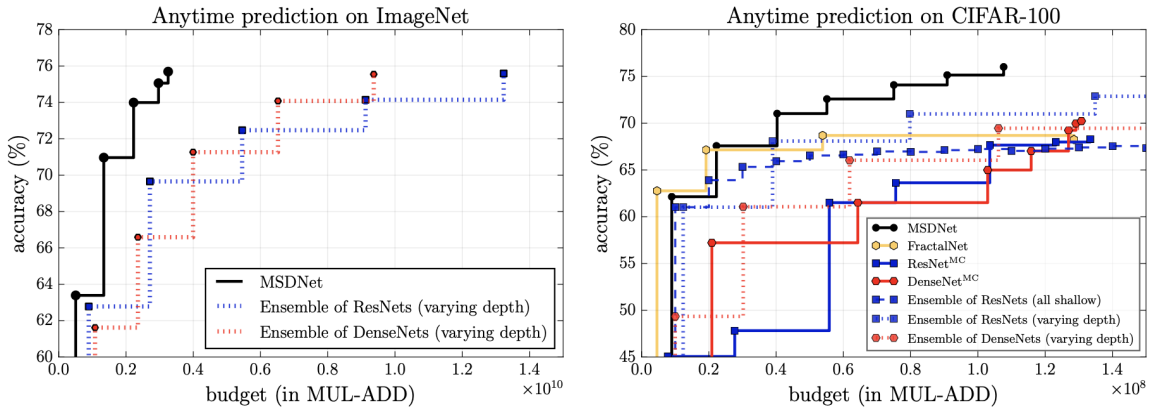


Figure 2.7: Anytime Prediction Results [37].

2.2.5 Adaptive Deep Neural Network Inference Optimization with EENet [38]

In the aforementioned works, the early classifiers are characterized by hand-tuning parameters that express the confidence measure of each classifier. EENet introduces a data-driven alternative: instead of hand-designed thresholds, it trains a network to automatically identify the best confidence thresholds for early exits. Moreover, the network learns the optimal confidence measure for each

Dataset, Model and Base Performance	Base Model Latency	Average Latency Budget per sample	Speed Gain	Metric (%)				
				BranchyNet	**MSDNet**	**PABEE**	**MAML-stop**	**EENet**
CIFAR-10		3.50 ms	1.34x	93.76	93.81	93.69	-	**93.84** ± 0.05
ResNet56 w/ 3 exits	4.70 ms	3.00 ms	1.56x	92.57	92.79	91.85	-	**92.90** ± 0.13
93.90% - *accuracy*		2.50 ms	1.88x	87.55	88.76	84.39	88.67	**88.90** ± 0.19
CIFAR-100		7.50 ms	1.36x	73.96	74.01	73.68	-	**74.08** ± 0.13
DenseNet121 w/ 4 exits	10.20 ms	6.75 ms	1.51x	71.65	71.99	68.10	-	**72.12** ± 0.20
75.08% - *accuracy*		6.00 ms	1.70x	68.13	68.70	61.13	69.00	**69.57** ± 0.22
ImageNet		125.00 ms	1.56x	74.10	74.13	74.05	-	**74.18** ± 0.13
MSDNet35 w/ 5 exits	195.14 ms	100.00 ms	1.95x	72.44	72.70	72.40	-	**72.75** ± 0.11
74.60% - *accuracy*		75.00 ms	2.60x	69.32	69.76	68.13	69.55	**69.88** ± 0.15
Cityscapes		100.00 ms	1.32x	79.22	78.75	72.20	-	**80.03** ± 0.21
HRNET-W48 w/ 3 exits	131.60 ms	50.00 ms	2.63x	75.54	75.50	70.09	-	**76.90** ± 0.17
80.90% - *mIoU*		15.00 ms	8.77x	68.12	68.35	63.95	65.76	**70.30** ± 0.35

Figure 2.8: EENet Performance Comparison [38].

exit, called the *exit utility scoring function*.

The method operates on a trained multi-exit model with K exits. For each exit k, EENet computes an exit score estimating the likelihood that the prediction is correct. This score is derived from the prediction probabilities and multiple confidence measures, including maximum probability, entropy, and agreement among exits. The network also estimates, for each sample, a probability distribution over exits, indicating the most suitable exit while satisfying a given computational budget.

EENet jointly optimizes the exit scoring functions and the exit assignment distribution on validation data. The scoring functions are trained with a weighted binary cross-entropy loss, ensuring that each exit specializes for the subset of samples likely to exit there. The assignment functions are optimized to match a target distribution that maximizes expected accuracy while respecting the average per-sample budget. Thresholds for early exiting are then automatically determined from the learned exit scores, ensuring an adaptive and data-driven early-exit policy.

EENet consistently outperforms previous early-exit approaches across a range of datasets and tasks, including image classification (CIFAR-10, CIFAR-100, ImageNet) and image segmentation (Cityscapes). Compared to representative methods such as BranchyNet, PABEE, MSDNet, and MAML-stop, EENet achieves higher accuracy at the same or lower average latency per sample. Overall, EENet demonstrates that data-driven estimation of exit utility scores and thresholds enables more efficient and accurate early-exit inference across multiple tasks and architectures.

2.2.6 Concluding Remarks on Early Exiting

In this section, we introduced the concept of EE in DNNs, highlighting its potential to reduce inference cost by allowing inputs to exit the network as soon as sufficient confidence is reached. We discussed representative approaches to EE design, from BranchyNet, which relies on manually tuned thresholds, to MSDNet, which improves early classifier accuracy through multi-scale feature representations and dense connectivity. Finally, we presented EENet, a data-driven approach that automatically learns both the confidence measures and the optimal thresholds for each exit. This removes the need for hand-tuning, allows each exit to specialize on the most suitable subset of inputs, and consistently achieves higher accuracy at lower latency across a variety of tasks.

As a final remark, EE architectures introduce additional optimization complexity due to multiple

intermediate classifiers, which may increase training time compared to standard networks. In DNN-based applications within EC systems, the training process depends on several factors, including privacy requirements and the frequency of model updates.

Privacy constraints may lead to local training, where mobile devices rely solely on local data to train their models. However, in such cases, model performance is often limited due to reduced data availability, and training times may be longer because of the limited computational resources of edge devices. Centralized edge or cloud training can achieve higher performance but may compromise data privacy. To address this trade-off, hybrid distributed mechanisms such as Federated Learning (FL) are usually adopted, where a global model is trained by aggregating only the weights of local models, thereby preserving data privacy [49].

The training frequency strongly depends on the application scenario. In many practical deployments, models are trained offline in the cloud and updated only periodically, while inference is executed continuously on edge devices. Paradigms such as FL, however, require more frequent training, due to the decentralized and continuously evolving nature of local data, increasing the relevance of training efficiency in edge environments. An exhaustive discussion on DNN training in EC systems is reported in [32].

In this work, we focus primarily on the inference time benefits, which justify the trade-off in deployment scenarios.

2.3 Distributional Reinforcement Learning

In the previous section, we introduced the concept of EE, outlining its principles, motivations, and the main approaches proposed to reduce the computational cost of deep models. In the following, we shift our focus to the second component of our framework, which is DistRL. To facilitate its understanding, we begin by presenting the fundamentals of classical RL, which provide the theoretical foundation upon which the distributional extension is built.

2.3.1 Reinforcement Learning Background

RL is a framework in which an *agent* learns to make decisions by interacting with an *environment*. The goal of the agent is to learn a *policy* that maximizes the *cumulative reward* obtained over time. Formally, the interaction proceeds as follows:

1. At time step t, the agent observes the current *state* of the environment $x_t \in \mathcal{X}$.
2. Based on this state, the agent chooses an *action* $a_t \in \mathcal{A}$.
3. The environment responds by transitioning to a new state x_{t+1} and providing a *reward* $r_t \in \mathcal{R}$ to the agent.

Here $\mathcal{X}$ is the set of all possible *states*, typically assumed to be finite, $\mathcal{A}$ is the set of all possible *actions*, also often finite, and $\mathcal{R}$ is the set of possible *rewards*, which may be discrete or continuous.

To distinguish between observed quantities and their stochastic nature, we denote the corresponding *random variables* with uppercase letters: X_t state at time t, A_t action at time t and R_t reward at time t.

The evolution of the system is driven by the actions undertaken by the agent and, in particular, according to the system *dynamics* $p(x'|x,a)$ which expresses the probability of going into state x' when in state x and taking action a. Note that here we are implicitly assuming that the system dynamics is *stationary* since these probabilities do not depend on the specific instant of time we visit the system states. Moreover, another important property that is assumed is the *Markov* property since such probabilities do not depend on previous system states and actions. In fact, a typical RL task is formulated as a (finite) Markov Decision Process (MDP), a topic that will be better formalized in the next chapter.

The agent's objective is usually to learn a policy $\pi(a|x)$, which defines the probability of taking action a given state x, in order to maximize the discounted cumulative reward [59], which can be defined, starting from time t, as

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k} = R_t + \gamma G_{t+1}, \tag{2.6}$$

where γ is the discount factor. The cumulative reward G_t is generally a random variable, as it depends on the sequence of visited states and selected actions. Because both state transitions and action choices are governed by probability distributions, classical RL methods aim at finding a policy π that maximizes the *expected* cumulative return $\mathbb{E}_\pi[G_t]$.

Value and Action-Value Functions

The expected cumulative return is represented by the *value* function. In particular, we can express the value obtained when following a policy π and starting from a state x according to the Bellman equation [60]

$$V^\pi(x) = \mathbb{E}_\pi\Big[G_t|X_t = x\Big] = \mathbb{E}_\pi\Big[R_t + \gamma V^\pi(X_{t+1})|X_t = x\Big]. \tag{2.7}$$

Another metric of interest is the *action-value* function for the policy π, defined as

$$Q^\pi(x,a) = \mathbb{E}_\pi\Big[G_t|X_t = x, A_t = a\Big], \tag{2.8}$$

which represents the value obtained when taking a specific action a in state x.

Typical objective of RL algorithms is to find an estimate of the value function, or the action-value function, and an optimal policy π^* which gives the optimal value function $V^*(x)$, where a policy π is considered to be better than π' if

$$V^\pi(x) \geq V^{\pi'}(x), \forall x \in \mathcal{X}. \tag{2.9}$$

The optimal value function $V^*(x)$ and the optimal action-value function $Q^*(x,a)$ are related through the Bellman optimality equation as

$$V^*(x) = \max_a Q^*(x,a), \tag{2.10}$$

and we can express the Bellman optimality equation for Q^* as

$$Q^*(x,a) = \mathbb{E}_\pi \left[R_t + \gamma \max_{a'} Q^*(X_{t+1}, a') | X_t = x, A_t = a \right]. \tag{2.11}$$

The Bellman Operator

For learning the value (or action-value) function it is possible to rely on the use of the Bellman operator T^π. Indeed, the operations performed on $V^\pi(x)$ described by the right hand-side of Eq. 2.7, namely the evaluation of $V^\pi(x)$ at X_{t+1}, its scaling by γ, the summation with R_t, and taking the expectation, are summarized by the *Bellman operator* T^π; then Eq. 2.7 can be rewritten as

$$(T^\pi V)(x) = \mathbb{E}_\pi \left[R_t + \gamma V^\pi(X_{t+1}) | X_t = x \right]. \tag{2.12}$$

The Bellman operator is useful because allows to write a more compact version of Eq. (2.12):

$$V = T^\pi V, \tag{2.13}$$

where $V \in \mathbb{R}^{\mathcal{X}}$ is the value function expressed for all the states of the system.

It can be shown that the Bellman operator T^π is a *contraction mapping*, meaning that repeatedly applying it to an initial estimate of the value function $V_0 \in \mathbb{R}^{\mathcal{X}}$, it will generate a sequence of estimates closer and closer to the desired value function V^π [13]. Furthermore, if we consider that a general solution of Eq. (2.13) is called *fixed point*, it is possible to show that V^π is the only fixed point of the Bellman operator.

While the Bellman operator T^π characterizes the value function associated with a fixed policy π, the ultimate goal in reinforcement learning is to compute the value function of the *optimal* policy. To formalize this, we introduce the Bellman optimality operator, which plays a role analogous to T^π but removes the dependence on a specific policy by selecting, in each state, the action that maximizes the expected return. The Bellman optimality operator T applied to a value function V is defined as

$$(TV)(x) = \max_{a \in \mathcal{A}} \mathbb{E} \left[R_t + \gamma V(X_{t+1}), \big| , X_t = x, A_t = a \right]. \tag{2.14}$$

The optimal value function V^* is then the unique fixed point of T, that is, $V^* = TV^*$.

Similarly, for the action-value function, we can define the optimality operator acting on Q:

$$(TQ)(x,a) = \mathbb{E}_\pi \left[R_t + \gamma \max_{a'} Q(X_{t+1}, a'), \big| , X_t = x, A_t = a \right]. \tag{2.15}$$

The Bellman optimality operator shares key properties with T^π: it is a contraction mapping and admits a unique fixed point. Repeated application of T, as done, for example, in Value Iteration, converges to the optimal value function, providing the theoretical foundation for optimal control in MDPs.

Deep RL Motivation

Among the classical RL algorithms, Dynamic Programming (DP) solves the RL problem exactly by alternating:

- *policy evaluation*, using T^π;
- *policy improvement*, using the Bellman optimality operator.

However, DP requires full knowledge of the transition probabilities. Monte Carlo and Temporal Difference (TD) methods, on the other hand, learn from sampled experience without requiring knowledge of the dynamics.

Furthermore, in classical RL, value functions and policies are typically represented in a *tabular* form, storing one entry per state (or state-action pair). While this is exact and straightforward for small problems, it quickly becomes infeasible in environments with large or continuous state and action spaces.

To overcome this limitation, RL algorithms can employ *function approximators* f_θ, parameterized by θ, to represent value functions or policies:

- **Value function approximation:** $V^\pi(x) \approx f_\theta(x)$ or $Q^\pi(x, a) \approx f_\theta(x, a)$.
- **Policy approximation:** $\pi_\phi(a|x) \approx g_\phi(x)$, where g_ϕ outputs a distribution over actions.

Function approximation enables *generalization* across states, allowing the agent to infer values or actions for states not explicitly visited during training. DNNs are a popular choice for f_θ and g_ϕ, giving rise to *Deep Reinforcement Learning* (DRL).

DRL algorithms are typically *model-free*, meaning that the agent does not have access to the underlying dynamics $p(x'|x, a)$ of the environment. Instead, the agent must learn optimal policies and value functions from experience, by interacting with the environment and collecting transitions (x, a, r, x') over time.

Learning purely online from consecutive transitions, however, poses several challenges:

- **Correlated data:** Consecutive transitions are highly correlated, violating the i.i.d. assumption that most neural network optimizers rely on.
- **Non-stationarity:** The distribution of states visited changes as the policy improves, making it difficult for the network to converge.
- **Sample inefficiency:** Each interaction with the environment is costly, and learning from each experience only once is inefficient.

To address these issues, Deep RL algorithms often employ a *replay buffer* $\mathcal{B}$ [51], which stores past transitions experienced by the agent:

$$\mathcal{B} = \{(x_t, a_t, r_t, x_{t+1})\}_{t=1}^{N}. \tag{2.16}$$

During training, mini-batches of transitions are sampled uniformly (or according to a priority) from $\mathcal{B}$. This has several advantages:

- It **breaks temporal correlations**, stabilizing the training of neural networks.
- It allows for **reusing past experiences**, improving sample efficiency.
- It supports **off-policy learning**, since the policy used to collect transitions may differ from the current policy being optimized.

Using a replay buffer, the expectation over transitions in the loss functions is approximated by an empirical average over a mini-batch sampled from $\mathcal{B}$, denoted by

$$\mathbb{E}_{(x,a,r,x')\sim\mathcal{B}}[\cdot].$$

Soft Actor-Critic

Soft Actor-Critic (SAC) is a model-free reinforcement learning algorithm designed for continuous action spaces [35]. It belongs to the family of *maximum entropy* reinforcement learning algorithms and extends classical RL by augmenting the reward objective with an entropy term, encouraging exploration.

Specifically, SAC replaces the classical objective $\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t R_t\right]$ with the *entropy-augmented* objective

$$J(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t\big(R_t + \alpha\mathcal{H}(\pi(\cdot|x_t))\big)\right], \tag{2.17}$$

where $\alpha > 0$ controls the relative importance of policy entropy. Increasing entropy encourages persistent exploration and prevents premature convergence to suboptimal policies.

To optimize the maximum entropy objective, SAC introduces the *soft* Q-function

$$Q^\pi_{\text{soft}}(x,a) = \mathbb{E}_\pi\left[R_t + \gamma V^\pi_{\text{soft}}(X')\right], \tag{2.18}$$

where the corresponding soft value function is defined as

$$V^\pi_{\text{soft}}(x) = \mathbb{E}_\pi[Q^\pi_{\text{soft}}(x,a) - \alpha\log\pi(a|x)]\,. \tag{2.19}$$

The operations performed in Eq. (2.18) can be represented by the soft Bellman operator T^π_{soft}. Under suitable assumptions, the soft Bellman operator remains a contraction, ensuring the existence and uniqueness of the fixed point Q^π.

Looking at the architecture of the algorithm, SAC uses two Q-networks Q_{θ_1} and Q_{θ_2}, to mitigate overestimation bias, a value network V_ψ and an actor network π_ϕ. Furthermore, to stabilize training, SAC employs *target networks*– exact copies of value and Q-networks– whose parameters $\bar{\psi}$ and $\bar{\theta}_i$ are updated according to:

$$\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i, i = \{1,2\} \tag{2.20}$$

$$\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}, \tag{2.21}$$

with $\tau \in (0,1)$. This soft update prevents oscillations and reduces the accumulation of approximation error.

The parameters of the Q function are trained to minimize the loss

$$\mathcal{L}_Q(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}}\Big[\frac{1}{2}\big(Q_{\theta_i}(s,a) - y\big)^2\Big], \quad i = 1,2 \tag{2.22}$$

where the target is given by the value network:

$$y = r + \gamma\mathbb{E}_{x'\sim p}[V_{\bar{\psi}}(x')]. \tag{2.23}$$

The value network $V_\psi(x)$ is trained to minimize the distance to the soft value estimated from the Q-networks:

$$\mathcal{L}_V(\psi) = \mathbb{E}_{x\sim\mathcal{B}}\Big[\frac{1}{2}\big(V_\psi(x) - \hat{V}(x)\big)^2\Big], \tag{2.24}$$

with

$$\hat{V}(x) = \mathbb{E}_{a\sim\pi(\cdot|x)}\Big[\min_{i=1,2} Q_{\theta_i}(x,a) - \alpha\log\pi(a|x)\Big]. \tag{2.25}$$

The actor is updated to minimize

$$\mathcal{L}^\pi(\phi) = \mathbb{E}_{x\sim\mathcal{B}}\left[\alpha\log\pi_\phi(a|x) - \min_i Q_{\theta_i}(x,a)\right]. \tag{2.26}$$

2.3.2 Distributional Variant Definition

When coming to DistRL, we are interested in learning the probability distribution of the cumulative discounted reward defined in Eq. (2.6), rather than its expected value represented by the value function. The reason behind this decision is because classical RL allows to learn policies that perform well *on average*. No information is provided instead on the *variability* of such performance.

Having control policies with only on average guarantees is not suitable for application scenarios where *risk* cannot be tolerated, which is the case for instance of CAVs applications. Conversely, by learning the full probability distribution of the random return, DistRL allows to learn so-called *risk-sensitive* policies which allows to balance performance and risk reduction. This trade off is particularly beneficial in safety-critical scenarios where risk reduction should be prioritized with respect to optimizing average performance.

The Distributional Bellman Equation

In order to formalize DistRL, we look back at the cumulative discounted reward as a function of the initial state x:

$$G^\pi(x) = \sum_{k=0}^{\infty}\gamma^k R_{t+k} = R_t + \gamma G_{t+1}. \tag{2.27}$$

For this random variable, referred to as the return-variable function, the following *random-variable* Bellman equation holds:

$$G^\pi(x) \overset{\mathcal{D}}{=} R + \gamma G^\pi(X'), \tag{2.28}$$

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

The previous equation is called *random-variable* because it only refers to the random variable $G^\pi(x)$, rather than its probability distribution. We are actually interested in what is called the *distributional* Bellman equation because it expresses a relationship directly on the probability distribution of the random return. If we denote with $\eta^\pi(x)$ the probability distribution of $G^\pi(x)$, then we can formulate the distributional Bellman equation as [13]

$$\eta^\pi(x) = \mathbb{E}_\pi[(b_{R,\gamma})_\#\eta^\pi(X')|X = x], \tag{2.29}$$

where $b_{R,\gamma}$ is the bootstrap function which performs the operations of scaling by γ and shifting by R, while $(b_{R,\gamma})_\#\eta^\pi(X')$ is the *pushforward* distribution of the transformation $b_{R,\gamma}$ applied to the

probability distribution $\eta^\pi(X')$, which can be explicitly expressed as

$$(b_{r,\gamma})_\# \eta^\pi(x') = \mathcal{D}(r + \gamma G^\pi(x')). \tag{2.30}$$

Eq. (2.29) tells that the probability distribution of the random return at state x is obtained by 3 operations on probability distributions, namely scaling, shifting and mixing. While the scaling and shifting operations might be obvious, the mixing operation requires explanation. The return distribution of the next (random) state X' is indeed a mixture of return distributions of all the next possible states [13]

$$\mathcal{D}_\pi(G^\pi(X')|X = x) = \sum_{x' \in \mathcal{X}} \mathbb{P}(X' = x'|X = x)\eta^\pi(x') = \mathbb{E}_\pi[\eta^\pi(X')|X = x], \tag{2.31}$$

where the weights of such combination are the state transition probabilities. In Fig. 2.9 a graphical representation of the distributional Bellman equation is reported.

