UNIVERSITY OF CALIFORNIA,
IRVINE

Semantic Mobile Edge Computing Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Networked Systems

by

Sharon Ladron de Guevara Contreras

Dissertation Committee:
Professor Marco Levorato, Chair
Associate Professor Faisal Nawab
Assistant Professor Sangeetha Abdu Jyothi

2025

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Page

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

My deepest gratitude is reserved for my advisor, Professor Marco Levorato. He met every stage of my research with a blend of patience and intellectual curiosity that consistently transformed challenges into moments of discovery. From him, I learned the essential discipline of research: to frame questions with clarity, to pursue ideas relentlessly, and to evaluate outcomes with rigor. He balanced affording me the autonomy to explore with providing incisive direction at crucial milestones. This mentorship was foundational, shaping not only this dissertation but my identity as a researcher.

I am sincerely thankful to my committee members, Professor Sangeetha Abdu Jyothi and Professor Faisal Nawab, for their keen and direct feedback. Their interventions at pivotal stages sharpened the technical core of this work and clarified its narrative, enabling me to better articulate its place within systems, networking, and hardware management.

My research horizons were significantly expanded through the mentorship and collaboration of Professor Francesco Restuccia. His generous guidance was instrumental in helping me contextualize my ideas within the broader scientific discourse and to engage with that community effectively.

The daily work of research was made vibrant by my labmates in the IASL group. The intellectual camaraderie we built—through spirited whiteboard debates, ambitious reading groups, and late-night collaborations to decode results—is something I will always cherish. In our lab, assumptions were challenged, and support was unconditional, creating a climate where even the most formidable problems felt within reach.

This journey was anchored by the love of my family. To my parents, America and Marcos, thank you for the unconditional belief that has been my foundation. To my brothers, Marco and Enrique, thank you for the encouragement and the crucial reminders to see beyond the immediate pressures. To my partner, Alberto, I offer my most profound thanks. His love, patience, and unwavering faith in me, especially when my own faltered, were the essential forces that saw me through. And to my friends, thank you for the joy and compassion that brought light and balance to this entire process.

This dissertation also rests upon the work of a vast community. It was made possible by the developers of open-source tools, the curators of public datasets, and the spirit of intellectual generosity that pervades our field. To all who contributed, seen and unseen, this work is a reflection of your efforts as well as my own.

Finally, I would like to acknowledge the following:

- Portions of Chapter 3 of this dissertation in a reprint of the material as it appears in IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (2023) https://doi.org/10.1109/WoWMoM57956.2023.00034, used with permission from IEEE. The co-authors listed in this publication are Marco Levorato.

Marco Levorato directed and supervised research that forms the basis for the dissertation.

- Portions of Chapter 4 of this dissertation in a reprint of the material as it appears in IEEE 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (2025) https://doi.org/10.1109/DCOSS-IoT65416.2025.00014, used with permission from IEEE. The co-authors listed in this publication are Foysal Haque Khandaker, Francesco Restuccia, and Marco Levorato. Marco Levorato directed and supervised research that forms the basis for the dissertation.

- Portions of Chapter 5 of this dissertation in a reprint of the material as it appears in IEEE Military Communications Conference (2025), in Press (an earlier version of portions of this chapter was accepted for unclassified paper at IEEE MILCOM 2025; the final version appears in the conference proceedings). The co-authors listed in this publication are Hyomin Choi, Fabien Racape, Subhramoy Mohanti, and Marco Levorato. Marco Levorato directed and supervised research that forms the basis for the dissertation.

The work in this dissertation was partially supported by the following grants:

- National Science Foundation (NSF) grant CCF 2140154, CNS-2134973, and ECCS-2229472, by the Air Force Office of Scientific Research under contract number FA9550-23-1-0261, and by the Office of Naval Research under award number N00014-23-1-2221.

- Intel Corporation and the NSF grant MLWiNS-2003237 and CCF-2140154.

VITA

Sharon Ladron de Guevara Contreras

EDUCATION

Doctor of Philosophy in Networked Systems **2025**
University of California, Irvine *Irvine, CA*

Master of Science in Electrical Engineering **2020**
University of Southern California *Los Angeles, CA*

Bachelor of Science in Telematics Engineering **2016**
Instituto Tecnologico Autonomo de Mexico *Mexico City, Mexico City*

Bachelor of Science in Computer Engineering **2015**
Instituto Tecnologico Autonomo de Mexico *Mexico City, Mexico City*

RESEARCH EXPERIENCE

Graduate Research Assistant **2020–2025**
University of California, Irvine *Irvine, California*

Graduate Research Intern **2021** and **2022**
IBM *Yorktown Heigths, NY*

Graduate Research Intern **2024**
Interdigital *Los Altos, CA*

TEACHING EXPERIENCE

Teaching Assistant **2022–2025**
University of California, Irvine *Irvine, CA*

Research Mentor **2023–2025**
University of California, Irvine *Irvine, CA*

CONFERENCE PUBLICATIONS

Context-Aware Heterogeneous Task Scheduling for Multi-Layered Systems **Dec 2023**
WoWMoM

SOAR: Semantic MU-MIMO Communications for Reliable Offloading of Computer Vision Tasks **Jun 2025**
DCOSS-IOT