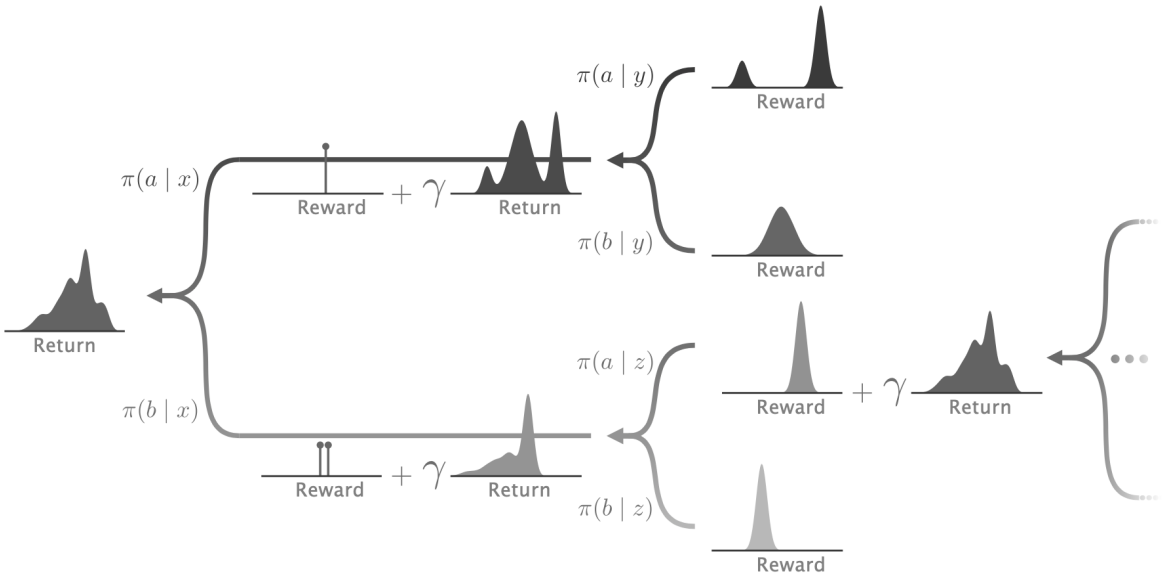


Figure 2.9: Graphical representation of Eq. (2.29) [13].

The Distributional Bellman Operator

Similarly to the previous section, we can summarize the operations performed on the return distribution by the *distributional* Bellman operator $\mathcal{T}^\pi$ as

$$(\mathcal{T}^\pi \eta)(x) = \mathbb{E}_\pi\big[(b_{R,\gamma})_\# \eta^\pi(X') \mid X = x\big]. \tag{2.32}$$

In analogy with the classical Bellman operator T^π, it is possible to show that $\mathcal{T}^\pi$ is a contraction mapping and that η^π is the unique fixed point of the distributional Bellman operator . However, in the distributional setting the contraction must be formulated with respect to *probability metrics*, that is, metrics defined directly on probability measures rather than on real numbers. In particular,

the distributional Bellman operator is a contraction mapping under several probability metrics, including the supremum p-Wasserstein distance and the Cramér distance [13].

Contractivity alone is not sufficient to guarantee that the sequence $\eta_{k+1} = \mathcal{T}^\pi \eta_k$ converges to η^π. Two additional conditions must be satisfied. First, the target distribution η^π must belong to the *finite domain* of the chosen probability metric d, denoted by $\mathscr{P}_d(\mathbb{R})$, defined as

$$\mathscr{P}_d(\mathbb{R}) = \big\{ \nu \in \mathscr{P}(\mathbb{R}) : d(\nu, \delta_0) < \infty \text{ and } \mathbb{E}_{Z\sim\nu}[|Z|] < \infty \big\}, \tag{2.33}$$

where $\mathscr{P}(\mathbb{R})$ is the space of probability distributions and δ_0 is the Dirac delta distribution that puts all of its mass in 0. Second, the space $\mathscr{P}_d(\mathbb{R})$ must be *closed* under the action of $\mathcal{T}^\pi$, meaning that for any $\eta \in \mathscr{P}_d(\mathbb{R})^{\mathcal{X}}$ we have $\mathcal{T}^\pi \eta \in \mathscr{P}_d(\mathbb{R})^{\mathcal{X}}$. When these two assumptions hold, the iterates η_k converge to η^π with respect to the supremum extension of the metric d [13], defined as

$$\bar{d}(\eta, \eta') = \sup_{x\in\mathcal{X}} d\big(\eta(x), \eta'(x)\big). \tag{2.34}$$

It can be shown that these conditions are satisfied whenever the reward distribution, namely the distribution from which instantaneous rewards are drawn, has a finite first moment.

The situation changes drastically when considering the *distributional Bellman optimality operator*. Unlike $\mathcal{T}^\pi$, the distributional optimality operator is not a contraction mapping under any probability metric, and therefore its iterates are not guaranteed to converge to an optimal return distribution. Convergence may still occur when the optimal policy is unique. However, if multiple optimal policies exist, they may induce different return distributions while sharing the same optimal expected value [13]. In such cases, the application of the distributional optimality operator may converge to a single distribution which, however, corresponds to a family of *non-stationary* optimal return distributions rather than a single stationary optimal solution.

As a final remark, we note that despite the absence of theoretical convergence guarantees in the optimality setting, DistRL has gained substantial attraction in practice. Its empirical performance often surpasses that of classical RL algorithms, and its ability to model and control risk makes it especially appealing in safety-critical applications. Nonetheless, when employing DistRL algorithms, it is crucial to carefully monitor the convergence of the learning process, in order to prevent undesired behaviors resulting from the lack of contraction properties in the distributional optimality operator.

Return Distribution Representations

We briefly described the fundamental equations of the DistRL, but one particular aspect that is missing in our tractation is how to *represent* a probability distribution. The desired characteristics of a proper representation are

1. it must be a *finite* representation, so that it can be stored and manipulated by a finite-memory computer;

2. it must be *closed* under the under the distributional Bellman operator, meaning that applying the operator to a distribution expressed in the chosen representation yields another distribution expressible within the same family.

A set of representations with such characteristics are the *empirical* representations. The set of

empirical representations can be defined as [13]

$$\mathscr{F}_E = \left\{ \sum_{i=1}^{m} p_i \delta_{\theta_i} : m \in \mathbb{N}^+, \theta_i \in \mathbb{R}, p_i \geq 0, \sum_{i=1}^{m} p_i = 1 \right\}, \tag{2.35}$$

where the pair (p_i, θ_i) is called *particle*.

We can then express the return distribution in the form of an empirical representation as

$$\eta(x) = \sum_{i=1}^{m(x)} p_i(x) \delta_{\theta_i(x)}. \tag{2.36}$$

As we may note, we should memorize for each state of the system the list of particles. This becomes untractable especially if we have to consider iterative procedures. We therefore look for more compact representations.

In particular, we look at the so-called *fixed-size* empirical representations, where we fix, per each state of the system, the number of particles. Fixed-size representations differ among themselves according to the aspects of the distribution we want to describe. If we are interested in modeling only the locations of the particles, we can rely on the *m-quantile* representation, defined as

$$\mathscr{F}_{Q,m} = \left\{ \frac{1}{m} \sum_{i=1}^{m} \delta_{\theta_i} : \theta_i \in \mathbb{R} \right\}. \tag{2.37}$$

If instead we fix the set of (equally distant) possible locations θ_i, with $\theta_1 < \cdots < \theta_m$, and we want to parametrize only the probability of the particles we obtain the *m-categorical* representation:

$$\mathscr{F}_{C,m} = \left\{ \sum_{i=1}^{m} p_i \delta_{\theta_i} : p_i \geq 0, \sum_{i=1}^{m} p_i = 1 \right\}. \tag{2.38}$$

Finally, if we want to parametrize both the locations and the probabilities we obtain the *m-particle* representation, defined as

$$\mathscr{F}_{E,m} = \left\{ \sum_{i=1}^{m} p_i \delta_{\theta_i} : \theta_i \in \mathbb{R}, p_i \geq 0, \sum_{i=1}^{m} p_i = 1 \right\}. \tag{2.39}$$

Fig. 2.10 shows the differences between the fixed-size empirical representations.

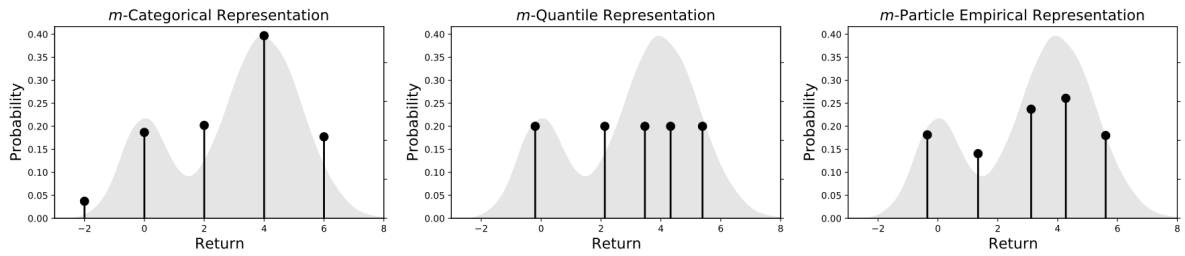


Figure 2.10: Fixed-size empirical representations [13].

Another approach to compactly representing probability distributions is to use a *parametric*

representation. In this case, the distribution is fully described by a small set of parameters. For example, a normal distribution is completely determined by its mean and variance; a uniform distribution is characterized by the endpoints of its support; and a Bernoulli distribution is fully specified by its success probability p.

Projection Steps

Fixed-size empirical and parametric representations are compact representations but they are not closed under the distributional Bellman operator. In fact, applying the operator generally produces a new distribution whose support and structure differ from those of the original representation. As a result, the updated distribution cannot be represented exactly within the fixed number of atoms or samples available.

This mismatch implies that the application of the distributional Bellman operator must be followed by a *projection step*, in which the resulting distribution is projected back onto the chosen representation class [13]. We can define the projector operator $\Pi_{\mathscr{F}}$ as

$$\Pi_{\mathscr{F}} : \mathscr{P}(\mathbb{R}) \to \mathscr{F}, \tag{2.40}$$

hence as a mapping from a probability distribution to a desired representation $\mathscr{F}$.

The projection step introduces irreversible information loss since tail behavior may be truncated, and the support may be compressed toward the fixed resolution of the representation. Despite these drawbacks, empirical representations remain widely used in Distributional RL due to their computational efficiency and conceptual simplicity. They allow closed-form updates for the projection step, straightforward implementation, and stable training behavior when used in conjunction with neural networks. Nevertheless, the projection bias they introduce must be considered when analyzing convergence properties, evaluating policy performance, or interpreting the learned distributions.

When using parametric representations, the need for a projection step after applying the distributional Bellman operator becomes even more evident. Recall that the distributional Bellman operator involves shifting, scaling, and mixing probability distributions. Even in the simple case of Gaussian returns, applying the operator generally produces a mixture of Gaussians, rather than a single Gaussian distribution. Consequently, the result of the Bellman update cannot be represented exactly within the chosen parametric family and must be projected back—for instance, a Gaussian mixture must be approximated by a single Gaussian using an appropriate projection rule.

Finally, we anticipate that we will employ a *Deep Distributional Reinforcement Learning* (Deep DistRL) approach, in which the return distribution is approximated by deep neural networks. As with empirical or parametric representations, the approximation introduced by a neural network is generally not closed under the distributional Bellman operator. Therefore, in the deep setting, each Bellman update effectively involves both a projection onto the chosen representation class and an approximation by the network. Despite this additional source of approximation error, Deep DistRL allows us to scale distributional RL to high-dimensional and continuous state spaces.

2.3.3 Concluding Remarks on Distributional Reinforcement Learning

In this section, we introduced the key ideas behind DistRL, highlighting the shift from modeling only the expected return to representing the full probability distribution of cumulative rewards.

This richer perspective allows agents to account for the variability of outcomes and to develop risk-sensitive policies, which is crucial in safety-critical or resource-constrained applications.

We discussed the distributional Bellman equation and operator, outlining their theoretical properties as well as practical considerations. While the distributional Bellman operator ensures convergence under suitable conditions, the distributional optimality operator does not guarantee contraction, requiring careful attention during learning. The choice of return distribution representation—empirical, fixed-size, or parametric—also influences computational efficiency and accuracy, and often involves projection steps to maintain tractable representations.

Finally, we anticipated the Deep Distributional RL setting, where neural networks approximate return distributions in high-dimensional or continuous state spaces. Despite introducing additional approximation errors and requiring projection after Bellman updates, DNNs make it possible to scale distributional RL to complex problems, combining expressive modeling with the benefits of risk-aware decision-making.

Chapter 3

Markovian Modeling and Optimization Frameworks

In this chapter, we develop three fundamental components that will be used in later chapters to model the dynamics of the source, the wireless channel, and the decision-making process at the control layer. Each of the three sections introduces a key building block for constructing a complete analytical framework based on Markov chains and Markov Decision Processes (MDPs).

The **first section** is dedicated to illustrate the framework needed to rigorously characterize a two-state Markov chain. Instead of directly specifying its transition probabilities, we aim to express them in terms of measurable quantities such as the long-run activity level and the duration of active periods. Achieving this requires a preliminary understanding of how Markov chains behave over extended time horizons, which motivates the theoretical background introduced in that section.

The **second section** introduces a finite-state Markov chain (FSMC) representation of the wireless channel, which preserves its temporal correlation while remaining tractable for MDP-based system analysis.

Finally, the **third section** formalizes MDPs and illustrates how optimal policies in MDPs can be derived with Linear Programming.

3.1 Markov Chain Fundamentals and Two-State Characterization

In order to characterize the dynamics of a two-state Markov chain in terms of observable and physically meaningful parameters, we first need to recall several fundamental concepts from the theory of Markov chains. Although our ultimate goal is to express the transition probabilities of a two-state model through quantities such as the steady-state distribution and the average sojourn time, this requires a clear understanding of how Markov chains behave over long time horizons.

For this reason, before deriving the desired parametrization, we introduce a set of structural and asymptotic properties that govern the evolution of Markov chains. These include the notions of communication between states, irreducibility, aperiodicity, recurrence, and the existence of limiting and stationary distributions. Together, these concepts allow us to describe the long-run behavior of the chain and to interpret stationary probabilities as time averages.

3.1.1 Markov Chain Definition and Properties

We start from the notion of a *Markov Process* $\{X_t\}$, which is a stochastic process where X_{t+1} only depends on X_t and not on X_s, with $s < t$. Then, we can define a *Markov Chain* as a Markov Process with a *numerable* state space $\mathcal{X}$. While Markov chains can be defined both in discrete and continuous time, in this text we focus on *discrete-time Markov chains*, where the process evolves in discrete steps $n = 0, 1, 2, \ldots$. In this case, the probability of being in a certain state depends only on the previous state [54]:

$$\Pr\{X_{n+1} = j | X_0 = i_0, \ldots, X_n = i_n\} = \Pr\{X_{n+1} = j | X_n = i_n\}. \tag{3.1}$$

Transition Probability Matrices

If we assume that the Markov process is *stationary*, we can further define the stationary *one-step* transition probabilities as

$$P_{ij}^{n,n+1} = \Pr\{X_{n+1} = j | X_n = i\} = P_{ij}, \tag{3.2}$$

where $P_{ij}^{n,n+1}$ is the probability of going into state j at time $n+1$, given that the process is in state i at time n. Stationarity implies that such probability does not depend on the specific time instant at which the states are visited; it depends only on the states themselves. Furthermore, the probabilities can be arranged in a matrix called *transition probability matrix* $\mathbf{P} = ||P_{ij}||$, with

$$P_{ij} \geq 0, \ \forall i, j, \qquad \text{and} \qquad \sum_{j=0}^{\infty} P_{ij} = 1, \ \forall i. \tag{3.3}$$

That is, each entry of the matrix is non-negative, and each row forms a probability distribution over the next-state space.

Since we are often interested in the probability of being in a given state after multiple time steps, a relevant element is the *n-step* transition probability matrix, whose generic entry $P_{ij}^{(n)}$ denotes the probability of transitioning from state i to state j in n steps. It can be shown that [54]

$$\mathbf{P}^{(n)} = \mathbf{P}^n, \tag{3.4}$$

so the n-step transition matrix is obtained simply by taking the matrix power of $\mathbf{P}$.

Relevant Properties

A natural starting point to identify properties of Markov chains is to formalize how states can be reached from one another, which leads to the concept of *communication between states*. In particular, a state j is said to be *accessible* from a state i if there exists a finite number of steps such that the chain can move from i to j with strictly positive probability. Formally [54],

$$i \to j \quad \text{if} \quad \exists\, n \geq 1 \text{ s.t. } P_{ij}^{(n)} > 0. \tag{3.5}$$

Two states i and j are said to *communicate* if each is accessible from the other:

$$i \leftrightarrow j \quad \text{if} \quad i \to j \text{ and } j \to i. \tag{3.6}$$

Communication partitions the state space into disjoint subsets called *equivalence classes*. All states within the same class communicate and share several structural properties, while states belonging to different classes do not interact.

Irreducible Markov Chains

The communication structure introduced previously naturally leads to the concept of *irreducibility*. A Markov chain with state space $\mathcal{X}$ is said to be *irreducible* if all states communicate with each other; that is, the entire state space forms a single equivalence class. Formally, the chain is irreducible if [54]

$$i \leftrightarrow j \qquad \forall\, i, j \in \mathcal{X}. \tag{3.7}$$

Equivalently, for any pair of states i and j, there exists a finite number of steps n such that the n-step transition probability from i to j is strictly positive:

$$\exists\, n \geq 1 \quad \text{s.t.} \quad P_{ij}^{(n)} > 0. \tag{3.8}$$

Intuitively, in an irreducible chain no region of the state space is isolated: starting from any state, the chain can eventually reach any other state with positive probability.

Aperiodicity

Another structural property that plays a crucial role in the long-term behavior of Markov chains is *aperiodicity*. The concept is related to the temporal pattern with which a state can be revisited.

For a given state i, define the set

$$\mathcal{N}(i) = \{n \geq 1 : P_{ii}^{(n)} > 0\}, \tag{3.9}$$

i.e., the set of time steps at which the Markov chain can return to state i with positive probability. The *period* of state i is then defined as the Greatest Common Divisor (GCD) of the elements of $\mathcal{N}(i)$ [54]:

$$d(i) = \gcd \mathcal{N}(i). \tag{3.10}$$

A state i is said to be *aperiodic* if its period is equal to one:

$$d(i) = 1. \tag{3.11}$$

Otherwise, the state is called *periodic* with period $d(i) > 1$.

Intuitively, a state with period $d(i) > 1$ can only be revisited at times that are multiples of $d(i)$, introducing a deterministic cyclic structure in the state evolution. On the contrary, if $d(i) = 1$, the chain can return to state i irregularly, without being constrained to a fixed cycle.

A key fact is that periodicity is a class property: if two states i and j communicate, then they share the same period. Consequently, in an irreducible Markov chain, either all states are aperiodic

or all states share a common period $d > 1$.

Recurrent States

Another important property of states in a Markov chain is *recurrence*, which describes whether the chain is guaranteed to return to a given state.

A state i is called *recurrent* if, starting from i, the chain will return to i with probability 1. Recurrence is a class property: if two states communicate, either both are recurrent or both are *transient*.

Formally, we can define the probability f_{ii} of returning into state i as a function of the probability of first returning in state i after n steps, denoted by $f_{ii}^{(n)} = \Pr\{R_i = n | X_0 = i\}$, where

$$R_i = \min\{n \geq 1; X_n = i\}. \tag{3.12}$$

is the first return time to state i. Then, we can express f_{ii} as [54]

$$f_{ii} = \sum_{n=0}^{\infty} f_{ii}^{(n)}. \tag{3.13}$$

Hence, we say that i is recurrent if $f_{ii} = 1$, otherwise is called a transient state.

3.1.2 Long-run behavior

We can now define the *limiting distribution* of a Markov chain, which describes its long-term behavior. Consider a discrete-time Markov chain $\{X_n\}_{n\geq 0}$ with *regular* transition probability matrix $\mathbf{P}$. In particular, a transition probability matrix $\mathbf{P}$ is said to be *regular* if some power k of the matrix $\mathbf{P}^k$ has all entries strictly positive. Intuitively, this means that, after a finite number of steps, it is possible to reach any state from any other state with positive probability.

Then, a probability distribution $\boldsymbol{\pi} = (\pi_i)_{i\in\mathcal{X}}$ called *limiting distribution* exists, with $\pi_i > 0 \quad \forall i$, where for any initial state $X_0 = i$, we have that [54]

$$\pi_j = \lim_{n\to\infty} \Pr\{X_n = j \mid X_0 = i\} = \lim_{n\to\infty} P_{ij}^{(n)}, \quad \forall j \in \mathcal{X}. \tag{3.14}$$

In other words, if the limiting distribution exists, the probability of finding the chain in state j becomes independent of the initial state after a large number of steps.

In particular, it can be shown that the limiting distribution $\boldsymbol{\pi}$ can be found by solving the following equations

$$\pi_j = \sum_{k=0}^{N} \pi_k P_{kj}, \forall j \in \mathcal{X}, \qquad \sum_{k=0}^{N} \pi_i = 1, . \tag{3.15}$$

A key aspect of modeling with Markov chains is the interpretation of the limiting distribution $\boldsymbol{\pi}$. Specifically, π_j represents the long-run fraction of time that the chain spends in state j. Formally,

if we start from state $X_0 = i$, the expected fraction of time spent in state j over m steps is

$$\mathbb{E}\left[\frac{1}{m}\sum_{k=0}^{m-1}\mathbf{1}_{\{X_k=j\}}\,\middle|\,X_0=i\right] = \frac{1}{m}\sum_{k=0}^{m-1}\mathbb{E}[\mathbf{1}_{\{X_k=j\}}\mid X_0=i] = \frac{1}{m}\sum_{k=0}^{m-1}\Pr(X_k=j\mid X_0=i) = \frac{1}{m}\sum_{k=0}^{m-1}P_{ij}^{(k)}, \tag{3.16}$$

where $\mathbf{1}_{\{X_k=j\}} = 1$ if $X_k = j$ and 0 otherwise.

The long-run mean fraction of time is instead defined as

$$\lim_{m\to\infty}\frac{1}{m}\sum_{k=0}^{m-1}P_{ij}^{(k)}. \tag{3.17}$$

As a final step, we show that Eq. (3.17) converges to the limiting distribution π_j. Recall a standard result from analysis: if a sequence of real numbers $a_0, a_1, \ldots$ converges to a limit a, then the averages of the sequence also converge to the same limit:

$$\lim_{m\to\infty}\frac{1}{m}\sum_{k=0}^{m-1}a_k = a.$$

Applying this to the sequence of transition probabilities $a_k = P_{ij}^{(k)}$, which converge to the stationary distribution π_j as $k \to \infty$, we obtain

$$\lim_{m\to\infty}\frac{1}{m}\sum_{k=0}^{m-1}P_{ij}^{(k)} = \pi_j.$$

Consequently, the long-run mean fraction of time spent in state j is

$$\lim_{m\to\infty}\mathbb{E}\left[\frac{1}{m}\sum_{k=0}^{m-1}\mathbf{1}_{\{X_k=j\}}\,\middle|\,X_0=i\right] = \pi_j,$$

which is independent of the starting state i. This justifies interpreting the stationary distribution π as the long-run fraction of time spent in each state.

An important implication of this interpretation is that if we consider the cost c_j of incurring into state j, we can evaluate the long-run mean cost as

$$\bar{c} = \sum_{j=0}^{N} c_j\,\pi_j. \tag{3.18}$$

Basic Limit Theorem of Markov Chains

Not all transition matrices of a Markov chain, however, are regular. Therefore, to guarantee the existence of a limiting distribution, we must rely on the structural properties of the chain itself. This leads to the formulation of the *Basic Limit Theorem*, which establishes conditions under which the limiting distribution exists [54]:

1. Consider a *recurrent, irreducible, and aperiodic* Markov chain. Let $P_{ii}^{(n)}$ denote the probability

of being in state i at the n-th transition, given that $X_0 = i$. Furthermore, let $f_{ii}^{(n)}$ be the probability of first returning to state i at the n-th transition, $n = 0, 1, 2, \ldots$, with the convention $f_{ii}^{(0)} = 0$. Then, the limiting probability exists and is given by

$$\lim_{n\to\infty} P_{ii}^{(n)} = \frac{1}{\sum_{n=1}^{\infty} n f_{ii}^{(n)}} = \frac{1}{m_i}, \tag{3.19}$$

where m_i is the mean time that passes when $X_0 = i$ and the next time the state is again in i. In particular, by recalling the random first return time R_i introduced previously, $m_i = \mathbb{E}[R_i | X_0 = i]$. Such result basically states that the chain comes back in state i every m_i steps, on average.

2. Under the same conditions as in (1), for any state j, the limiting probability of being in state i exists and is independent of the initial state j:

$$\lim_{n\to\infty} P_{ji}^{(n)} = \lim_{n\to\infty} P_{ii}^{(n)}, \quad \forall j \in \mathcal{X}, \tag{3.20}$$

Recalling the definition of limiting distribution $\lim_{n\to\infty} P_{ij}^{(n)} = \pi_j$, we are saying that $\lim_{n\to\infty} P_{ij}^{(n)} = \lim_{n\to\infty} P_{jj}^{(n)} = \pi_j = \frac{1}{m_j}$.

It is important to note that the basic limit theorem applies also to any aperiodic recurrent class. Indeed, in a recurrent class C $P_{ij}^{(n)} = 0$ if i belongs to the recurrent class and j not. Hence, the transition matrix $||P_{ij}||$ with $i, j \in C$ is a transition probability matrix implying that the associated Markov chain is irreducible and recurrent.

Stationary (Steady-State) Distribution

Let us consider an aperiodic recurrent class of states in a Markov chain. If for some state i in the class we have

$$\lim_{n\to\infty} P_{ii}^{(n)} > 0, \tag{3.21}$$

then it follows that

$$\lim_{n\to\infty} P_{jj}^{(n)} > 0, \quad \forall j \text{ in the class of } i. \tag{3.22}$$

In this case, the class is called *positive recurrent* or *strongly ergodic*. Intuitively, positive recurrence ensures that not only does the chain return to each state with probability 1 (recurrence), but also that the *expected return time* is finite. Indeed, by recalling that $\pi_i = \frac{1}{m_i}$ and m_i is the mean return time, we say that state i is positive recurrent if $m_i < \infty$, and *null recurrent* if $m_i = \infty$.

Alternatively, limiting distributions can be computed according to the following theorem. Consider a positive recurrent, aperiodic class of states $j = 0, 1, 2, \ldots$. Then, the following holds: In a positive recurrent aperiodic class, the limiting probabilities exist and are given by

$$\lim_{n\to\infty} P_{jj}^{(n)} = \pi_j = \sum_{i=0}^{\infty} \pi_i P_{ij}, \quad \sum_{i=0}^{\infty} \pi_i = 1, \tag{3.23}$$

where the set of probabilities $(\pi_i)_{i=0}^{\infty}$ is uniquely determined by

$$\pi_i \geq 0, \quad \sum_{i=0}^{\infty} \pi_i = 1, \quad \pi_j = \sum_{i=0}^{\infty} \pi_i P_{ij}, \quad \forall j \in \mathcal{X}. \tag{3.24}$$

Any set $(\pi_i)_{i=0}^{\infty}$ satisfying the above equations is called a *stationary probability distribution* of the Markov chain.

The term "stationary" arises from the property that if the chain is started according to the stationary distribution, it will remain distributed according to $\boldsymbol{\pi}$ at all future times.

As a final remark of this section, we say that a limiting distribution is also a stationary distribution. However, the converse is not true in general: a stationary distribution is any distribution that satisfies Eq. (3.24), whereas a limiting distribution may not exist. For instance, periodicity can limit or even prevent the existence of a well-defined long-term distribution. As an illustrative example, consider a two-state Markov chain with the following one-step transition probability matrix:

$$\mathbf{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{3.25}$$

In this case, the chain alternates deterministically between the two states at every time step. Therefore, it is easy to see that the chain does not converge to a limiting distribution since the probability of being in each state oscillates indefinitely over time. However, a stationary distribution $\pi = (\frac{1}{2}, \frac{1}{2})$ exists since it can be verified that satisfies Eq. (3.24).

3.1.3 Two-State Markov Chains

We can finally describe a two-state Markov chains by exploiting the concepts introduced so far. Consider a discrete-time Markov chain with two states, denoted x_0 (inactive) and x_1 (active), with one-step transition probability matrix

$$\mathbf{P} = \begin{pmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{pmatrix}, \tag{3.26}$$

where

$$\alpha = \Pr\{X_{t+1} = x_0 \mid X_t = x_0\} = P_{00}, \qquad \beta = \Pr\{X_{t+1} = x_1 \mid X_t = x_1\} = P_{11}.$$

Our goal is to express the transition probabilities α and β in terms of two quantities that are often observed in practice:

- $\pi_1 = \Pr\{X_t = x_1\}$: the steady-state probability of being active.
- b_1: the mean sojourn time in state x_1, also called *burstiness*.

The mean sojourn time b_1 is the expected number of consecutive time steps the chain remains in the active state x_1. Denote this random variable by T. Then

$$T = \min\{n \geq 1 : X_n = x_0 \mid X_0 = x_1\}.$$

Since the Markov chain has the *memoryless property*, the probability of leaving state x_1 in each step is constant: $1-\beta$. Therefore, the sojourn time T follows a geometric distribution:

$$\Pr\{T=k\} = \beta^{k-1}(1-\beta), \quad k = 1, 2, 3, \ldots \tag{3.27}$$

The mean of a geometric random variable is

$$b_1 = \mathbb{E}[T] = \frac{1}{1-\beta}, \tag{3.28}$$

therefore we can express the probability $1-\beta$ of transitioning from x_1 to x_0 as a function of the mean sojourn time as

$$1-\beta = \frac{1}{b_1}. \tag{3.29}$$

Recall that a stationary probability π_j satisfies $\pi_j = \sum_i \pi_i P_{ij}$. For a two-state Markov chain:

$$\pi_1 = \pi_0 P_{01} + \pi_1 P_{11} = (1-\pi_1)(1-\alpha) + \pi_1\beta. \tag{3.30}$$

where

- $\pi_1\beta$ is the steady-state probability of remaining in x_1,
- $(1-\pi_1)(1-\alpha)$ is the steady-state probability of transitioning from x_0 to x_1.

Solving the equation for $1-\alpha$ yields:

$$1-\alpha = \frac{\pi_1(1-\beta)}{1-\pi_1}. \tag{3.31}$$

Substituting $1-\beta = 1/b_1$ finally gives:

$$1-\alpha = \frac{\pi_1}{b_1(1-\pi_1)}, \tag{3.32}$$

$$1-\beta = \frac{1}{b_1}. \tag{3.33}$$

The probability β controls how long the chain tends to stay in the active state: higher $\beta \rightarrow$ longer bursts $\rightarrow$ higher b_1. The probability $1-\alpha$ is determined by both the fraction of time spent in the active state (π_1) and the burstiness b_1. Together, α and β fully characterize the long-term behavior of the two-state Markov chain in terms of observable quantities.

This simple, but very meaningful, model can be used in very different contexts. For instance, in [66], it is used for modeling a wireless channel where the mean sojourn time b allows to regulate the probability of failing packet transmissions, whereas it can also be used for modeling task and energy arrival processes [65],[67]. We anticipate that we are going to use this model in one of our contribution for modeling task arrival processes and remote computational resources availability.

3.2 Finite State Markov Chain Representation of the Wireless Channel

In the previous section we provided the main theoretical background in order to characterize a two-state Markov chain. Here, we provide a Markov chain-based model for representing the wireless channel process. The physical wireless fading process is inherently continuous and exhibits significant temporal correlation, as described by stochastic models such as Rayleigh or Rician fading. However, in many system-level analyses the channel is often approximated as a sequence of independent realizations, neglecting this correlation and thereby providing an unrealistic representation of the channel dynamics.

To capture the temporal dependence of the fading while retaining analytical tractability, it is common to approximate the continuous fading process through a FSMC. This model discretizes the fading amplitude into a finite number of levels and assigns transition probabilities that reflect the underlying correlation structure of the physical channel. This approach preserves the essential statistical properties of the fading process while yielding a model that is analytically convenient and computationally efficient, making it suitable for real-time decision-making in wireless systems.

In this section, we summarize the construction of such a model for a Rayleigh fading channel, following the first-order Markov modeling approach discussed in "On First-Order Markov Modeling for the Rayleigh Fading Channel" [57].

3.2.1 Multipath Fading and Rayleigh Channel Modeling

In wireless communication systems, the received signal is typically the superposition of multiple replicas of the transmitted waveform, each traversing a different propagation path with distinct delay, attenuation, and phase rotation. This phenomenon, known as *multipath propagation*, leads to rapid fluctuations of the received signal amplitude and phase, commonly referred to as *fading*. When no dominant Line-of-Sight (LOS) path is present and the individual multipath components experience independent scattering, the in-phase and quadrature components of the received baseband signal can be accurately modeled as jointly Gaussian random processes. Under these conditions, the envelope of the received signal follows a *Rayleigh* distribution, giving rise to the well-known *Rayleigh fading channel* model.

Under the assumption of small-scale fading, the complex wireless channel gain can be expressed as [52]

$$h(t) = h_I(t) + j\, h_Q(t), \tag{3.34}$$

where $h_I(t)$ and $h_Q(t)$ are zero-mean, jointly Gaussian random processes representing the in-phase and quadrature components, respectively. Assuming uniform scattering of plane waves arriving from all azimuthal directions, and a maximum Doppler frequency f_D determined by the relative velocity between transmitter and receiver, the autocorrelation function of the complex fading process takes the form [52]

$$R_h(\tau) = \mathbb{E}\left[h(t)h^*(t+\tau)\right] = J_0(2\pi f_D \tau), \tag{3.35}$$

where $J_0(\cdot)$ is the zeroth-order Bessel function of the first kind. This closed-form expression captures the temporal correlation induced by Doppler spread and is widely validated for isotropic scattering environments.

3.2.2 First-order Markov model

A first-order Markov chain is fully characterized by its initial state distribution and its one-step transition probabilities . Let $\{X_n\}_{n\geq 0}$ denote the discrete-time Markov chain obtained by sampling the wireless fading process at discrete instants n. Let also R denote the received signal amplitude, modeled as a Rayleigh random variable with parameter σ. Its Probability Density Function (PDF) and Cumulative Distribution Function (CDF) are given by [52]

$$f_R(r) = \frac{r}{\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right), \qquad r \geq 0, \tag{3.36}$$

$$F_R(r) = 1 - \exp\left(-\frac{r^2}{2\sigma^2}\right). \qquad r \geq 0. \tag{3.37}$$

To construct a FSMC, the amplitude range $[0, \infty)$ is partitioned into N non-overlapping intervals. The associated threshold values are denoted by

$$0 = \Gamma_0 < \Gamma_1 < \cdots < \Gamma_N.$$

In an equal-probability partitioning, each state is required to have the same initial-state occupancy probability. Therefore, the thresholds satisfy

$$F_R(\Gamma_i) = \frac{i}{N}, \qquad i = 0, 1, \ldots, N. \tag{3.38}$$

Substituting (3.37) into (3.38) yields

$$1 - \exp\left(-\frac{\Gamma_i^2}{2\sigma^2}\right) = \frac{i}{N},$$

which implies

$$\exp\left(-\frac{\Gamma_i^2}{2\sigma^2}\right) = 1 - \frac{i}{N}.$$

Taking logarithms on both sides gives the explicit expression for the thresholds:

$$\Gamma_i = \sigma\sqrt{-2\ln\left(1 - \frac{i}{N}\right)}, \qquad i = 0, 1, \ldots, N. \tag{3.39}$$

Ideally, $\Gamma_N = \infty$, but in a practical FSMC model the final threshold is set to a large but finite value so that the unrepresented tail probability

$$1 - F_R(\Gamma_N) = \exp\left(-\frac{\Gamma_N^2}{2\sigma^2}\right)$$

is negligible. Because $\Gamma_N < \infty$, the sum of the initial-state probabilities over the N states becomes

$$\sum_{i=0}^{N-1} \pi_i = F_R(\Gamma_N) < 1.$$

To ensure that the initial-state probabilities sum to one, they are uniformly scaled as

$$\tilde{\pi}_i = \frac{\pi_i}{F_R(\Gamma_N)} = \frac{F_R(\Gamma_{i+1}) - F_R(\Gamma_i)}{F_R(\Gamma_N)}, \qquad i = 0, \ldots, N-1. \tag{3.40}$$

The values $\tilde{\pi}_i$ represent the properly normalized initial-state occupancy probabilities used in the FSMC representation of the Rayleigh fading channel.

Transition Probabilities of the FSMC

The transition probabilities of the FSMC are derived from the *bivariate Rayleigh distribution*, which models the joint statistics of two consecutive samples of the fading amplitude.

Given the partitioning thresholds Γ_i, the transition probability from state i to state j is computed as

$$P_{ij} = \Pr\{R_{n+1} \in [\Gamma_j, \Gamma_{j+1}) \mid R_n \in [\Gamma_i, \Gamma_{i+1})\} = \frac{\int_{\Gamma_i}^{\Gamma_{i+1}} \int_{\Gamma_j}^{\Gamma_{j+1}} f_{R_n, R_{n+1}}(r_n, r_{n+1})\, dr_{n+1}\, dr_n}{\int_{\Gamma_i}^{\Gamma_{i+1}} f_R(r_n)\, dr_n}, \tag{3.41}$$

where $f_{R_n,R_{n+1}}(r_n, r_{n+1})$ is the joint pdf for a Bivariate Rayleigh distribution. The double integral in (3.41) can be efficiently approximated using series expansions, which allows computation of all transitions between states without restricting transitions to adjacent states only [57].

Finally, due to the finite range of amplitudes represented in the FSMC, a scaling of the transition matrix may be necessary to enforce row stochasticity:

$$\tilde{P}_{ij} = \frac{P_{ij}}{\sum_{k=0}^{N-1} P_{ik}}, \qquad i, j = 0, \ldots, N-1. \tag{3.42}$$

This guarantees that each row sums to one, resulting in a valid transition probability matrix.

3.2.3 Validity and Limitations of the FSMC Approximation

To assess how well the Markov approximation captures the temporal dynamics of the fading amplitude, the envelope correlation function is compared with the autocorrelation function generated by the Markov chain constructed earlier. For the Markov chain, the discretized envelope autocorrelation at lag m is computed as

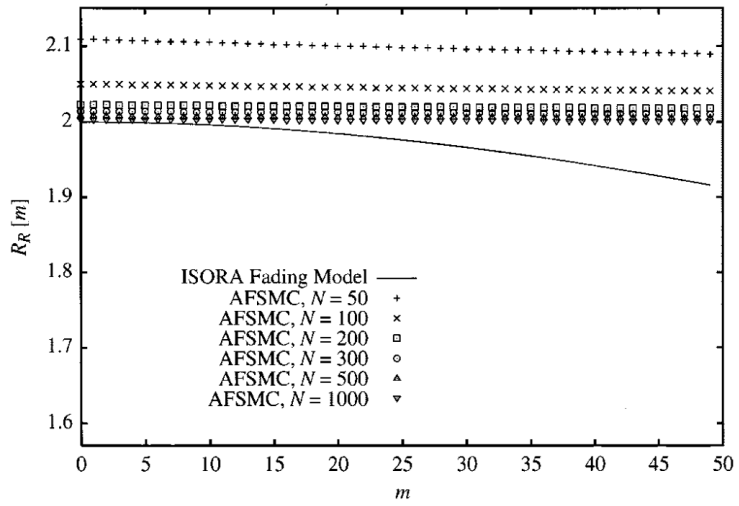
$$R_R[m] = \sum_{i=1}^{N} \sum_{j=1}^{N} r_i r_j \Pr\{R_m = i, R_0 = j\} = \sum_{j=1}^{N} r_j \pi_j \sum_{i=1}^{N} r_i p_{i,j}^{(m)} \tag{3.43}$$

While the complex fading process $h(t)$ is Gaussian, the signal envelope $r(t) = |h(t)|$ follows a Rayleigh distribution. Its temporal correlation structure is more involved and differs from the correlation of the underlying complex process. Specifically, the autocorrelation function of the envelope is given by [57]

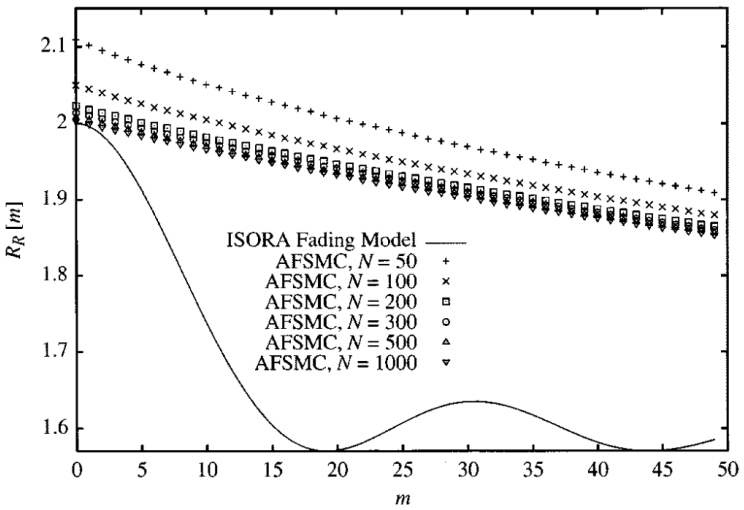
$$R_r(\tau) = \mathbb{E}[r(t)r(t+\tau)] = \frac{\pi b_0}{2}\, {}_2F_1\left(-\tfrac{1}{2}, -\tfrac{1}{2}; 1; R_h(\tau)^2\right), \tag{3.44}$$

where ${}_2F_1(a, b; c; z)$ is the hypergeometric function.

In Fig. 3.1, the envelope correlation function obtained from the theoretical Isotropic Scattering, Omnidirectional Receiving Antenna (ISORA) model is compared with the correlation function of the chosen Markov-based approximation. The comparison is shown for two values of the parameter $f_D T$ and for different numbers of channel states N. The results highlight that the autocorrelation function of the Markov-generated process differs from the theoretical one: while the true envelope correlation exhibits a decaying oscillatory behavior, the Markov model produces a correlation that decays approximately exponentially. Despite this mismatch, the analyzed model remains effective in capturing temporal correlation in the fading process, providing a substantially more realistic representation than i.i.d. Rayleigh samples.



(a) Autocorrelation comparison for $f_D T = 0.002$ and different values of N.



(b) Autocorrelation comparison for $f_D T = 0.02$ and different values of N.