CAMTEC: Context-aware Adaptive Inference for Machine Task Execution in Complex Environments **Oct 2025**
MILCOM

WORKSHOP PUBLICATIONS

Mixed Reality Visualization in Automotive SDR Networks **Sep 2021**
GNU Radio Conference

Heterogeneous Machine Learning Task Scheduler for Constrained Devices **April 2022**
CRA-WP

Towards Resilient Task Offloading using MU-MIMO **Oct 2022**
MLWiNS

SOFTWARE

DRL Multi-task Scheduler `https://tinyurl.com/context-aware-scheduler`
Heterogeneous Task Scheduler for Constrained Devices that supports DRL learning

Wireless Comm Control Algorithm `https://github.com/Restuccia-Group/SOAR`
Wireless Control Distributional DRL learning Control Algorithm

ABSTRACT OF THE DISSERTATION

Semantic Mobile Edge Computing Systems

By

Sharon Ladron de Guevara Contreras

Doctor of Philosophy in Networked Systems

University of California, Irvine, 2025

Professor Marco Levorato, Chair

Machine Learning (ML) tasks such as object detection and semantic segmentation are central to autonomous navigation and Internet of Things applications. These tasks must operate under constrained on-device compute and memory, limited energy budgets, and strict real-time requirements. Mobile Edge Computing Systems (MECS) alleviate these constraints by moving computation closer to data sources and distributing inference across a Mobile Edge Device (MED) and a nearby Mobile Edge Server (MES). The MED to MES wireless link, therefore, carries a continuous bidirectional exchange of data. Current inference pipelines emphasize system-level metrics such as latency, buffer occupancy, and workload balance. These approaches often overlook the semantics: task-relevant information and operational context that directly shape both performance and the quality of transmitted data.

This dissertation develops a semantics approach to MECS and organizes resource control into three logics: a Lower-Layered-Computing Control (computing) that allocates and schedules execution across processor units and across the MED and MES with awareness of contention and context switching; a Cross-Layered-Communication Control (communications) that configures the wireless link, including MU MIMO parameters and retransmission policy, to protect task performance under packet loss and time varying channels; and a Higher-Layered-Computing Control (computation) that selects inference modes and model complexity to

optimize task performance and resource efficiency, with safety as a primary objective in autonomous systems. Using task requirements and environmental context, the system decides what to compute, where to compute it, and how to adapt communication so that the most useful information is delivered within real-time constraints. The work targets three challenges common in dynamic edge environments. First, devices often run multiple inference task types, which introduces context switching overhead. Second, unreliable wireless links produce incomplete or corrupted inputs in both line-of-sight and non-line-of-sight conditions. Third, offloading choices should be guided by task-level performance metrics such as safety, rather than only throughput or latency.

The dissertation offers three contributions that enable semantics in MECS.

1. Lower-Layered Computing Control: An Adaptive Multi-Task Computing Allocation in MECS. The first framework reduces context switching delays that undermine real-time performance on resource-constrained devices. The method co-optimizes model selection, scheduling, and placement between edge devices and nearby servers using cues from scene and task context. By avoiding repeated loading of models onto limited CPU and GPU resources, and by matching model complexity to the importance of the performance requirements, the framework improves end-to-end latency and throughput while preserving task accuracy.

2. Cross-Layered Control: Resilient Wireless Communication in MECS. The second framework addresses wireless variability, which often dominates overall performance. Our framework, SOAR, uses distributional deep reinforcement learning along with context-aware neural networks to allocate Multi-User MIMO resources and to adapt to a task-specific performance related to the packet loss ratio for vision workloads. Rather than treating all packets as equal, the policy prioritizes information that carries higher task value. Evaluations in vehicular environments show that SOAR reduces resource usage and improves reliability when compared with fixed antenna baselines, while preserving application-level quality.

3. Higher-Layered Control: CAMTEC: Context-aware adaptive inference for machine task execution. The third framework fuses compute capacity, energy consumption, network state, and machine learning performance with environmental and task semantics to select inference modes and operating points dynamically. CAMTEC decides when to run models on the device, when to offload, and when to change fidelity, and it tunes operating parameters to meet performance and safety goals. Field experiments demonstrate that CAMTEC maintains reliable autonomous navigation under limited bandwidth and tight energy budgets.

A testbed supports these contributions and grounds them in realistic conditions. The platform spans heterogeneous device architectures and includes both WiFi 802.11 and 802.11ac links. The experiments cover indoor and outdoor settings with and without obstacles, and they include rover trajectory collection to measure responsiveness, channel reliability, and closed-loop effects on control. The testbed enables end-to-end perception pipelines that combine object detection with performance monitoring and control so that outcomes can be measured at the task level rather than only through packet or computational metrics.

Results show that semantics is an approach that enables adaptiveness and task-level efficiency in MECS. By aligning communication, computation, and control with the meaning of the data and the needs of the task, the proposed frameworks deliver stronger resilience to wireless disruptions, better support for concurrent inference, and more effective decision-making using task-level metrics. The dissertation presents methods and empirical evidence indicating that semantics-centered MECS can meet the demands of complex edge environments.