Figure 3.1: Comparison between the theoretical ISORA envelope autocorrelation function and the autocorrelation obtained from the proposed Markov model, for two values of the normalized Doppler parameter $f_D T$ and multiple state-space resolutions N [61].

As a final remark, a more accurate FSMC representation of the wireless channel can be obtained by enlarging the state space to include not only the signal amplitude but also its *rate of change* [20]. Such a model is capable of better reproducing the theoretical decaying oscillating autocorrelation

function of the Rayleigh fading envelope with significantly higher accuracy compared to first-order FSMCs.

3.3 Markov Decision Processes and Optimal Policy

In the previous chapter, we introduced MDPs within the context of RL. MDPs provide the fundamental theoretical foundation for RL, but RL algorithms are not the only tool for computing optimal control strategies. In this section, we better formalize MDPs and show that optimal policies for average-reward MDPs can also be obtained by solving a suitable Linear Programming (LP) problem.

3.3.1 Markov Decision Processes Definition

MDPs can be intended as a particular case of *sequential decision making* model. The main elements of such a model are the (set of) states of the system we want to control, the (set of) possible actions we can take in the specific states, a decision maker, a policy which describes how to select the actions according to the state, a reward function that expresses the obtained reward for the specific state-action pair and the dynamics of the system which describes how the system evolves according to the undertaken actions. In particular, the decision maker, or agent, observes the state, takes an action, receives a reward and then the decision process is repeated in the next state.

An MDP is formally defined by a tuple $(\mathcal{X}, \mathcal{A}, P, r)$ where:

- **State space** $\mathcal{X}$: the set of all possible configurations in which the system can be observed. Each state $i \in \mathcal{X}$ summarizes all relevant information needed for future decision-making.
- **Action space** $\mathcal{A}$: the set of control actions available to the decision maker. In general, not all actions need to be available in every state; this can be handled by defining feasible action sets $\mathcal{A}(i) \subseteq \mathcal{A}$.
- **Transition probabilities** $P_{ij}(a)$: the probability of transitioning from state i to state j when action a is taken. Formally,

  $$P_{ij}(a) = \Pr(X_{t+1} = j \mid X_t = i, A_t = a),$$

  where $\sum_{j \in \mathcal{X}} P_{ij}(a) = 1$ for all i and a.
- **Reward function** $r(i, a)$: the immediate reward obtained when action a is taken in state i. If the reward also depends on the next state j, we can compute the average reward $r(i, a)$ as

  $$r(i, a) = \sum_{j \in \mathcal{X}} P_{ij}(a)\, r(i, a, j). \tag{3.45}$$

In MDPs we are interested in learning a *policy* μ that prescribes how to select actions based on the currently observed state, with the goal of optimizing a given performance criterion, typically the expected cumulative or long-term average reward.

A policy can be either *deterministic* or *stochastic*. A deterministic policy selects a single action in each state, i.e.,

$$\mu(i) \in \mathcal{A}, \qquad \forall i \in \mathcal{X},$$

meaning that the agent always chooses the same action whenever the system is in state i. Instead, a stochastic (or randomized) policy assigns a probability distribution over actions:

$$\mu(a \mid i) = \Pr(A_t = a \mid X_t = i),$$

allowing the agent to randomize its behavior in order to explore the environment or satisfy additional performance constraints.

A particularly important class of policies is that of *stationary* policies. A policy μ is said to be stationary if it does not change over time, i.e., the rule used to select actions depends only on the current state and not on the time index t. Formally, a stationary randomized policy is defined through fixed action probabilities $\mu(a \mid i)$ for all (i, a) and all decision epochs. Stationary policies are appealing because of their simplicity and their ability to capture long-run behavior in controlled Markovian systems.

When a stationary policy μ is fixed, the state process $\{X_t\}$ evolves as a Markov chain with transition matrix

$$P_\mu(i, j) = \sum_{a \in \mathcal{A}} \mu(a \mid i)\, p(j \mid i, a),$$

and the sequence of pairs (X_t, A_t) is generated according to the policy and the system dynamics. This induced Markov chain plays a central role in analyzing long-run performance metrics such as average reward and occupation measures.

In many applications of MDPs the goal is to optimize long-term performance metrics such as the average reward, or the average discounted reward in classic RL, possibly subject to long-run average constraints on costs [56]. In particular, the performance of a policy is typically evaluated through *temporal averages*:

$$\bar{R}(\mu) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} r(X_t, A_t), \qquad \bar{C}_k(\mu) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} c_k(X_t, A_t), \tag{3.46}$$

with $\bar{R}(\mu)$ being the average reward obtained following policy μ and $\bar{C}_k(\mu)$ is the average cost associated to the cost function c_k, with $k = 1, \ldots, K$. In the following we will show how to find an *optimal* policy.

3.3.2 Linear Program Formulation

Assume that the Markov chain induced by μ is *ergodic*. Under this assumption, the chain admits a unique stationary distribution, and we can define the joint stationary distribution $\pi(i, a)$ of being in state i and selecting action a. With this notation, temporal averages can be rewritten as *stochastic averages*:

$$\bar{R}(\mu) = \sum_{i \in \mathcal{X}} \sum_{a \in \mathcal{A}} r(i, a)\, \pi(i, a), \qquad \bar{C}_k(\mu) = \sum_{i \in \mathcal{X}} \sum_{a \in \mathcal{A}} c_k(i, a)\, \pi(i, a). \tag{3.47}$$

Thus, under ergodicity, long-run performance depends only on the limiting distribution of state–action, rather than on the particular trajectory of the process.

The same conclusions hold under the weaker assumption that the Markov chain induced by π is *unichain* [56], meaning that it has a single recurrent class (possibly with transient states). In this case, the limiting state–action probabilities still exist and are independent of the initial state. We adopt the ergodicity assumption only for simplicity of exposition.

Once temporal averages are expressed as stochastic averages, the MDP can be reformulated as a linear program whose decision variables are the occupation probabilities $\pi(i,a)$. These variables, together with probability distribution constraints, must satisfy a *flow conservation* constraint

$$\sum_{a\in\mathcal{A}} \pi(j,a) - \sum_{i\in\mathcal{X}}\sum_{a\in\mathcal{A}} \pi(i,a)P_{ij}(a) = 0, \qquad j \in \mathcal{X}, \tag{3.48}$$

where the first term, $\sum_{a\in\mathcal{A}} \pi(j,a)$, represents the total long-run probability of being in state j irrespective of the chosen action, while the second term, $\sum_{i\in\mathcal{X}}\sum_{a\in\mathcal{A}} \pi(i,a)P_{ij}(a)$, represents the probability of arriving in state j from all other states according to the transition dynamics. In the stationary regime, these two probabilities must be equal, ensuring consistency of the occupation probabilities across states.

The resulting Linear Program (LP) is [56]

$$\begin{aligned}
\max \quad & \sum_{i\in\mathcal{X}}\sum_{a\in\mathcal{A}} r(i,a)\,\pi(i,a) && (3.49)\\
\text{subject to} \quad & \sum_{a\in\mathcal{A}} \pi(j,a) - \sum_{i\in\mathcal{X}}\sum_{a\in\mathcal{A}} \pi(i,a)P_{ij}(a) = 0, \quad \forall j \in \mathcal{X} && (3.50)\\
& \sum_{i\in\mathcal{X}}\sum_{a\in\mathcal{A}} c_k(i,a)\,\pi(i,a) \le C_k, \quad k = 1,\dots,K && (3.51)\\
& \sum_{i\in\mathcal{X}}\sum_{a\in\mathcal{A}} \pi(i,a) = 1, && (3.52)\\
& \pi(i,a) \ge 0, \quad \forall i \in \mathcal{X},\ a \in \mathcal{A}. && (3.53)
\end{aligned}$$

An optimal solution $\pi^\star(i,a)$ to this LP yields an optimal stationary policy via the normalization

$$\mu^\star(a|i) = \frac{\pi^\star(i,a)}{\sum_{a'\in\mathcal{A}} \pi^\star(i,a')} \qquad \text{whenever } \sum_{a'} \pi^\star(i,a') > 0.$$

Thus, the linear programming approach provides a direct method for computing optimal policies in MDPs based solely on steady-state occupancy measures.

3.4 Conclusions

In this chapter, we established the theoretical foundations needed to model and optimize the dynamics of edge computing systems through Markovian frameworks. We began by reviewing the key structural and asymptotic properties of Markov chains, which enabled us to express the behavior of a two-state process in terms of measurable long-run quantities such as steady-state probabilities and average sojourn times. This parametrization is essential for describing traffic or activity patterns in a way that is both analytically tractable and physically interpretable.

We then introduced a finite-state Markov representation of the wireless fading channel. By discretizing the Rayleigh envelope and deriving transition probabilities from the underlying joint distribution, the resulting FSMC captures temporal correlation while remaining compatible with MDP-based optimization. Despite its approximate nature, this modeling approach provides a significantly more realistic description than i.i.d. fading and offers a versatile basis for system-level analysis.

Finally, we formalized the MDP framework and showed how optimal long-run control policies can be computed using Linear Programming through state–action occupation probabilities. This offers an alternative to classical RL algorithms and establishes a direct link between probabilistic system dynamics and optimal decision strategies. Overall, the concepts developed in this chapter will be used in the following chapters to design, analyze, and optimize edge computing systems operating under stochastic dynamics.

Part II

Contributions

Chapter 4

Edge Computing with Early Exiting for Adaptive Inference in Mobile Autonomous Systems [10]

Starting from this chapter we present our contributions related to the analysis of an edge computing architecture enhanced with early exiting. In particular, the material presented here belongs to the paper "Edge Computing with Early Exiting for Adaptive Inference in Mobile Autonomous Systems" presented at the IEEE International Conference on Communications (ICC) 2024, Denver, CO.

4.1 Introduction

As already anticipated in the first chapter, we focus on a use-case scenario where Connected and Automated Vehicles (CAVs) must perform inference tasks with the support of an edge computing architecture. In this context, several works have proposed offloading mechanisms tailored to the constraints of CAV applications. For instance, [68] compares strategies for independent and interconnected edge servers, showing that cooperation among servers can effectively mitigate the impact of user mobility. Other studies address latency minimization through analytical optimization: [69] formulates the problem in game-theoretic terms, while [47] relies on deep reinforcement learning for the same purpose. Further contributions, such as [18, 19, 43], adopt an MDP-based framework to derive optimal offloading policies.

We recall that EE offers a mechanism to make DNN inference adaptive by allowing execution to stop once intermediate predictions reach sufficient confidence. This enables a flexible trade-off between accuracy and latency and allows the computational load to scale with both input complexity and system conditions. When combined with edge-assisted inference, such adaptability becomes particularly advantageous, as it can substantially reduce the execution time of offloaded tasks while improving resource utilization.

Motivated by these considerations, in this chapter we investigate the joint use of EE and edge computing for DNN task offloading in resource-constrained mobile systems. To the best of our knowledge, this was the first work that integrated these two paradigms and derived an optimal, low-complexity policy capable of adapting to dynamic application requirements and varying system states. More specifically, our main contributions are summarized as follows:

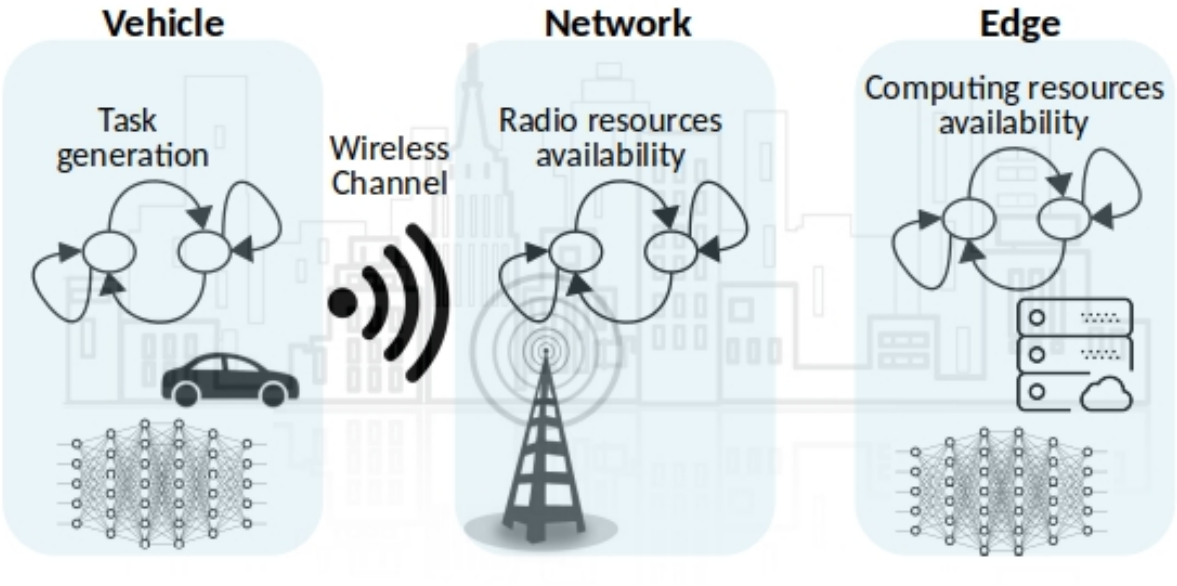


Figure 4.1: Reference system scenario and main components of the system model.

1. We develop a MDP framework to determine the optimal inference execution strategy in an edge computing scenario. Differently from [18, 19, 43], we consider that the CAV executes a simple DNN without EE, whereas the edge server can exploit EE, leading to a flexible and effective offloading strategy.

2. We formulate an optimization problem to identify the inference execution strategy that maximizes accuracy while satisfying application-level requirements in terms of inference time and task dropping rate.

3. We evaluate the obtained optimal policy under varying system settings and operational conditions, including propagation channel characteristics, network congestion, and edge server load. The performance of our policy is compared with that of an optimal policy without EE, showing that the integration of EE can achieve up to an 11% improvement in performance.

4.2 System Model

We consider a simplified system scenario, illustrated in Fig. 4.1, consisting of a CAV that performs DNN-based inference tasks and an edge server connected to a base station. The CAV can either execute the tasks locally or offload them to the edge server. We assume that the edge server employs a dynamic neural model with EE, whereas the CAV executes simpler DNN models without branches due to its limited computational resources. Time is assumed to be slotted, and we denote by t the time interval $[tT, (t+1)T)$, where T is the slot duration. For compatibility with the subframe duration in 4G and 5G radio interfaces, we set $T = 1\,\text{ms}$.

The remaining aspects of the system, including task generation and handling, data transfer over the radio interface, and network modeling, are described below.

4.2.1 Task generation

Tasks generated by the CAV are stored in a queue of finite capacity $Q_{\max}$. Let $q(t)$ denote the queue state, i.e., the number of backlogged tasks at time slot t. The queue evolves according to

$$q(t+1) = \min\{\max\{q(t) - e(t), 0\} + i(t), Q_{\max}\}, \tag{4.1}$$

where $e(t)$ is a binary indicator of task execution at slot t, and $i(t)$ is the number of incoming tasks. When the queue is full, newly generated tasks are discarded.

Task generation depends on the application state $v(t) \in \{v_0, v_1\}$, where v_1 denotes the active state (tasks are generated) and v_0 the idle state (no task generation). For simplicity, we assume at most one task can be generated per slot. Accordingly, the task generation probabilities are defined as $\Pr(i(t) = 0|v(t) = v_0) = 1$ and $\Pr(i(t) = 1|v(t) = v_1) = 1$.

The application state evolves according to a two-state Markov chain with transition probability matrix

$$\mathbf{P_V} = \begin{pmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{pmatrix},$$

where $\alpha = \Pr(v(t+1) = v_0 \mid v(t) = v_0)$ and $\beta = \Pr(v(t+1) = v_1 \mid v(t) = v_1)$. The transition probabilities are expressed in terms of the burstiness b_v (mean sojourn time in state v_1) and the steady-state probability ϕ_v of being active, as explained in Section 3.1.3:

$$1 - \alpha = \frac{\phi_v}{b_v(1-\phi_v)}, \tag{4.2}$$

$$1 - \beta = \frac{1}{b_v}. \tag{4.3}$$

A task is removed from the queue once its execution is complete. Denoting by $c_E(t)$ the remaining time slots required for execution, the execution indicator $e(t)$ is given by

$$e(t) = \begin{cases} 1, & \text{if } c_E(t) = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

4.2.2 Wireless Channel model

The wireless link between the CAV and the edge server is assumed to be affected by Rayleigh fading. To obtain a finite-state representation of the channel gain envelope $r(t)$, we rely on the Markov chain model of the fading process presented in Section 3.2.1, where the chain states are obtained by quantizing $r(t)$ through a finite set of intervals.

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, being N_r the desired number of states. Following [61], 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

$$\int_{\Gamma_i}^{\Gamma_{i+1}} f_r(r)dr = 1/N_r, \tag{4.5}$$

where $f_r(r)$ is the Rayleigh probability density function (pdf). Solving Eq. (4.5) yields

$$\Gamma_i = \sqrt{-2\sigma^2 \ln\left(1 - \frac{i}{N_r}\right)}. \tag{4.6}$$

Given the partitioning scheme in (4.6), we say that the chain is in state r_i if $r(t) \in [\Gamma_i, \Gamma_{i+1})$ and the channel output is given as the midpoint of the interval.

To define the transition probability matrix $\mathbf{P_R}=(p_{R,ij})_{i,j=1}^{N_r}$, we follow the approach in [61], which allows capturing the second-order properties of the fading process. In particular, we recall that for the autocorrelation function of the underlying complex Gaussian fading process we adopt the well known Clarke's model, where the correlation between two consecutive fading samples is given as $J_0(2\pi f_D T)$, with f_D being the maximum Doppler shift and T the observation time between two samples, which is assumed to be equal to the time slot duration.

Note that the channel state affects the opportunity of performing tasks offloading. Indeed, assuming for simplicity a fixed rate R_c supported by a specific Modulation and Coding Scheme (MCS), to guarantee reliable data transfer, the experienced Signal-to-Noise Ratio (SNR) should be above a given minimum level. If not, task offloading cannot take place.

4.2.3 Radio resources allocation

To offload a task, the CAV has to request the base station for the necessary radio resources. Let $n(t)\in\{n_0, n_1\}$ be a binary flag indicating the availability of radio resources at slot t: if $n(t)=n_0$, no radio resources are readily available; otherwise, if $n(t)=n_1$, a radio resource can be allocated. Assuming the CAV demands radio resources and the system is in n_0, a slot counter $c_N(t)$ is initialized to T_g. When $c_N(t)$ reaches 0, then $n(t)$ moves to state n_1 with probability p_g, corresponding to a successful resource assignment in that specific time slot. Conversely, when $n(t)$ stays in n_0 (which happens with probability $1 - p_g$), no offload can take place due to the lack of available resources. Then, the above procedure may repeat. Notice that, when in n_1, the user can offload the task to the server only if the SNR is above the minimum threshold for the previously selected MCS, otherwise the offload fails.

4.2.4 Inference time and accuracy

The execution time of an inference task depends on whether computation occurs at the CAV or at the edge. A local task execution has a duration T_l, due solely to the computation at the CAV; on the contrary, remote execution also involves offloading the task, which increases the overall latency.

Furthermore, since the edge server relies on EE, it can choose among several exits of the DNN, each exit e (e=1, ..., M) yielding a different level of accuracy, A_e, and execution time, T_e. Notice that we have: $T_1<T_2<\ldots<T_M$ and $A_1<A_2<\ldots<A_M$. Additionally, the execution time at the edge is affected by the current computational load the edge server is experiencing. To account for the case where the edge server might not have ready-to-use computing capabilities, we define the server state $s(t)\in\{s_0, s_1\}$, with s_0 and s_1 representing idle and busy computational resources, respectively. An additive random delay $d(t)$ is then associated with each state and characterized by the probabilities $\Pr(d(t)=0|s(t)=s_0)=1$ and $\Pr(d(t)=\tau|s(t)=s_1)=p(\tau)$, where $\tau\in\{0, 1, \ldots, \tau_{\max}\}$ and $p(\tau)$ is assumed to be a truncated decreasing geometric probability mass function (pmf).

The dynamic of $s(t)$ is modelled through a Markov chain with transition probability matrix $\mathbf{P_S}=\left(\begin{smallmatrix}\gamma & 1-\gamma\\ 1-\phi & \phi\end{smallmatrix}\right)$, where $\gamma=\Pr(s(t+1)=s_0|s(t)=s_0)$ and $\phi=\Pr(s(t+1)=s_1|s(t)=s_1)$. Similarly to the task generation model, transition probabilities can be expressed as a function of the burstiness b_s and the steady state distribution ϕ_s of state s_1.

Depending on the selected exit and the server state, the remote execution time is thus given by:

$$T_{\text{edge}}(t) = T_e + d(t). \tag{4.7}$$

4.3 Actions Set and System Behaviour

The overall system state $x(t)$ at time slot t is defined as

$$x(t) = (q(t), v(t), s(t), r(t), n(t), c_E(t), c_N(t)). \tag{4.8}$$

The time evolution of $x(t)$ is driven by uncontrollable system dynamics (such as the channel state, task generation, computational resource availability, etc.) as well as the undertaken action a_e. We denote with $\mathcal{A} = \{a_e\}_{e=0}^{M+2}$ the set of possible actions. In particular, action a_e, with $e=1,\dots,M$, represents the execution up to exit e deferred to the edge; $a_{M+1}=a_l$ indicates a local execution of the task; $a_{M+2}=a_g$ defines the action of making a request for a radio resource; finally, $a_0=a_w$ is the action corresponding to the CAV doing nothing. According to the defined action set, we can distinguish the following relevant cases:

$c_E(t)=0$, $c_N(t)=0$, $n(t)=n_0$ in this case the system is in idle state and a task waiting in the queue can be executed. However, since $n(t)=n_0$, there are no radio resources available and only local computing, resource request or waiting are admissible actions. By selecting action a_l, the task is executed locally and $c_E(t)$ is initialized to T_l. Denoting with $p(x'|x,a)=\Pr(x(t+1)=x'|x(t)=x, a(t)=a)$ the probability of moving into state x' from state x by taking action a, we can write the possible state transitions as follows:

$$p((q+1, v_1, s_1, r_j, n_0, T_l, 0)|(q, v_0, s_1, r_i, n_0, 0, 0), a_l) = (1-\alpha)\phi p_{R,ij}, \tag{4.9}$$

with similar transition probabilities that can be written in both the above cases for different values of states $v(t)$ and $s(t)$.

Instead, a radio resource can be requested by selecting action a_g, which allows transitions as

$$p((q, v_0, s_1, r_j, n_0, 0, T_g)|(q, v_0, s_1, r_i, n_0, 0, 0), a_g) = \alpha\phi p_{R,ij}. \tag{4.10}$$

$c_E(t)=0$, $c_N(t)=0$, $n(t)=n_1$ in this case also remote computing is possible if the SNR is high enough for the selected MCS. Then an action a_e, with $e=1,\dots,M$, can be selected, thus yielding transition probabilities of the type

$$p((q, v_0, s_0, r_j, n_0, T_e, 0)|(q, v_0, s_0, r_i, n_1, 0, 0), a_e) = \alpha\gamma p_{R,ij}. \tag{4.11}$$

Moreover, if the server is busy (i.e., $s(t)=s_1$), an additional random delay is required for obtaining computing resources, which yields the following state transitions:

$$p((q, v_0, s_1, r_j, n_0, T_e + \tau, 0)|(q, v_0, s_1, r_i, n_1, 0, 0), a_e) = \alpha\phi p(\tau) p_{R,ij}. \tag{4.12}$$

$c_E>0$ **or** $c_N>0$ in such situations, the system is either executing a task, locally or remotely, or counting the time spent for requesting a radio resource; it follows that the only possible action is a_w. Additionally, at any time slot either $c_E(t)$ or $c_N(t)$ decrease by 1, thus yielding the transition probabilities

$$p((q, v_0, s_0, r_j, n_0, N-1, 0)|(q, v_0, s_0, r_i, n_0, N, 0), a_w) = \alpha\gamma p_{R,ij}. \tag{4.13}$$

When $c_E(t)=1$, according to the behavior of the queue modeled in Eq. (4.1), we have

$$p((q-1, v_0, s_0, r_j, n_0, 0, 0)|(q, v_0, s_0, r_i, n_0, 1, 0), a_w) = \alpha\gamma p_{R,ij}. \tag{4.14}$$

Instead, when $c_N(t)=1$, a radio resource could not be assigned in the next time slot with probability $1-p_g$, giving the following transitions:

$$p((q, v_0, s_1, r_j, n_0, 0, 0)|(q, v_0, s_1, r_i, n_0, 0, 1), a_w) = \alpha\phi(1-p_g) p_{R,ij}, \tag{4.15}$$

or a radio resource can be assigned with probability p_g allowing the following transitions:

$$p((q, v_0, s_1, r_j, n_1, 0, 0)|(q, v_0, s_1, r_i, n_0, 0, 1), a_w) = \alpha\phi p_g p_{R,ij}. \tag{4.16}$$

4.4 Optimal Execution Policy

We now derive an optimal action selection strategy that maximizes the average inference accuracy while meeting the constraints on the average execution time and the task discarding rate. To do so, we formulate an optimization problem by exploiting the above MDP framework, and solve it through Linear Program (LP) mapping – a well-established and effective solution approach [56].

Without loss of optimality we limit our search within the class of past-independent randomized policies $\mu\colon \mathcal{X}\times\mathcal{A}\to[0,1]$, where $\mathcal{X}$ is the set of possible system states and $\mu(a|x)=\Pr(a(t)=a|x(t)=x)$ is the probability of taking action $a\in\mathcal{A}$ when in state $x\in\mathcal{X}$. Note that the distribution of the action is independent of the past history of the system, so that the state-action sample paths are Markovian with transition probabilities $p(x'|x,a)$. If the Markov process is ergodic (i.e., recurrent aperiodic for any control policy [50]), then time averages converge to state-space averages. Thus, by denoting with $A(x,a)$ the accuracy generated by taking action a when being in state x, we can express the expected accuracy as

$$\overline{A} = \sum_{x\in\mathcal{X}} \sum_{a\in\mathcal{A}} A(x,a)\pi_\mu(x,a), \tag{4.17}$$

where $\pi_\mu(x,a)$ is the joint steady state distribution of the states and the actions, and $A(x,a)=0$ for actions that do not correspond to the execution of an inference (i.e., a_w and a_g).

Next, following [19], we define the average execution time $\overline{T}_{EX}$ as the average total queuing time, which encompasses the execution time along with idleness epochs and the time required for

radio resource assignment. Formally, $\overline{T}_{EX}$ is given as the ratio between the fraction of time a task remains in the queue and the fraction of executed tasks, that is

$$\overline{T}_{EX} = \frac{\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} r_1(x,a)\pi_\mu(x,a)}{\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} r_2(x,a)\pi_\mu(x,a)}, \tag{4.18}$$

where

$$r_1(x,a) = \begin{cases} 1 & \text{if } a = a_w, a_g, \\ 0 & \text{otherwise,} \end{cases} \tag{4.19}$$

is an indicator function for the count of time slots in which execution-related actions are not chosen. By recalling that waiting (i.e., a_w) is the only admissible choice during both running executions (i.e., $c_E(t) > 0$) and radio resource negotiation (i.e., $c_N(t) > 0$), then $r_1(x,a)$ inherently accounts for the time spent in executing a task (either locally or remotely) and the time spent in obtaining radio resources. Similarly,

$$r_2(x,a) = \begin{cases} 1 & \text{if } a \neq a_w, a_g, \\ 0 & \text{otherwise,} \end{cases} \tag{4.20}$$

is a flag indicating whether a task is executed or not.

The optimal randomized policy for maximizing the average accuracy can be obtained by solving the following LP:

$$\underset{\mu}{\text{argmax}} \quad \sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} A(x,a)\pi_\mu(x,a) \tag{4.21}$$

$$\text{s.t.} \quad \frac{\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} r_1(x,a)\pi_\mu(x,a)}{\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} r_2(x,a)\pi_\mu(x,a)} \leq t_c \tag{4.22}$$

$$1 - \frac{\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} r_2(x,a)\pi_\mu(x,a)}{\phi_v} \leq d_r \tag{4.23}$$

$$\sum_{a\in\mathcal{A}} \pi_\mu(x',a) - \sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} \pi_\mu(x,a)p(x'|x,a) = 0, \forall x' \tag{4.24}$$

$$\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} \pi_\mu(x,a) = 1 \tag{4.25}$$

that yields the optimal steady state distribution $\pi^\star_\mu(a,x)$. Constraint (4.22) states that the average execution time cannot exceed a maximum tolerable value t_c. Constraint (4.23) imposes that the task dropping probability is lower than the target value d_r. Indeed, the ratio between the probability of executing a task and the probability of generating a task ϕ_v represents the fraction of tasks generated and effectively executed, while the left-hand side term in (4.23) represents the probability of not executing a generated task. Finally, constraints (4.24)–(4.25) come from the mapping of the MDP into the LP [56], where (4.25) represents a flow-balance constraint.

Finally, the optimal policy $\mu^\star(a|x)$ can be obtained as

$$\mu^\star(a|x) = \frac{\pi^\star_\mu(a,x)}{\sum_{a\in\mathcal{A}} \pi^\star_\mu(a,x)}, \tag{4.26}$$

Table 4.1: Accuracy and execution times associated with the different actions

	a_1	a_2	a_3	a_l
A_e	88.9	92.9	93.84	72.4
T_e [ms]	2	3	4	10

Table 4.2: Simulation parameters

Parameter	Value	Parameter	Value
$Q_{\max}$	10	T_g [ms]	5
b_s	5	t_c [ms]	30
b_v	5	d_r	0.1
$\tau_{\max}$ [ms]	15	N_r	4

that is, as the optimal probability of choosing action a, conditioned on the system being in state x.

4.5 Numerical Results

In this section, we evaluate the system performance when our optimal executing strategy is applied. The values of inference execution times and accuracy are presented in Tab. 4.1: for the early exiting model, they refer to a modified ResNet56 with 3 exits trained on the CIFAR-10 dataset [38], while, for the local inference, they refer to the MobileNetV2 model executed on a mobile device [70]. All relevant system parameters considered for results derivation are listed in Tab. 4.2 and are kept fixed, unless otherwise stated.

To find the set of channel states limiting the offloading procedure for the different MCS, we considered the SNR values reported in [28]. In particular, we fixed the transmission rate of the CAV, the transmit power, and the noise power. Then, depending on the value of the channel envelope power associated with each channel state, we identified a set of SNRs and identified which of the channel states allow for a sufficient MCS that can honor the rate constraint R_c.

We also remark that, since a non zero value of accuracy is obtained only if the task is executed, we normalize the maximum accuracy obtained under the optimal policy with respect to the fraction of tasks for which an inference is performed. We do so by defining

$$\overline{A} = \frac{\sum_{x\in\mathcal{X}} \sum_{a\in\mathcal{A}} A(x,a)\pi^{\star}_{\mu}(x,a)}{\sum_{x\in\mathcal{X}} \sum_{a\in\mathcal{A}} r_2(x,a)\pi^{\star}_{\mu}(x,a)}, \tag{4.27}$$

which is considered as the reference metric for performance assessment.

Fig. 4.2 shows the average accuracy as a function of the steady state probability of the application being in active state (ϕ_v) and for different values of the busy probability of the server (ϕ_s). Moreover, a comparison between the derived optimal policy and a benchmark scenario where the edge can only execute the full DNN (i.e., no EE is possible) is reported. In the latter case, we set the execution time and accuracy as in the third column of Tab. 4.1. Referring to the EE-based policy, we can observe how the obtained accuracy matches the DNN highest performance for low values of ϕ_v. However, when the task generation rate increases (higher values of ϕ_v), the accuracy starts slightly decreasing, since the action policy favors earlier exits of the DNN model to avoid queue

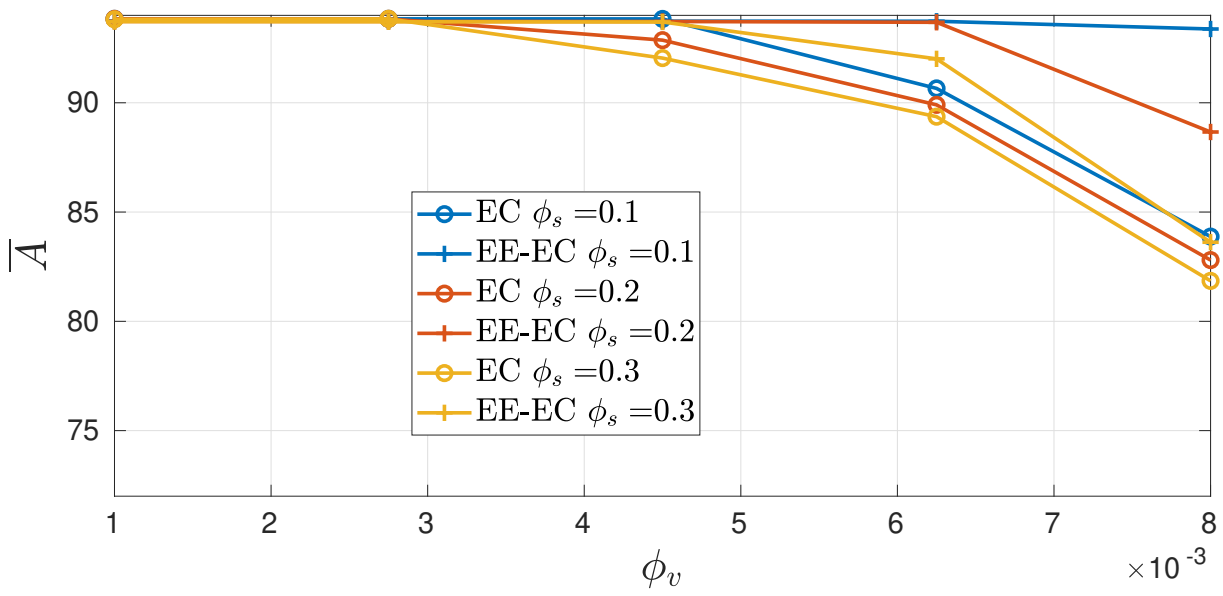


Figure 4.2: Average accuracy as a function of ϕ_v, as obtained with Edge Computing with and without Early Exiting, and for various settings of ϕ_s. It is assumed $p_g = 0.8$, $f_D T = 0.01$ and $R_c = 15$ Mbps.

overflow and meet the constraint on the task dropping rate. Then, as ϕ_v further increases, the system starts avoiding radio resource requests by prioritizing local inference (hence, avoiding the time to offload the task), which causes additional accuracy drop. These effects are exacerbated when the server availability is exiguous (high values of ϕ_s), since remote task execution is affected by delayed assignment of the computing resources. Importantly, when the task generation rate at the CAV grows (high values of ϕ_v), the traditional edge computing approach with no EE favors local inference, while our approach exploits the earlier classifiers at the edge server, thus reaching higher accuracy. This effect is substantial when the edge availability rate is high (i.e., low values of ϕ_s), with our policy producing an accuracy gain of 11% if compared to the traditional edge computing scheme.

Next, Fig. 4.3 shows how the running application requirements affect the average accuracy. In particular, the plot reports the average accuracy as a function of the constraint on the average execution time t_c and for different values of the target maximum dropping rate d_r. As expected, by relaxing the application constraints, performance improves, since the optimal policy favours remote execution to maximize accuracy. Furthermore, the results highlight the flexibility of the early exiting paradigm in satisfying the application constraints: under stringent constraints, remote execution is still viable by exploiting the earlier classifiers.

Finally, Fig. 4.4 illustrates how the inference executing strategy reacts to different channel dynamics and network conditions. The plot reports the average accuracy as a function of the probability of obtaining a radio resource p_g, and for different regimes of channel correlation $f_D T$. Two different MCSs (i.e., different rates R_c) are considered for the same channel partitioning scheme. As expected, when the probability of obtaining a radio resource is low, local execution is predominant: this allows the system to honor the target task dropping rate, at the cost of a lower accuracy value. Instead, as p_g increases, the CAV offloads the task more frequently, so as to maximize the accuracy. We can also observe that transmitting at higher rate requires a higher minimum SNR and this may prevent successful offloading, hence leading to a longer waiting time before a task can be actually dispatched to the server. In this case, the action policy is prone to opt for local computing thus

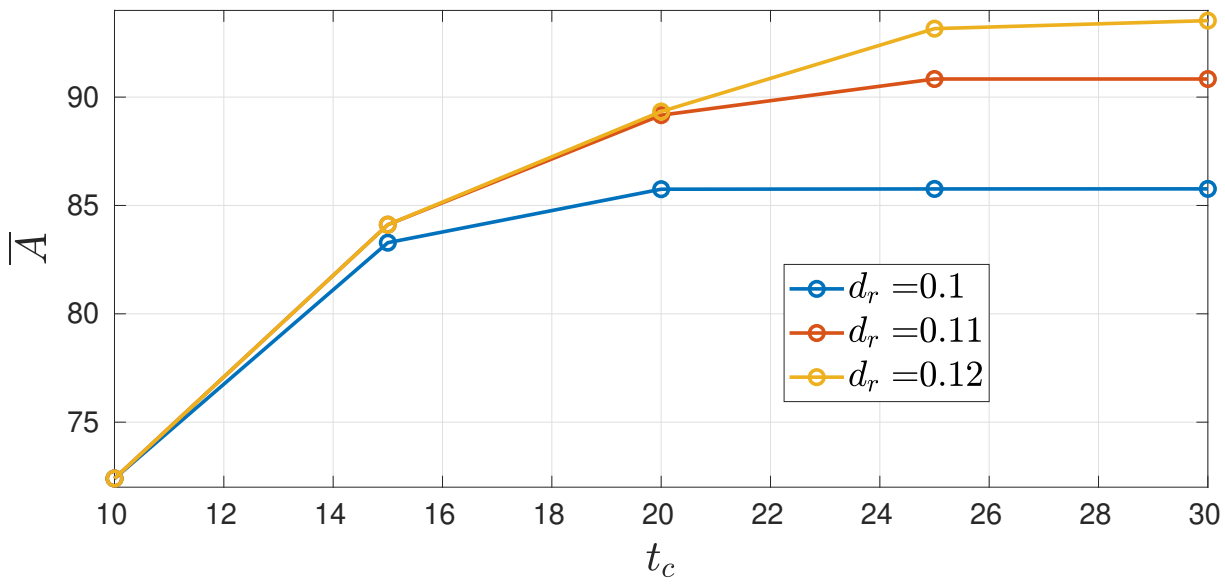


Figure 4.3: Average accuracy as a function of t_c and for various settings of d_r. It is assumed $\phi_v = 5\text{e}-3$, $\phi_s = 0.8$, $p_g = 0.6$, $f_D T = 0.01$ and $R_c = 30\,\text{Mbps}$.

reducing the attained accuracy. This highlights the importance of carefully adapting the MCS according to the wireless link conditions, since channel impairments may induce severe degradation of the attained accuracy if the transmission rate is selected regardless of the propagation phenomena. Finally, looking at the impact of the channel dynamics, one can observe how a higher accuracy level can be achieved when the channel exhibits low correlation, since in this case the channel tends to remain in unfavorable states for a shorter time. This effect is even more evident with higher rate constraints.

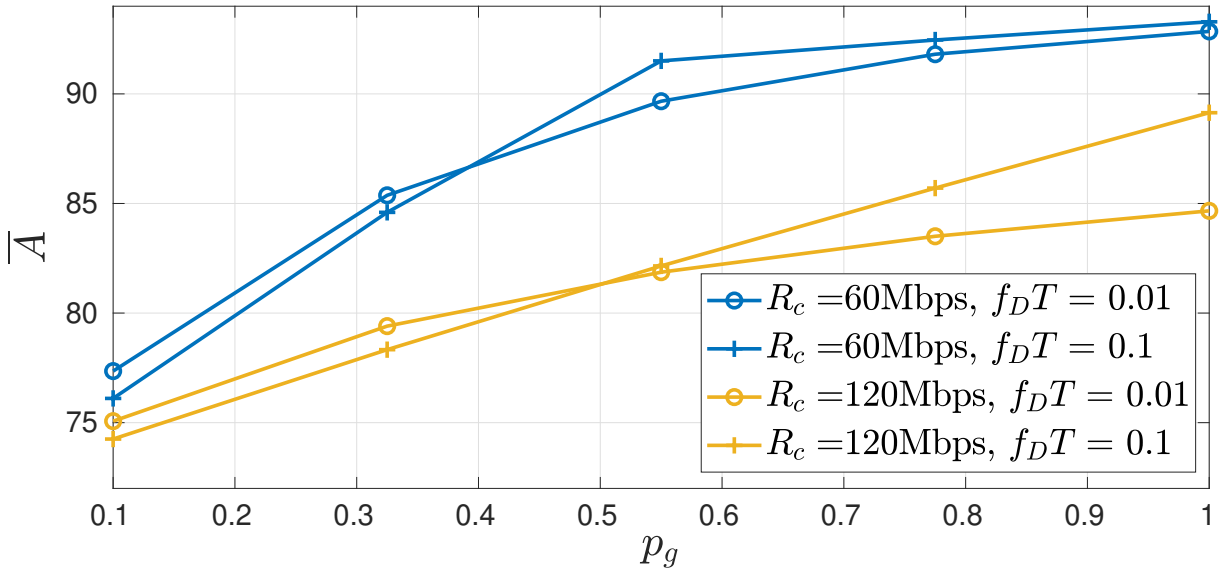


Figure 4.4: Average accuracy as a function of p_g and for various settings of R_c, and $f_D T$. It is assumed ϕ_v=0.001 and ϕ_s=0.5.

4.6 Conclusions

We considered mobile devices that have to execute DNN tasks, either locally or by offloading them to the network edge. By leveraging both the Early Exiting and the Edge Computing paradigms, and modeling their interaction through a Markov Decision Process, we derived a low-complexity offloading policy that maximizes the tasks accuracy. We investigated the benefits of our policy in a scenario where automated vehicles are connected with an edge server. Results show that early exiting provides high flexibility in adapting the performance to different operational conditions and application-specific requirements. In particular, the obtained accuracy level can be tailored to the task generation rate at the vehicles, to the computing capabilities available at the server, and to the channel conditions and network load, thus demonstrating that Early Exiting can be a powerful tool to optimally orchestrate DNN tasks in dynamic scenarios.

Chapter 5

Distributional RL for Task Offloading, Resource Allocation and Early Exit Selection at the Edge [11]

In this second chapter of contributions, we present an extension of our previous work. The motivation for this extension arises from the consideration of a more complex scenario. In our previous work, we analyzed a simplified setting consisting of a single CAV and an edge server. In the new scenario, we consider multiple CAVs with heterogeneous task requirements.

Moreover, while our prior study focused solely on the problem of execution decision-making and early exit selection in the case of remote execution, the current work addresses a more comprehensive problem that also includes the assignment of computational and radio resources, known as Task Offloading and Resource Allocation (TORA) problem. However, due to the increased state space and the presence of a hybrid action space, our previous solution approach was no longer viable. To tackle this complexity, we employ a DistRL method, which additionally allows us to quantify the variability of the learned control policy. Finally, we further refine the typical system model adopted in edge computing scenarios by including communication-related details that are often overlooked in the literature.