Chapter 1

Introduction

Machine learning tasks such as object detection and semantic segmentation are key to the functionality of complex applications, composed of connected modules (e.g., autonomous navigation). These tasks typically are required to meet real-time guarantees despite operating on devices with constrained computational resources and limited energy budgets. Mobile Edge Computing Systems (MECS) address these challenges by distributing inference workloads between a Mobile Edge Device (MED) and one or more Edge Servers (ESs) located nearby. This system architecture enables expansion of the MEDs hardware capabilities, and it introduces a wireless link that must support continuous, bilateral exchange of data.

The deployment of MECS that perform interference in real-world environments presents three main challenges that have not been considered as part of the MECS design before. First, wireless channel conditions fluctuate due to mobility, interference, and the propagation environment, which potentially reflects in incomplete or corrupted inputs and unpredictable latency. Second, in real-world environments, tasks are heterogeneous; when such a workload runs on a resource-constrained device, compute availability fluctuates and significant latency overheads arise, particularly from context-switching across processing units. Third,

on more complex tasks, when there is control involved, such as in an autonomous navigation pipeline, the MECS do not consider the impact of the computing allocation on the overall task performance.

Existing solutions typically employ system-level optimizations that allocate compute resources based on load to minimize average latency, manage buffer occupancy, or balance workloads. Although these strategies are effective in stable environments, they often decouple computation and communication as separate problems where aggregated metrics are optimized, overlooking the task-level metrics.

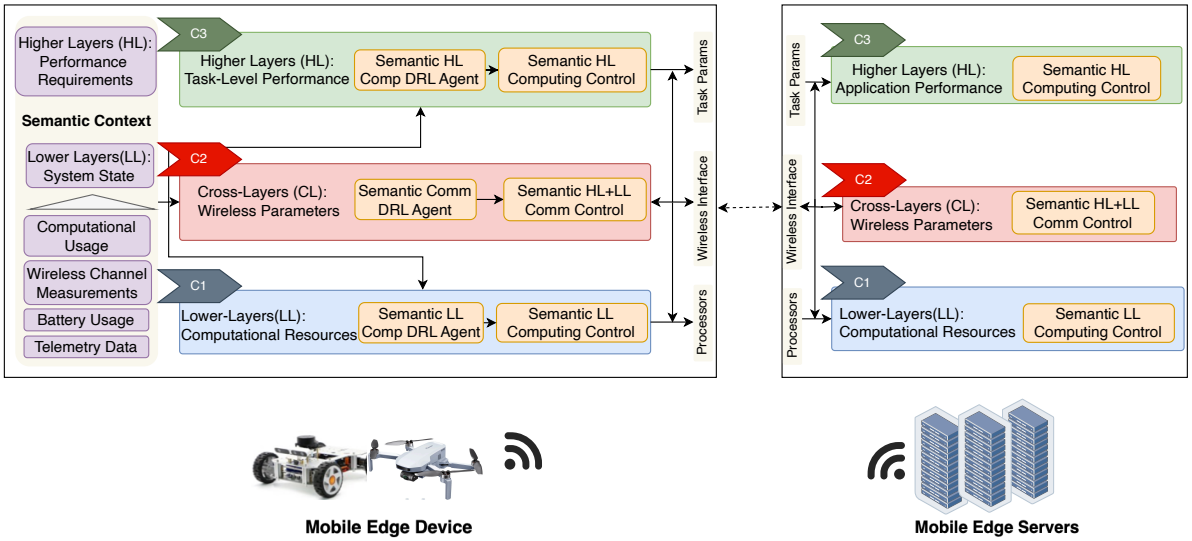


Figure 1.1: Semantic Modules in MECS

This dissertation develops a semantic approach to resource management in MECS. In this work, *semantics* refers to the systematic awareness of the computation and communication context relative to the resulting joint control of computing and communication resources that ultimately affect task performance as shown in Fig.1.1. On this basis, MECS can make intelligent decisions regarding what to compute, where to compute it, and how to adapt communication parameters to time-varying channels, ensuring the most useful information

is delivered within real-time constraints.

A novel three-tier hierarchical control architecture for MECS is proposed. This hierarchical control jointly manages computation and communication resources with explicit attention to task-level performance. The architecture decomposes the overall problem into three interdependent logical layers. Each layer is governed by a semantic Deep Reinforcement Learning (DRL) agent that observes continuous measurements of the system state—comprising compute and communication resources—and issues control actions accordingly, as depicted in Fig. 1.1.

Lower-Layered Computing Control. A semantic DRL agent allocates compute resources (CPUs and GPUs) across the MED and the Edge Server ES in response to heterogeneous task arrivals. The agent explicitly models context-switching latency, the overhead incurred when swapping ML models on a processor, and incorporates this cost into its optimization to meet per-task latency and performance objectives.

Cross-layered communications control. A second semantic DRL agent tunes wireless communication parameters to mitigate packet loss on unreliable MECS links. It links task-level performance to communication settings and packet-duplication strategies using observed packet-loss behavior and the propagation environment, optimizing configurations with respect to latency deadlines and energy expenditure.

Higher-Layered Computing Control. At the top layer, a semantic DRL agent selects inference modalities and other high-level parameters based on the computing and communication context. It optimizes the performance of complex tasks such as closed-loop applications that execute inference-driven pipelines by coordinating the use of computation, energy, and communication resources.

Control is distributed across MECS, with emphasis on MED given its constrained hardware resources. We validate each semantic module on dedicated testbeds featuring heterogeneous

hardware, Wi-Fi 802.11/802.11ac, a mobile rover on a fixed trajectory and a rover performing dynamic obstacle avoidance. Across all testbeds, we execute end-to-end perception pipelines that combine object detection, performance monitoring, and control, and we report task-level outcomes, channel reliability, and closed-loop effects directly connecting network and compute decisions to overall task performance.

The remainder of the dissertation is organized as follows.

- Chapter 2 reviews background on system-level optimization in MECS and the deep reinforcement learning methods used in this work.
- Chapter 3 develops the Lower-Layered Computing Control framework to minimize context-switching delay in multi-task inference; it details the dataset and a heterogeneous device testbed and reports results.
- Chapter 4 introduces the Cross-Layered Communications Control framework, which adapts wireless MU-MIMO parameters to the propagation environment and task-level objectives; it leverages the correlation between packet loss and task-level performance, details the dataset and a wireless testbed, and reports results.
- Chapter 5 presents the Higher-Layered Computing Control framework, which optimizes performance of complex, closed-loop tasks (e.g., autonomous navigation) under computational and communication constraints; it details the dataset and a rover testbed with obstacle avoidance and reports results.
- Chapter 6 concludes the dissertation.

Chapter 2

Literature Review

This chapter reviews two core areas: mobile edge computing systems (MECS), which support the offloading of machine learning tasks, and deep reinforcement learning (DRL), which enables online offloading and resource allocation. We emphasize the joint design of computation and communication to optimize the computational and energy resources of the mobile edge device (MED).

2.1 Mobile Edge Computing Systems

Mobile edge computing systems (MECS) place edge servers (ESs) close to mobile edge devices (MEDs) to augment local compute and memory while reducing task end to end latency and relieving backhaul load [50, 27]. MEDs operate under tight power, memory, and processing budgets, whereas machine learning models are compute and memory intensive; continuous execution of inference can drain batteries and miss real-time deadlines on embedded platforms [84, 76, 15]. These constraints motivate task offloading and model adaptation.

Task offloading strategies in MECS are typically defined by two dimensions: the portion of

the computational pipeline transferred and the location of execution. A system can offload an entire application or partition its computation graph, with execution occurring on the MED, an ES, in the cloud, or distributed across these tiers [83, 40, 69]. For machine learning inference, this leads to three primary execution models: local, full, and partial offloading. Local inference contains the entire model on the MED, while full offloading transmits raw sensor data to an ES for complete execution, a strategy often used to maximize accuracy for workloads like video analysis or speech recognition [32, 30, 3, 1]. Partial offloading, or split computing, partitions the model between the MED and ES, sending intermediate features over the wireless link. Techniques for partial offloading include layer or subgraph partitioning, using early-exit classifiers to stop inference when local confidence is high, and applying compression to reduce data volume [54, 55, 51, 52, 46, 65, 8]. The goal across these models is to dynamically balance accuracy, latency, and energy consumption as system conditions evolve.

To achieve this balance, research has extensively addressed the challenge of computational resource allocation through adaptive management strategies. At the task level, algorithms manage complex dependencies, as Shu et al. [73] proposed a dependency-aware, latency-optimal offloading scheduler. The scope of optimization frequently extends to other system resources; for instance, Tran et al. [80] jointly optimized service caching and task offloading to reduce both latency and operational costs. Many of these approaches employ multi-objective optimization to navigate trade-offs. Bi et al. [11] focused on minimizing MED energy consumption under strict delay constraints, while Lakew et al. [44] designed a workload balancing system for heterogeneous edge nodes. Enabling these dynamic policies at a system level requires sophisticated orchestration, often implemented with technologies like network function virtualization (NFV) and serverless computing to map tasks to physical resources [12, 5]. While computational optimization is essential, the performance of any MECS offloading policy is fundamentally governed by the quality of the wireless link. This critical dependence necessitates policies that co-adapt communication and computation; Callegaro

et al. [14], for example, developed a solution that adjusts the DNN split point in direct response to fluctuating wireless channel quality. The reliability of this link is a primary concern, as empirical studies show that channel noise can degrade the accuracy of offloaded tasks—an effect that can be mitigated with computational pre-processing such as denoising [39, 63]. Although modern standards like Wi-Fi and 5G New Radio provide features for low-latency, reliable communication [72, 66], significant practical limitations remain. Field measurements of emerging 5G-Advanced systems confirm that uplink bottlenecks continue to pose a considerable challenge for the efficacy of computation offloading policies [29, 24].

A key finding in the literature on task offloading is that adaptive strategies, achieved through dynamic resource allocation, consistently outperform the static modeling of individual system components. This research emphasis illustrates a preference for managing a system's response to an unpredictable environment rather than attempting to model it completely, a particularly relevant approach when machine learning tasks introduce real-world operational constraints. Current methodologies achieve this adaptiveness by concentrating on the multi-objective optimization of measurable outcomes like latency, energy, and the stability of control loops [26], which addresses environmental complexity by managing its effects.