The results and methodology presented in this chapter correspond to the work reported in [11].

5.1 Introduction

The reference scenario extends the one presented in the previous chapter, including a set of CAVs, rather than only one CAV, performing inference tasks, with different application requirements, by relying on edge servers enhanced with Early Exiting (EE) capabilities.

While existing works have highlighted the potential of EE to reduce inference latency and computational load, they typically consider EE in isolation or in simplified settings, without embedding it into a comprehensive TORA framework. This leaves open the fundamental question of how EE should be jointly optimized with offloading and resource allocation decisions in realistic, resource-constrained edge environments. This topic remains largely unexplored, except for our previous work [10] and another study [58].

Although the integration of EE into edge computing systems holds clear potential benefits in

terms of performance improvement, modeling such architectures leads to challenging optimization problems due to the large number of system variables and the hybrid nature of the decision space. To tackle these challenges, we adopt a novel solution approach capable of not only optimizing system performance but also quantifying the uncertainty of the learned control policy. Additionally, we propose a refined system model that captures important aspects often oversimplified in prior studies. A centralized controller is assumed to orchestrate the system, making decisions on task offloading, DNN exit point selection, radio resource allocation, and computational resource assignment according to a defined policy. In this context, the main contributions of this work are as follows.

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.

While our previous work [10] 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 DistRL approach, which were not considered in our previous study.

The main contributions of this work are summarized as follows:

- 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 chapter is organized as follows. In Section II, we provide an overview of the related literature and further clarify the specific contributions of this work. Section III introduces the system modeling, describing in detail the assumptions, the task and network models, and the considered resources. Section IV presents the algorithmic approach adopted to solve the formulated problem, including the rationale behind the chosen methodology. In Section V, we report and analyze the obtained results, highlighting the performance improvements and insights gained from the proposed framework. Finally, Section VI summarizes the conclusions and discusses possible directions for future research.

5.2 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 [14, 15, 21, 72], while others differentiate between single-edge server deployments [14, 15] and multi-edge server architectures [29, 72]. In addition, several studies incorporate fog and cloud computing nodes into the system model [36, 40, 42], 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 [71] 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 [15], [21], [44], [36]. Some studies further incorporate attributes

like task priority [29], as well as Random Access Memory (RAM) and Graphical Processing Unit (GPU) resource requirements [39].

Resource handling Remote task execution requires various resources, primarily radio and computational. A prevalent modeling strategy assumes a fixed or discretized pool of remote resources, framing the problem as one of allocating them among users [14, 15, 21, 29, 44, 71, 72]. Alternatively, some works consider resources as constant and pre-allocated [15], shifting the focus to deciding whether tasks should be executed locally or offloaded. More dynamic approaches model resource availability through queue-based systems [40], which capture workload variations over time. Additional constraints such as RAM and storage [14, 40, 42] are also considered, and energy efficiency often emerges as a critical optimization objective alongside performance [15, 21]. 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 [14, 15, 21, 29, 44, 71, 72]. Another common option is to assume that radio or computing resources are constant [15]. In such a case, no resources 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 [40], 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 [14, 40, 42], and energy, whose usage is usually minimized [15, 21].

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 [15, 21, 29, 30, 36, 44, 71]. In most of these works, channel gains are assumed to follow a Rayleigh fading model and to be temporally independent. In [36], 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 [14, 40, 42, 45] 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 [42], minimizing task execution time or overall system latency [44], and reducing latency weighted by task priority [15] or model precision [72]. Other works focus on maximizing the probability of task admission at edge servers [14]. Furthermore, several studies adopt composite metrics, such as joint optimization of task throughput and latency [39], task throughput and energy consumption [15], or latency and energy consumption [36, 71], 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 dynam-

ics, Deep Reinforcement Learning (DRL) methods are commonly employed, as demonstrated in [15, 21, 36, 72]. In contrast, when resource availability is modeled using queue dynamics, and stability is a concern, Lyapunov optimization is often adopted [29, 30, 39, 40, 42]. Alternatively, less commonly used approaches include hybrid methods that combine clustering techniques with convex optimization based on the Karush-Kuhn-Tucker (KKT) conditions [44], bio-inspired heuristics such as Grey Wolf Optimization [71], and submodular optimization frameworks [14].

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, [41], [34], and [46] jointly consider early exit selection and model partitioning while optimizing different performance metrics. Other contributions extend the control space to include computational resource assignment [24, 72], or additional model-level adaptations such as quantization [53].

5.2.1 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. Specifically, early exit selection is never jointly optimized with all the core TORA decisions, namely task execution and offloading, radio resource allocation, and computational resource assignment. Existing studies either focus on EE and offloading only, or assume simplified and fixed resource configurations. In contrast, this work explicitly studies the integration of early exiting within the TORA problem, and formulates a unified optimization framework, referred to as TORA-EE, in which early exit decisions, offloading choices, and resource allocation are simultaneously optimized. To the best of our knowledge, this is the first work that treats EE as an integral component of the TORA decision-making process.

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 [15, 21, 29, 30, 36, 44, 71], 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 [10]. 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 [15, 21, 36, 72]. 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

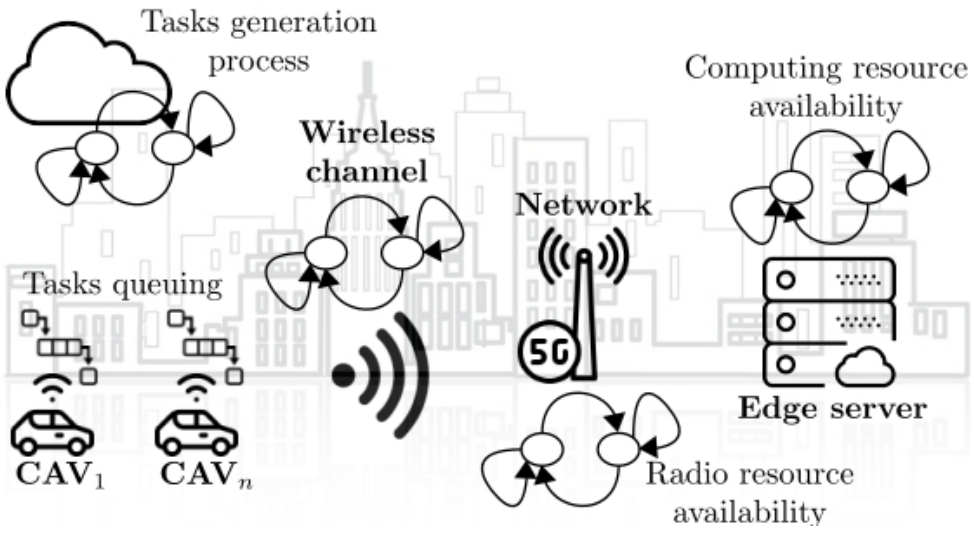


Figure 5.1: Considered system scenario.

(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 Reinforcemnt 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.

5.3 System Model

We consider the system scenario depicted in Fig.5.1, 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 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, Tab. 5.1 provides an overview of the adopted notation and its corresponding definitions.

Table 5.1: 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

5.3.1 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 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 $e \in \{1, \ldots, N_{\epsilon,e}\}$, specifies the number of Million Operations (MOP) required for executing the task when exit e of the DNN model required for task 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 $a_{\epsilon,1} < \cdots < a_{\epsilon,N_\epsilon}$.

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 $m = 0, 1, \ldots,$ is given by

$$\Pr(i_n(t + mL) = k) = \frac{\exp(-\zeta_n mL)(\zeta_n mL)^k}{k!}, \tag{5.1}$$

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

Each CAV n stores tasks to be executed in a queue $\boldsymbol{q}_n(t) = [\tau_{n,1}(t), \tau_{n,2}(t), \ldots, \mathbf{0}_n(t)] \in \mathbb{N}^{Q_{\max}}$ with finite capacity $Q_{\max}$, where $\tau_{n,i}(t)$ 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. 5.2 illustrates an example of the queue dynamics. Note that if a task i is generated at time slot t_0 at CAV n, then $\tau_{n,i}(t_0) = \tau_n$, $\tau_{n,i}(t_0 + 1) = \tau_n - 1$ and so on until the task is removed from the queue due to task execution or task expiration, which occurs when $\tau_{n,i}(t) \leq 0$.

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

$$q_n(t + 1) = \min\{\max\{q_n(t) - u_n(t), 0\} + i_n(t), Q_{\max}\}, \tag{5.2}$$

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.

5.3.2 Wireless Channel Model

We assume that the wireless link between the n-th CAV and the base station (gNodeB) connected to the edge server is subject to small-scale fading, described by a complex channel gain $h_n(t) = h_{I,n}(t) + jh_{Q,n}(t)$, 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, $r_n(t) = \sqrt{h^2_{I,n}(t) + h^2_{Q,n}(t)}$, 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.

As in our previous contribution, we adopt the FSMC model of the fading process with the same partioning scheme for the channel gain $r_n(t)$. Same considerations applies for the evaluation of the transition probability matrix $\mathbf{P_R}$=$(p_{R,ij})_{i,j=1}^{N_r}$.

However, differently from our first contribution, we emphasize the critical role of the channel state 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.

Importantly, to adapt the transmission rate, each CAV dynamically selects the Modulation and Coding Scheme (MCS) based on $r_n(t)$ [33]. 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, with lower target BLER values the system could exploit higher MCSs, while with worse target BLER values, the system should rely on lower MCSs in order to counteract the larger transmission failures. The impact of this approximation is quantitatively assessed in Sec. V. We referred to [64] 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.

5.3.3 Radio Resources 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 $\{W_0, \ldots, W_{N_w}\}$, 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 [4]

$$R_{\text{off},n}(w_n(t), r_n(t)) = \sum_{j=1}^{J} v_{\text{layer}}^{(j)} Q_m^{(j)} f^{(j)} R_{\max} \frac{12 w_n(t)}{T_s^{\mu}} (1 - \text{OH}^{(j)}), \tag{5.3}$$

where J is the number of aggregated component carriers, $v_{\text{layer}}^{(j)}$ 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, equivalentely, 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

$$T_{\text{off},n}(t) = \frac{\delta_n}{R_{\text{off},n}(t)}. \tag{5.4}$$

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.

5.3.4 Computational Resources Allocation

The edge server shares its total amount of computational resources $c(t)$, which are expressed in terms of FLOPs per seconds (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

$$T_{\text{loc},n}(t) = \frac{c_{\epsilon,e}}{C_n}. \tag{5.5}$$

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

$$T_{\text{rem},n}(t) = T_{\text{off},n}(t) + \frac{c_{\epsilon,e}}{c_n(t)}, \tag{5.6}$$

where $T_{\text{off},n}(t)$ denotes the time required to transmit the task to the edge server as defined in Eq. (5.4).

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

$$\boldsymbol{x}(t) = [\boldsymbol{q}_1(t), \dots, \boldsymbol{q}_{N_u}(t), \boldsymbol{r}(t), w(t), c(t)], \tag{5.7}$$

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

5.4 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.

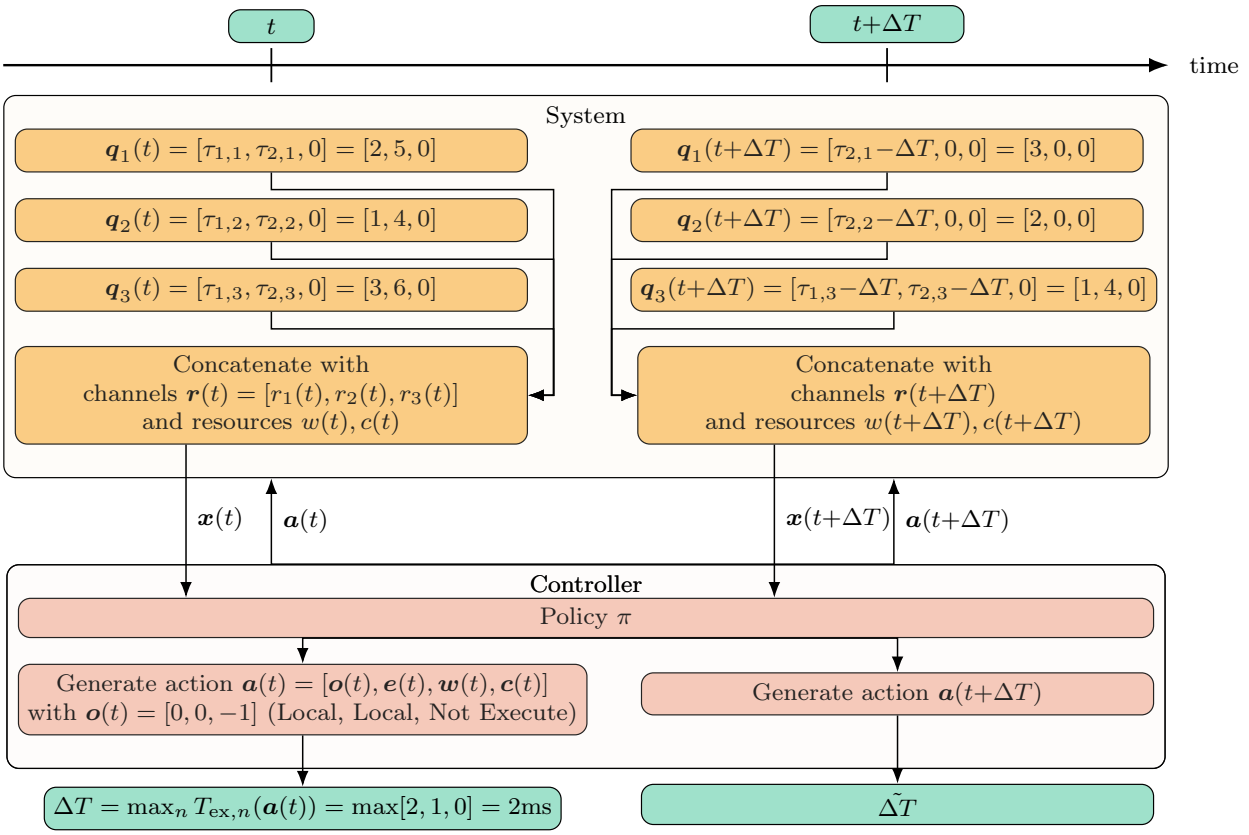


Figure 5.2: Example of queue evolution for 3 users. Each queue is a fixed-length vector of remaining time slots, denoted by $\tau_{i,n}$, with i the queue position and n representing the user. The controller observes the system state $\boldsymbol{x}(t)$ and outputs the action $\boldsymbol{a}(t)$. In this example, the controller decides to execute locally the tasks of the users 1 and 2, while for user 3 the system preferes to not 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.

We firstly 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.

5.4.1 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 and its execution is deferred to subsequent time slots. 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. 5.2 illustrates this behavior. Moreover, let $e_n(t) \in \{1, \ldots, N_{n,e}\}$ 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 an optimal policy $\boldsymbol{a}(t)$ that balances model accuracy and execution la-

tency, while ensuring reliable system operation by preventing queue overflows, we first define the instantaneous utility function $\psi(t)$ we want to maximize as

$$\psi(t) = \sum_{n}^{N_u} \nu_1 \frac{A_n(e_n(t))}{A_{\max,n}} + \nu_2 \frac{T_{\text{ex},n}(\boldsymbol{a}(t))}{\tau_n}, \tag{5.8}$$

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 $\boldsymbol{a}(t)$, defined as

$$T_{\text{ex},n}(\boldsymbol{a}(t)) = \begin{cases} T_{\text{loc},n}(t), & \text{if } o_n(t) = 0, \\ T_{\text{rem},n}(t), & \text{if } o_n(t) = 1. \end{cases} \tag{5.9}$$

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:

$$\underset{\boldsymbol{a}(t)}{\text{argmax}} \quad \lim_{T\to\infty} \frac{1}{T} \sum_{t}^{T} \psi(t) \tag{5.10}$$

$$\text{s.t.} \quad T_{\text{ex},n}(\boldsymbol{a}(t)) \leq \tau_{1,n}(t), \forall n, t \tag{5.11}$$

$$q_n(t) \leq Q_{\max}, \forall n, t \tag{5.12}$$

$$\sum_{n} w_n(t) \leq w(t), \forall t \tag{5.13}$$

$$\sum_{n} c_n(t) \leq c(t), \forall t \tag{5.14}$$

$$\xi_n(t) \in [0, 1], \forall n, t \tag{5.15}$$

$$\rho_n(t) \in [0, 1], \forall n, t \tag{5.16}$$

$$o_n(t) \in \{-1, 0, 1\}, \forall n, t \tag{5.17}$$

$$e_n(t) \in \{1, \ldots, N_{n,e}\}, \forall n, t \tag{5.18}$$

where the constraint in Eq. (5.11) ensures that the task of user n is executed within the remaining deadline $\tau_{1,n}(t)$, the constraint in Eq. (5.12) guarantees that queues storing the tasks do not overflow whereas the constraints in Eqs. (5.13)–(5.14) ensure that the allocation of radio and remote computational resources does not exceed their limits. Finally, the constraints in Eqs. (5.15)–(5.18) limit the search space of the decision variables.

5.4.2 A Novel Solution based on Deep Distributional Reinforcement Learning

The TORA-EE problem, defined in Eqs. (5.8)–(5.18), 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 previously proposed, where a significantly simpler problem was addressed by leveraging a standard procedure for mapping a MDP into a 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.

In particular, we leverage Distributional Reinforcement Learning (DistRL) [12], introduced in Chapter 2, which models the full probability distribution of returns rather than just their expected value. By capturing the entire return distribution, DistRL provides a richer representation of system behavior and allows the design of risk-aware policies that account for variability or worst-case outcomes, not just average performance. This feature is particularly valuable in scenarios where minimizing variability or mitigating high-risk events is as important as optimizing the mean reward.

Specifically, to solve our optimization problem, we adopted a distributional variant of the SAC presented in Section 2.3.1, specifically the Distributional SAC with Three Refinements (DSAC-T) [26]. DSAC-T extends the original DSAC algorithm [25] 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 $\mathcal{Z}_\theta(x, a)$ 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

$$\bar{\theta} \leftarrow \eta\theta + (1 - \eta)\bar{\theta}, \tag{5.19}$$

and

$$\bar{\phi} \leftarrow \eta\phi + (1 - \eta)\bar{\phi}, \tag{5.20}$$

where η is the update rate.

As the classic SAC algorithm, the DSAC-T is also a *maximum entropy* RL algorithm, since the random return encompasses a term representing the entropy of the policy π. Then, the entropy-augmented discounted cumulative random return, when following a policy π being in state x_t, is expressed as

$$G^\pi(x_t) = \sum_{k=0}^{\infty} \gamma^k [R_{t+k} + \alpha\mathcal{H}(\pi(\cdot|X_{t+k}))], \tag{5.21}$$

where

$$\mathcal{H}(\pi(\cdot|x_t)) = \mathbb{E}\Big[-\log \pi(a_t|x_t)\Big] \tag{5.22}$$

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

$$\alpha \leftarrow \alpha - \beta_\alpha \, \mathbb{E}[-\log \pi(a_t|x_t) - \bar{\mathcal{H}}], \tag{5.23}$$

where β_α is the learning rate and $\bar{\mathcal{H}}$ is a target entropy.

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

$$J_{\mathcal{Z}}(\theta) = \mathbb{E}\Big[D_{\mathrm{KL}}((\mathcal{T}^{\pi_{\bar{\phi}}}\mathcal{Z}_{\bar{\theta}})(x,a), \mathcal{Z}_\theta(x,a))\Big], \tag{5.24}$$

where $D_{\mathrm{KL}}(\cdot)$ is the Kullback-Leibler divergence, which represents the distance between two probability distributions, $\mathcal{T}^{\pi_{\bar{\phi}}}$ is the distributional Bellman operator referred to the policy learned by the target actor network of parameters $\bar{\phi}$, $\mathcal{Z}_{\bar{\theta}}(x,a)$ and $\mathcal{Z}_\theta(x,a)$ are the return distribution learned by the critic networks and $\pi_{\bar{\phi}}$ 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. (5.24) is then rewritten in a sample-based form, for the critic network i, as

$$J_{\mathcal{Z}}(\theta_i) = \omega_i \, \mathbb{E}\left[\frac{(y_z - Q_{\theta_i}(x,a))^2}{2\sigma^2_{\theta_i}(x,a)} + \log(\sqrt{2\pi}\sigma_{\theta_i}(x,a))\right], \tag{5.25}$$

where

$$y_z = r + \gamma(Z(x',a') - \alpha \log \pi_{\bar{\phi}}(a'|x')) \tag{5.26}$$

is the target value for the return, with $Z(x',a') \sim \mathcal{Z}_{\bar{\theta}_i}(x',a')$ being x', a' the next state and chosen action, respectively, 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 $J_{\mathcal{Z}}(\theta_i)$ when $\sigma_{\theta_i}(x,a) \to 0$. This scaling parameter is dynamically updated according to the following rule

$$\omega_i \leftarrow \eta \, \mathbb{E}\Big[\sigma^2_{\theta_i}(x,a)\Big] + (1-\eta)\omega_i, \tag{5.27}$$

where η is the update rate.

The actor update rule is the maximization of

$$J_\pi(\phi) = \mathbb{E}\Big[\min_{i=1,2} Q_{\theta_i}(x,a) - \alpha \log(\pi_\phi(a|x)) - \lambda\sigma_{\theta_i}(x,a)\Big], \tag{5.28}$$

where the term $\lambda\sigma_{\theta_i}(x,a)$ has the purpose of diminishing the randomness of the return. The critic and actor updates follow Stochastic Gradient Descent (SGD) and Stochastic Gradient Ascent (SGA) with learning rates β_Z and β_π, respectively. The algorithmic description of the DSAC-T training process is reported in Alg. 1 [26].