While this approach is effective, it also exposes a gap in existing research, which has not yet fully explored how adaptiveness could be improved by interconnecting the computational and communication state with their respective control systems. The current use of continuous, intelligent resource reallocation has proven superior for meeting performance objectives under the variable conditions of MEC systems. However, some areas remain unexplored and could yield significant efficiency gains, including the deeper integration of state awareness into control logic, reasoning over the wireless propagation environment, and understanding the correlation between task-level performance and resource allocations.

2.2 Deep Reinforcement Learning Methods

Conventional optimization methods fail to adapt to the dynamic environments of Mobile Edge Computing Systems (MECS), where network conditions, computational loads, and user demands fluctuate and typically are unpredictable. This dissertation proposes a novel semantic approach to MECS optimization, and to evaluate its performance, we employ a suite of Deep Reinforcement Learning (DRL) methods. DRL extends the principles of optimal control to environments where an explicit system model is unknown or intractable. This review outlines the model-free DRL algorithms leveraged in our contributions, from foundational value-based methods to advanced policy-gradient techniques.

We evaluated our proposed approach with model-free DRL, which derives control policies directly from high-dimensional state vectors through trial-and-error [58]. DRL uses deep neural networks as function approximators to learn either a value function, which estimates the long-term utility of an action in a given state, or an optimal policy, a direct mapping from states to actions. A critical challenge in real-world systems like MECS is partial observability, where an agent's perception is an incomplete or noisy representation of the true state. We addressed this using algorithms such as the Deep Q-Network (DQN), which approximates the optimal action-value function, $Q^*(s, a)$. DQN's stability relies on two key mechanisms: experience replay which breaks temporal correlations by sampling from a large data buffer, and a target network, which provides stable Bellman targets. To correct for DQN's optimistic value bias, we also employ Double DQN (DDQN), which decouples action selection and evaluation to achieve more accurate value estimates [82].

A limitation of both DQN and DDQN is their focus on the expected return ($E[R(s, a)]$). By modeling only the mean of the return distribution (R), the agent becomes risk-neutral, which is often undesirable. To evaluate the proposed system under more nuanced risk profiles, we explore using methods from Distributional RL [9]. These algorithms learn the entire

probability distribution of returns instead of just its expectation. Specifically, we use C51, which approximates the distribution as a categorical distribution over a discrete set of supports, and Quantile Regression DQN (QR-DQN), which models the distribution by learning its quantiles directly, offering a more flexible representation [23]. This provides a richer and more stable learning signal for evaluating our approach.

While value-based learning is effective, many MECS problems involve continuous or high-dimensional action spaces where such methods are cumbersome. To address this, our evaluation framework includes policy-gradient methods, which directly optimize the parameterized policy $\pi_\theta(a|s)$. We specifically leverage Proximal Policy Optimization (PPO) due to its stability and sample efficiency [71]. Standard policy gradient algorithms can suffer from performance drops from a single large, poorly chosen update. PPO mitigates this by optimizing a 'clipped surrogate objective' function, which effectively creates a trust region to ensure that policy updates are not destructively large. This allows for more stable and monotonic performance improvement, providing a robust benchmark for our semantic MECS framework.

Chapter 3

Lower-Layered-Computing Logic: Adaptive Multi-Task Computing Allocation in MECS

3.1 Introduction

Machine learning is becoming an increasingly crucial component of mobile applications. Emerging applications, such as augmented reality (AR), often produce streams of data analysis tasks (e.g., image analysis or speech-to-text) that not only are computationally intense, but also heterogeneous in nature. While the "local" execution of the tasks onboard mobile platforms is inherently challenging due to the limitations of these platforms, emerging distributed computing strategies over layered systems - i.e., mobile-edge-cloud - suffer from the need to transfer information-rich data and signals over wireless and wireline channels with finite and time-varying capacity, as well as from the limitations of infrastructure-level servers (and especially edge servers) and the inevitable need to share such resources across multiple

mobile devices.