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. (5.7), 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 $\boldsymbol{o}(t)$,

Algorithm 1 DSAC-T [26]

Require: $\theta_1, \theta_2, \phi, \alpha, \beta_Z, \beta_\pi, \beta_\alpha, \eta$
Initialize target networks:
  $\bar{\theta}_1 \leftarrow \theta_1$
  $\bar{\theta}_2 \leftarrow \theta_2$
  $\bar{\phi} \leftarrow \phi$
for each iteration **do**
  for each sampling step **do**
    Calculate action $a \sim \pi_\phi(a|s)$
    Get reward r and new state x'
    Store sample (x, a, r, x') in buffer $\mathcal{B}$
  end for
  for each update step **do**
    Sample data from $\mathcal{B}$
    Update critic: $\theta \leftarrow \theta - \beta_Z \nabla_\theta J_Z(\theta)$
    Update actor: $\phi \leftarrow \phi + \beta_\pi \nabla_\phi J_\pi(\phi)$
    Update temperature using Eq. (5.23)
    Update target networks:
  $\bar{\theta}_i \leftarrow \eta\theta_i + (1-\eta)\bar{\theta}_i$
  $\bar{\phi} \leftarrow \eta\phi + (1-\eta)\bar{\phi}$
  end for
end for

remote resources 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. The reward at each time step is computed according to Eq. (5.8). Importantly, queue overflow and execution deadline constraints in (5.12)–(5.11) are handled through a penalty mechanism in the reward function. Specifically, whenever a task violates its execution deadline or is dropped due to queue overflow, a negative reward term r_{neg} is applied. 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 space-related constraints are satisfied are reported in Sec. 5.5.1 instead, where training details are specified. A diagram illustrating our solution approach is depicted in Fig. 5.3.

5.5 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.

5.5.1 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

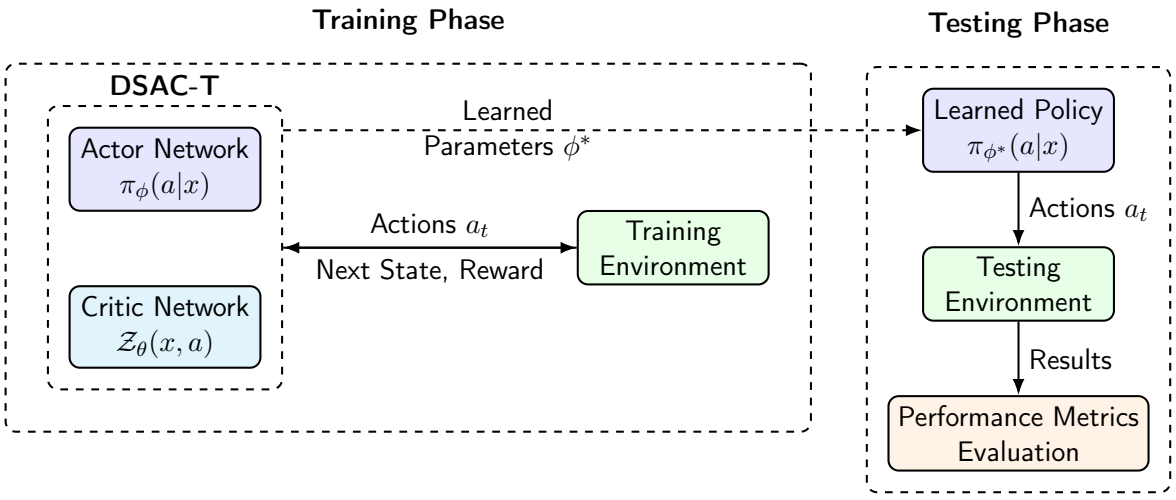


Figure 5.3: Workflow diagram: During the training phase, the DSAC-T algorithm interacts with the environment and updates its parameters for learning the value distribution while optimizing the policy loss function. After the training phase, the learned policy is tested with the environment for assessing its performance.

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 [58]. 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 $[10, 50]$ ms, the Poisson parameter λ_n for the generation of the tasks from the interval $[0.01, 0.1]$ tasks/slot and the task data size d_n from $[10000, 100000]$ Mbit, 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 $[200, 270]$ 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 $[0.01, 0.1]$, 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 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

Table 5.2: Values of hyperparameters, unless otherwise stated

Hyperparameter	Value
β_Z	0.0001
β_π	0.0001
β_α	0.0003
γ	0.99
η	0.005
r_{neg}	-100
λ	0

approach described in Sec. 5.3.2. The maximum capacity of the queues $Q_{\max}$ was instead fixed to 10.

The hyperparameters of the DSAC-T algorithm are listed in Tab. 5.2, 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.

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 [63]. 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. (5.13)–(5.14), we adopt a proportional normalization mechanism. Specifically, let $\rho(t) = [\rho_1(t), \ldots, \rho_{N_u}(t)]$ denote the raw actor outputs for computational allocation. The executed allocation is obtained as

$$\tilde{\rho}_n(t) = \frac{\rho_n(t)}{\sum_{k=1}^{N_u} \rho_k(t)}, \tag{5.29}$$

which guarantees by construction that constraint in Eq. 5.14 is satisfied. This transformation preserves the relative proportions among users decided by the policy, while strictly enforcing resource feasibility at every time step. 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 $\boldsymbol{\zeta} = [\zeta_1, \ldots, \zeta_{N_u}]$ 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.

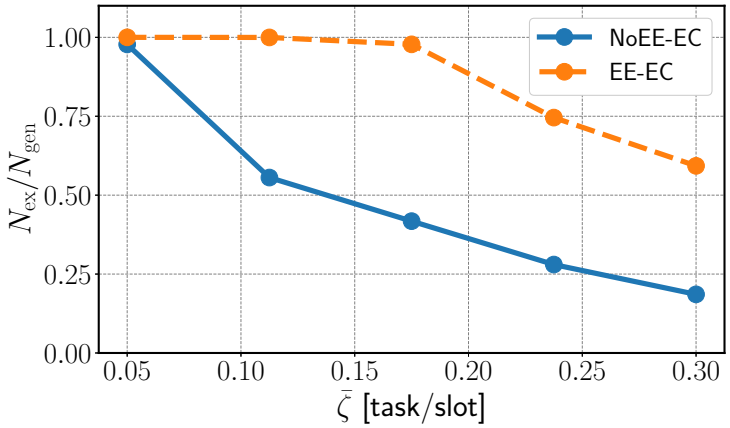
We first show the benefits of relying on EE with respect to a classic EC system which does not rely on EE. In particular, we indicate with EE-EC the system relying on EE, while with NoEE-EC the classic EC system. We then show the impact of the chosen reward function and the penalties on the learned policy, using a classic SAC algorithm as performance benchmark. The SAC algorithm shares a similar network architecture with DSAC-T, however it does not learn the full probability distribution of the return as it only focuses on learning the expected return.

5.5.2 EE vs NoEE

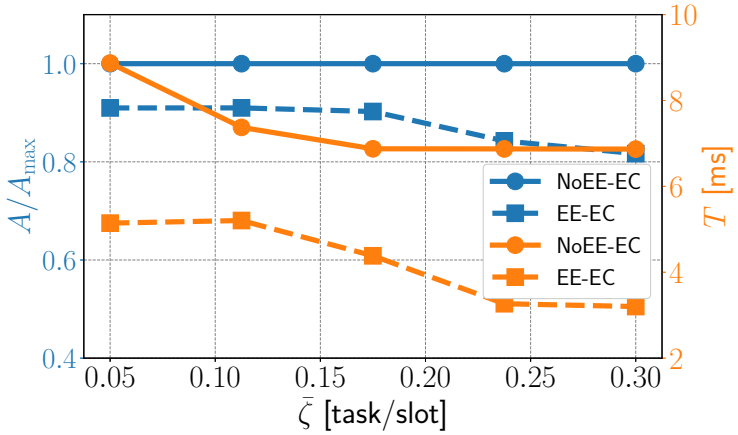
In Fig. 5.4a we compare the average completion ratio of the 3 users as a function of the task generation parameter $\boldsymbol{\zeta}$. The horizontal axis represents the average of the Poisson parameters of the 3 users, denoted by $\bar{\zeta}$. Results show that as $\bar{\zeta}$ increases, the completion 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. 5.4b. 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.

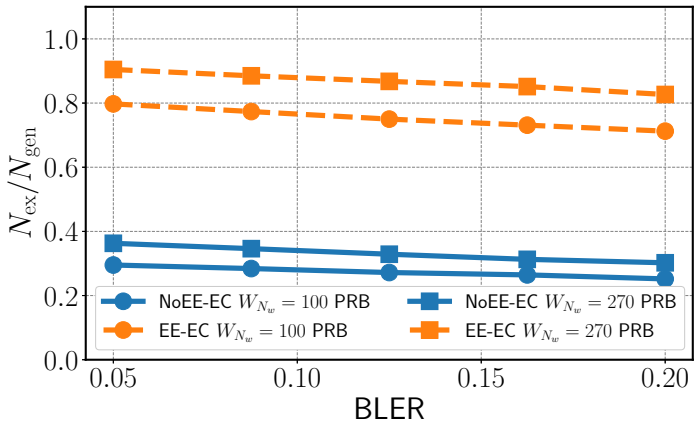
In Fig. 5.4c, we compare the performance of the two systems in terms of completion ratio as a function of the target BLER. The completion ratio is further evaluated under two different radio resource availability conditions, specifically for $W_{N_w} \in 100, 270$ PRB. As expected, the completion ratio 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 configurations. Furthermore, the results indicate that the **EE-EC system exhibits greater sensitivity to increased radio**



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

Figure 5.4: Performance comparison between NoEE-EC and EE-EC architectures.

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 =

Table 5.3: 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

0.1, we evaluate the induced average rate error when the target BLER differs from the reference value. Tab. 5.3 reports the average relative rate error for different BLER targets and slope parameters β. The detailed derivation, and a deeper discussion, are reported in Sec. 5.6. 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.

5.5.3 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 choose the classic SAC algorithm as a baseline to compare its performance with its distributional variant. The values of the hyperparameters used for training the SAC were the same we used for the DSAC-T, where possible.

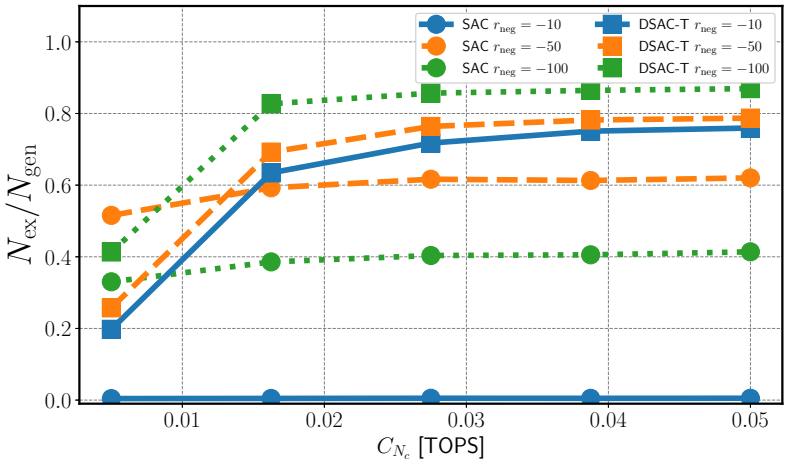
Penalty

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 choose the classic SAC algorithm as a baseline to compare its performance with its distributional variant. The values of the hyperparameters used for training the SAC were the same we used for the DSAC-T, wherever possible.

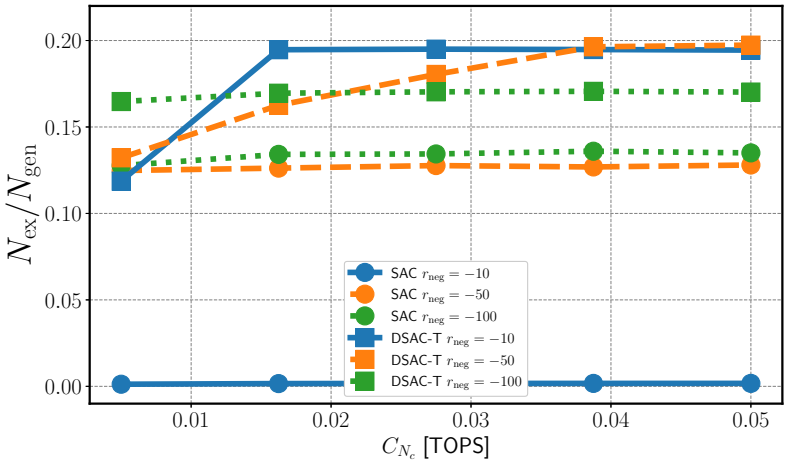
Penalty

In Fig. 5.5a 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. In general, **When taking r_{neg} constant, the DSAC-T performs better than the classic SAC.** Notably, the highest completion ratio is achieved by the policy trained with $r_{\text{neg}} = -100$ when using the DSAC-T; conversely, the lowest one corresponds to the policy trained with $r_{\text{neg}} = -10$ with both 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.



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



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

Figure 5.5: Penalties effect.

However, when using the DSAC-T algorithm, overly severe penalties may lead to suboptimal policies, as illustrated in Fig. 5.5b, which presents the same performance comparison as Fig. 5.5a, 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. 5.5a, achieving the lowest performance with $r_{\text{neg}} = -10$, and the highest one with $r_{\text{neg}} = -50$.

Weights

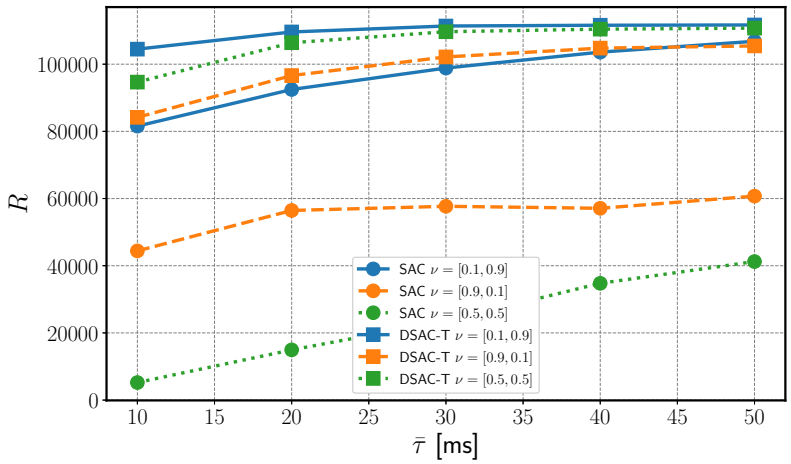
The weights of the reward function shown in Eq. (5.8) have to be chosen according to the desired system requirements. In Fig. 5.6a we report the cumulative reward as a function of the average tasks deadlines $\bar{\tau}$ for three different reward weights configurations, namely $\boldsymbol{\nu} = [0.9, 0.1]$, $\boldsymbol{\nu} = [0.5, 0.5]$ and $\boldsymbol{\nu} = [0.1, 0.9]$. 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 algorithms results show that the models trained with $\boldsymbol{\nu} = [0.1, 0.9]$ achieve the largest cumulative rewards, whereas the models trained with $\boldsymbol{\nu} = [0.9, 0.1]$ achieves worst performance. Insights into the cumulative rewards shown in Fig 5.6a can be better understood by examining the corresponding average completion ratios reported in Fig 5.6b. The results indicate that models trained to prioritize execution time tend to achieve larger completion ratios. Specifically, the model trained with $\boldsymbol{\nu} = [0.1, 0.9]$, which places greater emphasis on minimizing execution time, achieves the largest completion ratio. In contrast, the model trained with $\boldsymbol{\nu} = [0.9, 0.1]$ prioritizes higher precision, leading to longer execution times and a smaller completion ratio. Interestingly, the differences in performance between the two algorithms are evident for $\boldsymbol{\nu} = [0.9, 0.1]$ and $\boldsymbol{\nu} = [0.5, 0.5]$, whereas they achieve similar performance with $\boldsymbol{\nu} = [0.1, 0.9]$.

5.5.4 Risk-Sensitive Policy

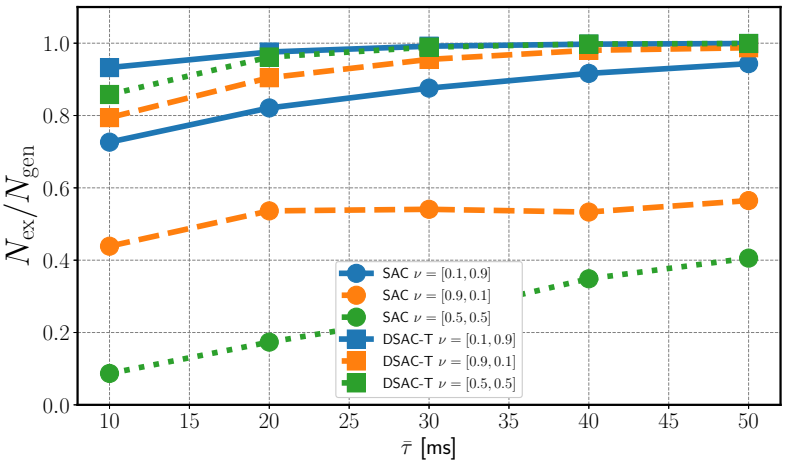
In Fig. 5.7 we illustrate the benefits of employing a distributional RL approach. In particular, we compare multiple policies trained with different values of the standard deviation weighting factor λ, evaluating their average completion ratio as a function of the available remote computational resources 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.5.4c, the completion ratio generally decreases with increasing BLER. However, it can now be observed from Fig. 5.7 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



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

Figure 5.6: Performance comparison between DSAC-T and SAC: Reward weights effect.

$\lambda = 5$ 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.

These findings highlight the value of incorporating reward standard deviation into the training objective of RL algorithms. By tuning λ, it is possible to develop policies that strike a balance between performance and reliability, adapting to different system conditions and failure risks.

The mismatch reported in Tab. 5.3 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 behavior of the learned policy under extreme network unreliability, rather than to provide quantitative performance claims.

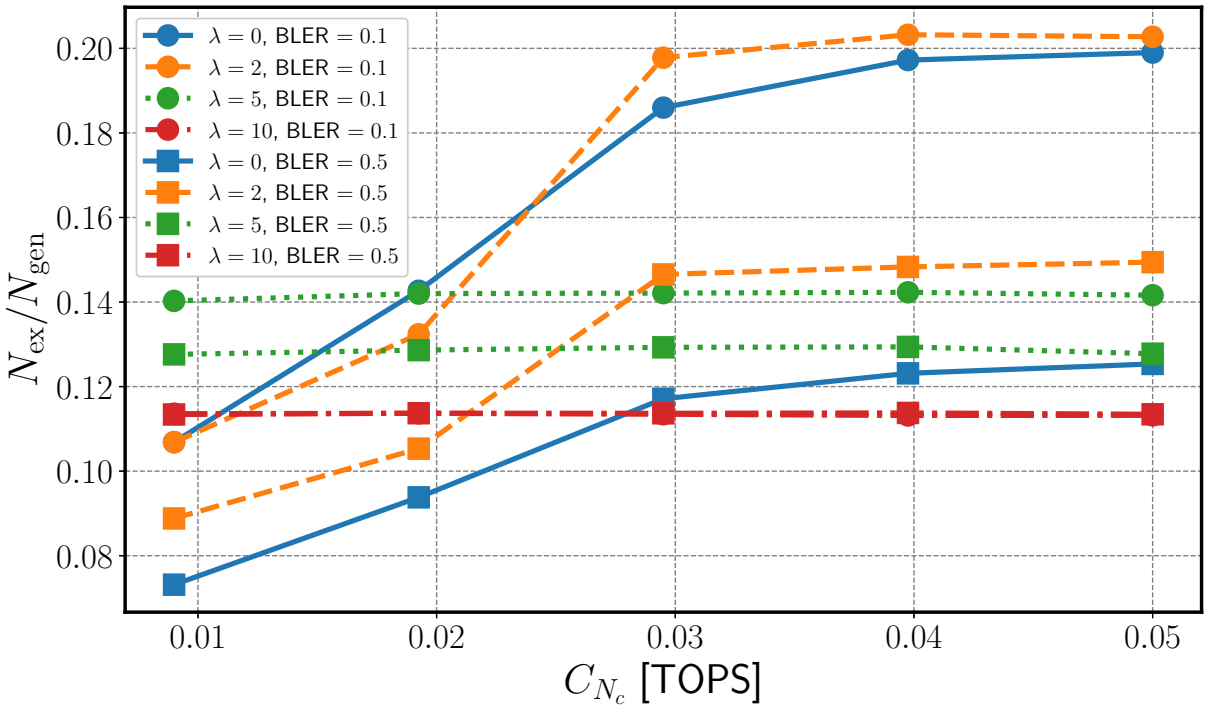


Figure 5.7: Risk-sensitivity: Average completion ratio versus C_{N_c}. $\boldsymbol{\zeta} = [0.1, 0.1, 0.1]$ task/slot, $\boldsymbol{\tau} = [30, 30, 30]$ ms, $\boldsymbol{\delta} = [1, 1, 1]$e4 Mbit, $\boldsymbol{\epsilon} = [1, 1, 1]$, $C_n = 0.001$ TOPS, $W_{N_w} = 50$ PRB, $f_D L = 0.65$.

5.5.5 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. 5.8a 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. 5.8b 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. (5.21) 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. 5.8c 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. 5.8d 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. (5.28). 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. 5.8e 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. 5.8f 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. 5.8g. 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. 5.8h 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.

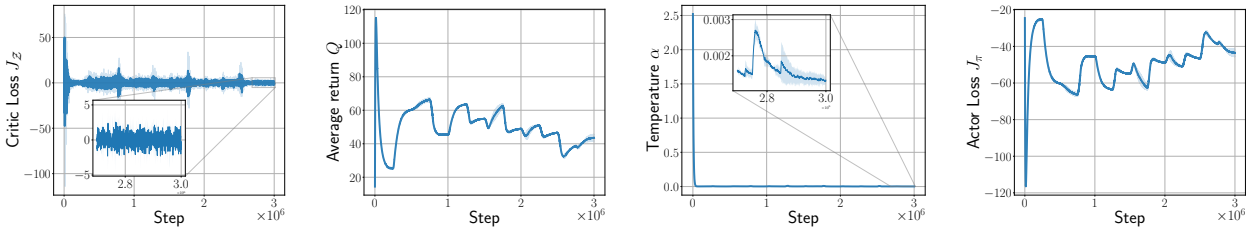
5.6 SNR-MCS Mapping

Using the same SNR-MCS mapping for BLER values different from a reference BLER value BLER_{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 BLER_{ref} to a generic target BLER_{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

$$\text{BLER}_m(\gamma) = \frac{1}{1 + \exp(\beta(\gamma - \gamma_m))}, \tag{5.30}$$



(a) $J_{\mathcal{Z}}$ versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\alpha = 0.2$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(b) Q versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\alpha = 0.2$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(c) α versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\alpha = 0.2$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(d) J_π versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\alpha = 0.2$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(e) σ versus training steps. $\gamma = 0.99$, $\lambda = 0$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\alpha = 0.2$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.1, 0.9]$.

(f) $J_{\mathcal{Z}}$ versus training steps and λ. $\gamma = 0.99$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\alpha = 0.2$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.5, 0.5]$.

(g) Q versus training steps and λ. $\gamma = 0.99$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\alpha = 0.2$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.5, 0.5]$.

(h) σ versus training steps and λ. $\gamma = 0.99$, $Q_{\text{lr}} = 0.0001$, $\pi_{\text{lr}} = 0.0001$, $\eta = 0.005$, $\alpha = 0.2$, $\beta_\alpha = 0.0003$, $\boldsymbol{\nu} = [0.5, 0.5]$.

Figure 5.8: Training performance.

where γ denotes 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. (5.30), the SNR required to achieve a generic target BLER for MCS m is

$$\gamma = \gamma_m + \frac{1}{\beta} \ln \left(\frac{1 - \text{BLER}}{\text{BLER}} \right). \tag{5.31}$$

Evaluating this expression at BLER_{ref} and at a different target BLER_{tgt}, the corresponding SNR thresholds satisfy

$$\gamma_m^{(\text{tgt})} = \gamma_m^{(\text{ref})} + \Delta\gamma, \tag{5.32}$$

where

$$\Delta\gamma = \frac{1}{\beta} \ln \left(\frac{\text{BLER}_{\text{ref}}(1 - \text{BLER}_{\text{tgt}})}{\text{BLER}_{\text{tgt}}(1 - \text{BLER}_{\text{ref}})} \right). \tag{5.33}$$

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 BLER_{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 BLER_{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 [64] 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

$$p_{\text{mis}|r_i} = \mathbb{P}\left(\gamma + \Delta\gamma \notin [\gamma_i, \gamma_{i+1}]\right). \tag{5.34}$$

Under the uniform assumption, this probability reduces to

$$p_{\text{mis}|r_i} = \min\left(1,\ \frac{|\Delta\gamma|}{\gamma_{i+1} - \gamma_i}\right). \tag{5.35}$$

The average relative rate error is then computed as

$$\bar{\Delta R} = \sum_{i=1}^{N_r} \pi_i \, p_{\text{mis}|r_i} \, \Delta R_i, \tag{5.36}$$

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. (5.3) is the spectral efficiency, the relative error reduces to the ratio between the corresponding spectral efficiencies, taken from the reference mapping in [64], and it can be defined as follows:

$$\Delta R_i = \frac{R_i^{(\text{tgt})} - R_i^{(\text{ref})}}{R_i^{(\text{ref})}} = \frac{Q_i^{(\text{tgt})} - Q_i^{(\text{ref})}}{Q_i^{(\text{ref})}}, \tag{5.37}$$

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

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

$$\bar{\gamma}_i^{\text{shift}} = \bar{\gamma}_i - \Delta\gamma. \tag{5.38}$$

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

$$\tau_{m-1} \leq \bar{\gamma}_i < \tau_m. \tag{5.39}$$

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

$$\bar{\gamma}_i^{\text{shift}} < \tau_{m^\star}. \tag{5.40}$$

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.

Table 5.4: 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

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. 5.36. Tab. 5.3 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 Tab. 5.3 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.

Tab. 5.4 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:

$$P_{\text{tx\_err}} = \sum_{i=1}^{N_r} \pi_i \, p_{\text{mis}|r_i} \, \mathbf{1}_{\left\{(R_i^{(\text{tgt})}>0) \neq (R_i^{(\text{ref})}>0)\right\}}, \tag{5.41}$$

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.

5.7 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 performance of up to 212% with respect to classic edge computing systems. 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.

Chapter 6

Conclusions

In this thesis, we investigated a system architecture aimed at meeting the stringent latency requirements of future Intelligent Transportation Systems. The proposed solution integrates edge computing with Early Exiting mechanisms for adaptive inference, enabling a dynamic trade-off between accuracy and computational effort under varying network and processing conditions.

Through a detailed system modeling effort, we demonstrated the benefits of combining edge offloading and adaptive inference. Our first contribution focused on a simplified scenario with a single CAV and an edge server. We showed that enhancing conventional edge computing with Early Exiting allows the system to effectively adapt inference latency to the current load, task generation rate, and wireless channel quality. Numerical results confirmed that this integration provides increased flexibility and better compliance with latency constraints compared to classical offloading architectures.

Our second contribution extended the analysis to a multi-user setting, where radio and computational resources also become part of the control problem. To address the resulting high-dimensional decision-making problem, we adopted a DistRL approach capable of jointly learning optimal offloading, resource allocation, and exit-selection policies. DistRL further enabled quantifying uncertainty in the learned value distributions, offering a richer representation of system behavior. The obtained results highlight how early-exit decisions can be meaningfully combined with resource allocation policies, revealing complex interactions between inference accuracy, latency, and available resources. Additionally, the learned control policy exhibited risk-aware behavior: in adverse conditions, such as severely degraded wireless channels, the agent prioritizes safer execution strategies by reducing average performance in favor of lower return variance.

Overall, this thesis demonstrates that coupling edge computing with adaptive inference and advanced decision-making techniques such as DistRL provides a promising foundation for the development of low-latency, reliable ITS services in next-generation vehicular networks.

Future Work

The research presented in this thesis opens several directions for future investigation.

A first natural extension concerns the integration of more realistic wireless and mobility models. While the proposed FSMC-based representation captures temporal channel correlation, incorporating spatial dynamics or multi-link connectivity would provide a more comprehensive evaluation

under realistic vehicular scenarios.

Another promising line of research is the adoption of multi-agent Reinforcement Learning (MARL) approaches. In dense ITS environments, vehicles may need to coordinate their decisions on resource usage, offloading, and inference, especially under shared wireless and computing constraints. MARL could enable cooperative or competitive strategies, potentially improving system-wide performance while maintaining fairness across users.

From a learning perspective, several extensions to DistRL can be explored. These include alternative return parameterizations beyond Gaussian distributions and more refined risk-sensitive objectives based on coherent risk measures (e.g., CVaR). Additionally, the development of theoretical guarantees for convergence and stability of DistRL in high-dimensional control problems remains an open and compelling question.

Finally, validating the proposed framework on real-world datasets or hardware testbeds would represent a crucial step toward practical deployment. Implementing adaptive inference pipelines on edge facilities and evaluating end-to-end latency under real vehicular traffic traces are all key aspects for translating the proposed solutions into operational ITS systems.

Overall, these research directions highlight the potential for significantly advancing the reliability, robustness, and efficiency of next-generation connected vehicle ecosystems, building on the foundations presented by the present work.

Bibliography

[1] 3GPP. Study on LTE support for Vehicle-to-Everything (V2X) services. Technical Report (TR) 22.885, 3rd Generation Partnership Project (3GPP), 2015. V.14.0.0.

[2] 3GPP. Study on enhancement of 3GPP support for 5G V2X services. Technical Report (TR) 22.886, 3rd Generation Partnership Project (3GPP), 2018. V.16.2.0.

[3] 3GPP. Service requirements for enhanced V2X scenarios. Technical Specification (TS) 22.186, 3rd Generation Partnership Project (3GPP), 2025. V.19.0.0.

[4] 3rd Generation Partnership Project (3GPP). User Equipment (UE) radio access capabilities. Technical Specification (TS) 38.306, 2021. Version 16.3.0.

[5] 5G Automotive Association (5GAA). C-V2X Use Cases and Service Level Requirements Volume I. White paper, 2019.

[6] 5G Automotive Association (5GAA). C-V2X Roadmap – Press Briefing. Online presentation, Sept. 2020. Available online.

[7] 5G Automotive Association (5GAA). C-V2X Use Cases and Service Level Requirements Volume II. White paper, 2020.

[8] 5G Automotive Association (5GAA). C-V2X Use Cases and Service Level Requirements Volume III. White paper, 2025.

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

[10] S. Angelucci, R. Valentini, M. Levorato, F. Santucci, and C. F. Chiasserini. Edge Computing with Early Exiting for Adaptive Inference in Mobile Autonomous Systems. In *Proc. of IEEE ICC*, Denver, Colorado, USA, 2024.

[11] S. Angelucci, R. Valentini, M. Levorato, F. Santucci, and C. F. Chiasserini. Distributional rl for task offloading, resource allocation and early exit selection at the edge. Nov. 2025.

[12] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proc. of PMLR International conference on machine learning*, Sydney, Australia, 2017.

[13] M. G. Bellemare, W. Dabney, and M. Rowland. *Distributional Reinforcement Learning*. MIT Press, 2023. `http://www.distributional-rl.org`.

[14] A. Ben-Ameur, A. Araldo, and T. Chahed. Multiple Resource Allocation in Multi-Tenant Edge Computing via Sub-Modular Optimization. In *Proc. of IEEE ICC*, Rome, Italy, 2023.

[15] S. Birhanu Engidayehu, T. Mahboob, and 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.

[16] D. Burgstahler, C. Peusens, D. Boehnstedt, and R. Steinmetz. Horizon.KOM: A first step towards an open vehicular horizon provider. In *Proceedings of the International Conference on Vehicle Technology and Intelligent Transport Systems - Volume 1: VEHITS,*, pages 79–84. INSTICC, SciTePress, 2016.

[17] D. Burgstahler, A. Xhoga, C. Peusens, M. Moebus, D. Boehnstedt, and R. Steinmetz. Remote-horizon.kom: Dynamic cloud-based ehorizon. In *AmE 2016 - Automotive meets Electronics; 7th GMM-Symposium*, pages 1–6, 2016.

[18] D. Callegaro and M. Levorato. Optimal Edge Computing for Infrastructure-Assisted UAV Systems. *IEEE Transactions on Vehicular Technology*, 70(2):1782–1792, 2021.

[19] D. Callegaro, Y. Matsubara, and M. Levorato. Optimal Task Allocation for Time-Varying Edge Computing Systems with Split DNNs. In *IEEE GLOBECOM*, 2020.

[20] R. Carruthers and N. C. Beaulieu. On an improved markov chain model of the rayleigh fading channel. In *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, pages 1529–1534, 2007.

[21] Z. Cheng, M. Min, Z. Gao, and L. Huang. Joint Task Offloading and Resource Allocation for Mobile Edge Computing in Ultra-Dense Network. In *Proc. of IEEE GLOBECOM*, Taipei, Taiwan, 2020.

[22] N. Cordoș, I. Duma, D. Moldovanu, A. Todoruț, and I. Barabás. An overview of intelligent transportation systems in europe. *World Electric Vehicle Journal*, 16(7), 2025.

[23] G. Di Sciullo, L. Zitella, E. Cinque, F. Santucci, M. Pratesi, and F. Valentini. Experimental Validation of C-V2X Mode 4 Sidelink PC5 Interface for Vehicular Communications. In *2022 61st FITCE International Congress Future Telecommunications: Infrastructure and Sustainability (FITCE)*, 2022.

[24] F. Dong, H. Wang, D. Shen, Z. Huang, Q. He, J. Zhang, L. Wen, and T. Zhang. Multi-exit dnn inference acceleration based on multi-dimensional optimization for edge intelligence. *IEEE Transactions on Mobile Computing*, 22(9):5389–5405, 2023.

[25] J. Duan, Y. Guan, S. E. Li, Y. Ren, Q. Sun, and B. Cheng. Distributional Soft Actor-Critic: Off-Policy Reinforcement Learning for Addressing Value Estimation Errors. *IEEE Trans. on Neural Networks and Learning Systems*, 33(11):6584–6598, 2022.

[26] J. Duan, W. Wang, L. Xiao, J. Gao, S. E. Li, C. Liu, Y.-Q. Zhang, B. Cheng, and K. Li. Distributional Soft Actor-Critic With Three Refinements. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 47(5):3935–3946, 2025.

[27] ETSI. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM); Rationale for and guidance on standardization . TR 102 863, European Telecommunication Standards Institute (ETSI), 2011. V.1.1.1.

[28] J. Fan, Q. Yin, G. Y. Li, B. Peng, and X. Zhu. MCS Selection for Throughput Improvement in Downlink LTE Systems. In *ICCCN*, 2011.

[29] W. Fan, Z. Chen, Z. Hao, F. Wu, and Y. Liu. Joint Task Offloading and Resource Allocation for Quality-Aware Edge-Assisted Machine Learning Task Inference. *IEEE Trans. on Vehicular Technology*, 72(5):6739–6752, 2023.

[30] J. Feng, Q. Pei, F. R. Yu, X. Chu, J. Du, and L. Zhu. Dynamic Network Slicing and Resource Allocation in Mobile Edge Computing Systems. *IEEE Trans. on Vehicular Technology*, 69(7):7863–7878, 2020.

[31] M. García, I. Urbieta, M. Nieto, J. Mendibil, and O. Otaegui. iLDM: An interoperable graph-based local dynamic map. *Vehicles*, 2022.

[32] H. Gauttam, G. Nain, K. Pattanaik, and P. Mendes. Edge-ai: A systematic review on architectures, applications, and challenges. *Journal of Network and Computer Applications*, 245:104375, 2026.

[33] A. Goldsmith and S.-G. Chua. Adaptive coded modulation for fading channels. *IEEE Trans. on Communications*, 46(5):595–602, 1998.

[34] J. Guan, Q. Zhang, I. Murturi, P. K. Donta, S. Dustdar, and S. Wang. Collaborative inference in dnn-based satellite systems with dynamic task streams. In *ICC 2024 - IEEE International Conference on Communications*, pages 3803–3808, 2024.

[35] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, 2018.

[36] M. Hao, D. Ye, S. Wang, B. Tan, and R. Yu. URLLC Resource Slicing and Scheduling in 5G Vehicular Edge Computing. In *IEEE VTC2021-Spring*, [Online], 2021.

[37] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-Scale Dense Networks for Resource Efficient Image Classification, 2018.

[38] F. Ilhan, K.-H. Chow, S. Hu, T. Huang, S. Tekin, W. Wei, Y. Wu, M. Lee, R. Kompella, H. Latapie, G. Liu, and L. Liu. Adaptive deep neural network inference optimization with eenet, 2023.

[39] J. Jang, K. Tulkinbekov, and D.-H. Kim. Task Offloading of Deep Learning Services for Autonomous Driving in Mobile Edge Computing. *MDPI Electronics*, 12, 2023.

[40] P. Lai, Q. He, X. Xia, F. Chen, M. Abdelrazek, J. Grundy, J. Hosking, and Y. Yang. Dynamic User Allocation in Stochastic Mobile Edge Computing Systems. *IEEE Trans. on Services Computing*, 15(5):2699–2712, 2022.

[41] C. Li, J. Wang, K. Jiang, C. Xiong, and S. Wan. Meoci: Model partitioning and early-exit point selection joint optimization for collaborative inference in vehicular edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 37(3):666–679, 2026.

[42] L. Li, Q. Guan, L. Jin, and M. Guo. Resource Allocation and Task Offloading for Heterogeneous Real-Time Tasks With Uncertain Duration Time in a Fog Queueing System. *IEEE Access*, 7:9912–9925, 2019.

[43] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief. Delay-optimal computation task scheduling for mobile-edge computing systems. In *IEEE ISIT*, 2016.

[44] J. Liu, Y. Wang, W. Zhang, and 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.

[45] Q. Liu and T. Han. VirtualEdge: Multi-Domain Resource Orchestration and Virtualization in Cellular Edge Computing. In *Proc. of IEEE ICDCS*, Dallas, Texas, USA, 2019.

[46] Z. Liu, J. Song, C. Qiu, X. Wang, X. Chen, Q. He, and H. Sheng. Hastening stream offloading of inference via multi-exit dnns in mobile edge computing. *IEEE Transactions on Mobile Computing*, 23(1):535–548, 2024.

[47] B. Lv, C. Yang, X. Chen, Z. Yao, and J. Yang. Task Offloading and Serving Handover of Vehicular Edge Computing Networks Based on Trajectory Prediction. *IEEE Access*, 2021.

[48] Y. Matsubara, M. Levorato, and F. Restuccia. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges. *ACM Computing Surveys*, 55(5), 2022.

[49] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

[50] S. Meyn, R. L. Tweedie, and P. W. Glynn. *Markov Chains and Stochastic Stability*. Cambridge Mathematical Library. Cambridge University Press, 2 edition, 2009.

[51] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. 2013.

[52] A. Molisch. *Wireless Communications*. John Wiley & Sons Inc., United States, 2005.

[53] T. Niu, Y. Teng, Z. Han, and P. Zou. An adaptive device-edge co-inference framework based on soft actor-critic. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 2571–2576, 2022.

[54] M. A. Pinsky and S. Karlin. In *An Introduction to Stochastic Modeling (Fourth Edition)*. Academic Press.

[55] C. Ress, A. Etemad, D. Kuck, and J. Requejo. Electronic horizon - providing digital map data for adas applications. In *Proceedings of the 2nd International Workshop on Intelligent Vehicle Control Systems - Volume 1: IVCS, (ICINCO 2008)*, pages 40–49. INSTICC, SciTePress, 2008.

[56] K. W. Ross. Randomized and Past-Dependent Policies for Markov Decision Processes with Multiple Constraints. *Oper. Res.*, 1989.

[57] J. G. Ruiz, B. Soret, M. C. Aguayo-Torres, and J. T. Entrambasaguas. On finite state Markov chains for Rayleigh channel modeling. In *2009 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology*, 2009.

[58] C. Singhal, Y. Wu, F. Malandrino, M. Levorato, and C. F. Chiasserini. Resource-aware Deployment of Dynamic DNNs over Multi-tiered Interconnected Systems. In *Proc. of IEEE INFOCOM*, Vancouver, Canada, 2024.

[59] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[60] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[61] C. Tan and N. Beaulieu. On first-order Markov modeling for the Rayleigh fading channel. *IEEE Trans. on Communications*, 48(12):2032–2040, 2000.

[62] S. Teerapittayanon, B. McDanel, and H. T. Kung. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks, 2017.

[63] M. Theile, L. Dirnberger, R. Trumpp, M. Caccamo, and A. L. Sangiovanni-Vincentelli. Action Mapping for Reinforcement Learning in Continuous Environments with Constraints, 2024.

[64] A. K. Thyagarajan, P. Balasubramanian, V. D, and K. M. SNR-CQI Mapping for 5G Downlink Network. In *Proc. of IEEE APWiMob*, Bandung, Indonesia, 2021.

[65] R. Valentini, N. Dang, M. Levorato, and E. Bozorgzadeh. Modeling and control battery aging in energy harvesting systems. In *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 515–520, 2015.

[66] R. Valentini and M. Levorato. Optimal aging-aware channel access control for wireless networks with energy harvesting. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 2754–2758, 2016.

[67] R. Valentini, M. Levorato, and F. Santucci. Aging aware random channel access for battery-powered wireless networks. *IEEE Wireless Communications Letters*, 5(2):176–179, 2016.

[68] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie. Efficient Mobility-Aware Task Offloading for Vehicular Edge Computing Networks. *IEEE Access*, 2019.

[69] J. Zhang, H. Guo, J. Liu, and Y. Zhang. Task Offloading in Vehicular Edge Computing Networks: A Load-Balancing Solution. *IEEE Trans. on Veh. Tech.*, 2020.

[70] Q. Zhang, X. Che, Y. Chen, X. Ma, M. Xu, S. Dustdar, X. Liu, and S. Wang. A Comprehensive Deep Learning Library Benchmark and Optimal Library Selection. *IEEE Trans. on Mob. Comp.*, 2023.

[71] W. Zhang and K. Tuo. Research on Offloading Strategy for Mobile Edge Computing Based on Improved Grey Wolf Optimization Algorithm. *MDPI Electronics*, 12(11), 2023.

[72] X. Zhang, Y. Teng, N. Wang, B. Sun, and G. Hu. Accelerating Deep Neural Network Tasks Through Edge-Device Adaptive Inference. In *Proc. of IEEE PIMRC*, Toronto, Canada, 2023.