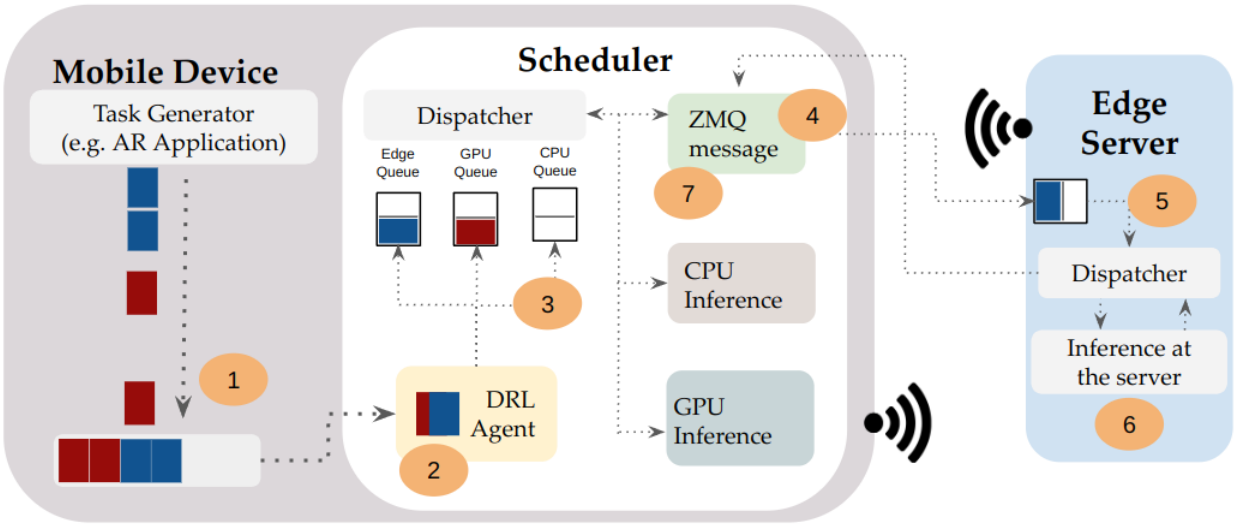


Figure 3.1: Illustration of system model considered in this chapter: a multi-layered mobile-edge system collaborates toward the timely execution of a heterogeneous stream of computing tasks. Each layer has multiple resources, e.g., CPUs and GPUs.

In this context, the allocation and optimization of computing resources across the layers of articulate systems is an extremely important problem. Recent contributions present frameworks that optimize performance metrics such as energy consumption [7], task performance [90, 30, 32], and latency [13] individually. However, most existing work tends to base their solutions on abstract models of the system's operations that often fail to capture the intricate behavior of real-world deployments, while also typically considering a homogeneous streams of tasks (e.g., object detection).

The focus of this chapter is the management of streams of heterogeneous tasks over layered systems composed of mobile devices (MD) and edge servers (ES) or ES and cloud servers (CS), where each unit embeds multiple resources: i.e., *Central Processing Units* (CPUs) and *Graphical Processing Units* (GPUs). The system is illustrated in Fig. 3.1. Different from most prior work, we consider the influence on the system dynamics of context switching, task accumulation, and the interdependence between communication layer timing and computing performance, which all have a considerable impact on the system behavior and performance.

In this context, we propose a predictive framework that controls in real-time (i) the routing of individual tasks across the resources and layers of the system, and (2) the machine learning model assigned to each task to minimize latency, energy consumption while maximizing task performance.

The engine of the framework we propose takes the form of a deep reinforcement learning algorithm (DRL), whose input features include multiple core system components such as CPU, GPU, memory usage, and wireless channel statistics. In order to train and test the framework, we developed a real-world system composed of a task generator, the DRL agent that receives and allocates the input queue of tasks, and a dispatcher that routes the tasks received based on the actions chosen by the DRL agent. Based on this setup, we collect a comprehensive dataset, described in detail in Section 3.7.

To summarize, this chapter makes the following contributions:

(1) We develop a framework for the management of heterogeneous computing tasks' flow across multi-layered systems that accounts for real-world effects such as context switching, task accumulation, and intricate interdependencies between communication and computing components of the system.

(2) We build a multi-layer system empowered with the ability to dynamically route tasks across layers and computing components. Notably, in contrast with contributions focusing computing on one of the layers (e.g., edge computing), all the layers of the system collaborate toward the efficient and timely completion of the incoming tasks. We perform dynamic allocation of heterogeneous machine learning tasks. We leverage the allocation and model decisions based on the status of each layer and implement a multi-objective performance function that maximizes the resources of each layer.

(3) We design and train a deep reinforcement learning algorithm that controls task routing and allocation taking as input statistics of computing and wireless resources, the queue of

incoming tasks, the status of the models, and tasks dispatched. Training is performed on a comprehensive dataset that we pledge to release to the community. The source code of our framework and the dataset is in `https://tinyurl.com/heterogeneousTaskScheduling`

(4) We evaluate our solution and show empirically that different settings and characteristics of the system (e.g. task arrival distribution, hardware configuration) result in different decisions regarding the allocation policy. In a setting where the MD is an embedded platform with relatively low computing capabilities (e.g., an NVIDIA Jetson Nano) the optimal policy privileges allocation of tasks to the CPU to avoid a large delay penalty associated with context switching – while also using low-complexity neural models to contain the increase in execution time with respect to allocating the task to the GPU. If a more powerful embedded platform is used (e.g., an NVIDIA Xavier AGX in our experiments) the controller expresses a more pronounced preference toward using larger neural models and performing inference locally on the GPU. We also show that fixed policies such as traditional edge computing (that is, all tasks are routed to the edge server) incurs a degraded latency up to 5x times, in the worst case, compared to a dynamic optimized policy.

The rest of this chapter is organized as follows. In section 3.2, we illustrate the impact of context-switching on end-to-end delay when inference is performed in multiple processors and platforms. Next, in section 3.3 we summarize relevant contributions to the tasks allocation and routing in multi-layered systems and compare them with our work. We propose a system model and its performance metrics in section 3.4. In section 3.5, we provide details about the hardware setting and the software implementation of our system. We describe our solution in Section 3.6. Sections 3.7 and 3.8 describe the dataset collected to perform experiments as well as the results obtained using our proposed framework. Finally, Section 3.11 discusses the particular results of this chapter.

3.2 Preliminary Experiments:Context Switching Problem

In this section, we illustrate the impact of context switching - which to the best of our knowledge is not considered in existing literature on task management in collaborative multi-layered systems. In order to support heterogeneous tasks, unless sufficient resources (e.g., memory) are available, the system will need to switch context as the current task changes over time. For instance, a neural network model needs to be loaded in the memory of a GPU in order to be executed. This preparation phase may take a considerable amount of time (e.g., a large fraction of a second), and may impair the continuity of execution of the stream of tasks.

To illustrate the impact of model loading, we consider a setting where two platforms from the NVIDIA Jetson family – particularly the Jetson Nano, and Xavier AGX – are used as mobile devices. We note that they differ in terms of computing capabilities, the internal configuration of resources, and energy consumption. The NVIDIA Jetson Nano (NVIDIA JN) architecture has a shared physical memory between the CPU and GPU. The key advantage of this architecture is that data transfers are performed less frequently compared to an architecture with separate discrete memory per processor (that is, CPU memory isolated from GPU memory). However, the small size of the memory prevents the program from opening multiple processes to decouple the loading from the inference in neural network models. The latter architecture platform used as MD, the NVIDIA Xavier AGX (NVIDIA JXA), has high memory bandwidth – the speed we can read or store data into the memory, deep learning accelerators, and a cache coherence between CPU and GPU which helps reduce latency overhead and bandwidth usage. The cache coherence [10] is important in our case as a data tensor or neural model can be stored and retrieved directly from it instead of the main memory through the CPU. Our framework automatically adapts the management of

the computing tasks across the layers of the system and their components to these factors without the need for the designer to hard-code a platform-specific policy.

We illustrate our observations by means of simple experiments over the setting we will describe in detail later in the chapter. In these experiments, we generate a stream of 200 tasks composed of two distinct classes, that is, image classification and image segmentation, each associated with a neural model (SegFormer-B1 [86] and EfficientNet-B1 [77]). The number of tasks of the same class is distributed uniformly such that when there are 2 context changes in a stream of 200 tasks, we have 100 image classification tasks and 100 image segmentation tasks. As shown in Fig. 3.3 and 3.2, context switching has a computing platform-dependent impact on the execution timeline. In the figures, we can see that the CPU – a general-purpose processing unit that fetches instructions out-of-order but executes them sequentially – is not optimized to perform matrix operations associated, for instance, with the execution of neural network models and tends to have a stabilized task delay even against context switching. Conversely, the GPU, a specific purpose processing unit that performs parallel thread execution, requires data transfer from the main memory at least once, and potentially at every context switch. This can have a considerable impact on task delay when a new task is allocated to a specific unit. In addition to the expected differences among processors, there are some differences in the task delay dynamics which depend on the platform architecture. As shown in the pictures, if the NVIDIA JN is used, the CPU inference delay increases linearly as the frequency of context switching increases while the GPU inference increases close to exponentially due to their memory architecture.

We note that context switching is influenced by the order of task arrivals into the system. We also observe how the timing of task arrivals, which at a fine grain is also influenced by the communication layer as well if tasks are allocated to the edge server – has an impact on batch processing on the GPUs: a critical component of inference acceleration.

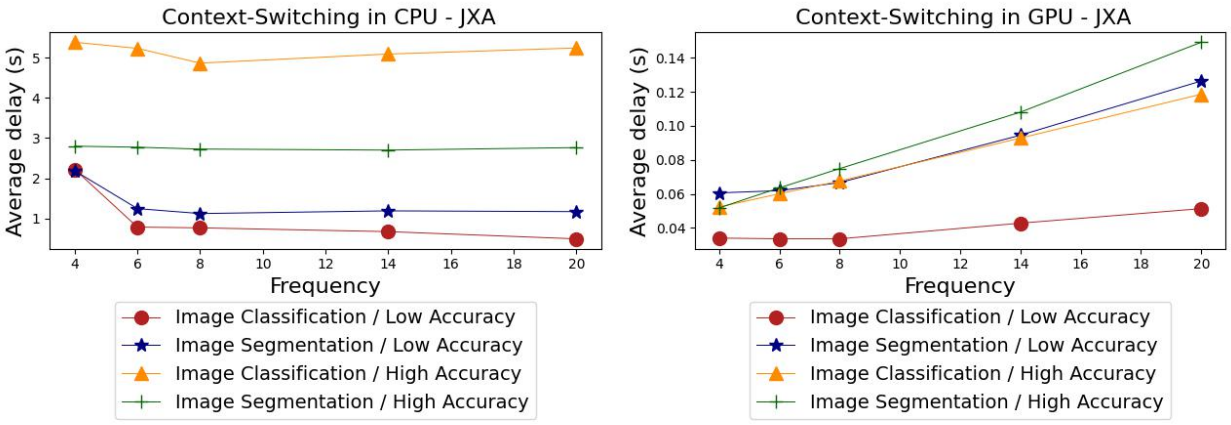


Figure 3.2: Loading model and inference latency in NVIDIA Jetson Xavier AGX for different context-change values across different ML tasks and processing units.

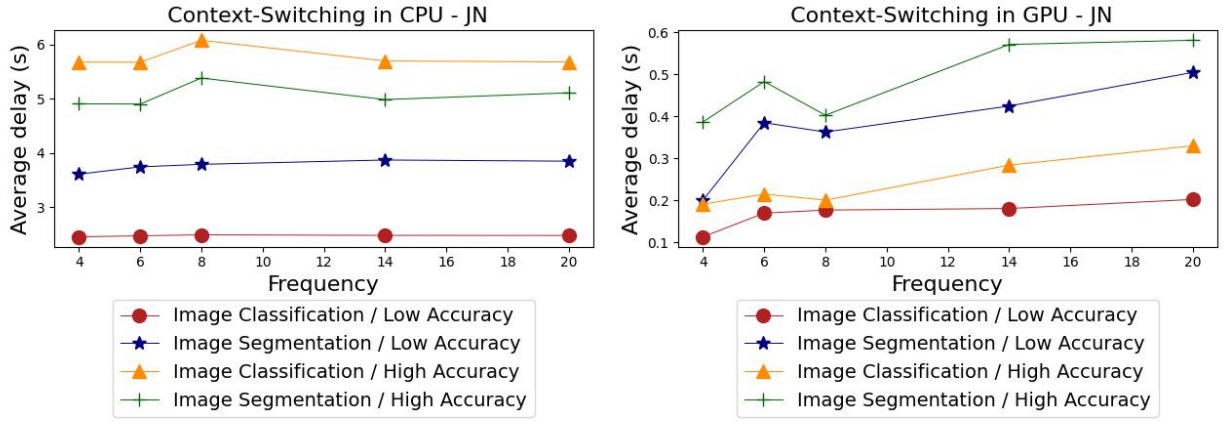


Figure 3.3: Model loading and inference latency in the Jetson Nano for different context-change frequencies across different ML tasks and processing units.

3.3 Related Work

To the best of our knowledge, there are no available studies on context-switching and inference performance interdependence focused on multi-layered systems. The closest class of contributions in the literature targets the optimization of inference performance by task offloading in one or more layers (e.g. mobile device to edge server or edge server to cloud). We summarize this work and contrast it with the contributions of this chapter in the following.

Task offloading in the context of edge computing – often referred to as Mobile Edge Computing Systems (MECS) – has been extensively studied [50]. We can categorize this problem by the execution methodology: *full offloading* - where the whole computing task is offloaded to the edge server, *partial offloading* - where portions of the computing task are offloaded and others are executed locally on the mobile device. Guo *et al.* [32] and Fresa *et al.* [30] are examples of full offloading where the objective is to maximize the accuracy of inference tasks. Matsubara *et al.* [52] and Zhao *et al.* [92] perform partial offloading to optimize the balance between end-to-end delay, energy consumption and task performance using techniques such as split computing, early exit, and data compression [54].

Mobile-Edge-Cloud cooperation frameworks have been developed. For instance, Hong *et al.* and Zhang *et al.* [36, 90] present methodologies to allocate data tasks on the available resources optimizing customer performance metrics. These frameworks showed promising results in simulated data and were controlled under the assumption of homogeneous tasks and hardware setups.

Task scheduling and placement in MECS has been largely studied to optimize multiple performance metrics such as end-to-end delay and available resources (e.g CPU, GPU) [78, 73, 80], and [91] among others. In particular, Shu *et al.* [73] consider explicit task dependencies and heterogeneous servers to reduce the overall task delay. Zhang *et al.* [91] also focuses on task delay and tasks dependencies, however, the proposed solution is evaluated on real-world

heterogeneous edge servers (i.e. multiple CPU, GPU, and network configurations).

3.4 System Model and Optimization Objectives

In this section, we first describe the model of the system we consider and then define performance measures and optimization objectives.

3.4.1 System Model

We consider a multi-layered system where a MD and an ES collaborate to complete a stream of tasks generated by the MD. Formally, we define the task arrival process $\mathbf{T} = \{T_i\}_{i=1,2,\dots}$, where $T_i{=}(t_i, c_i)$ is the i–th task arrival described by the arrival time t_i and $c_i{\in}\{1, \dots, K\}$ is the task class. The MD and ES are wirelessly connected and composed of multiple computing resources - namely GPUs and CPUs - characterized by different computing power. We denote resources at the MD as R_n^{MD}, $n{=}\{1, \dots, N^{\text{MD}}\}$ and at the ES as R_n^{ES}, $n{=}\{1, \dots, N^{\text{ES}}\}$, and group them in the set R_n, $n{=}\{1, \dots, N^{\text{MD}} + N^{\text{ES}}\}$.

At a high level, in this work, a task corresponds to a set of data to be analyzed using a deep neural network. When a task arrives at the MD, it enters the finite queue Q^{MD}. Tasks are then processed sequentially by the management function $(n_i, m_i){=}A(T_i)$, where n_i points to a resource in $\{1, \dots, N^{\text{MD}}{+}N^{\text{ES}}\}$. The control variable m_i determines the neural model used for the execution of the task. Multiple queues in the MD are used to route the incoming tasks to a particular resource queue (See Fig.3.1).

If n_i corresponds to a resource on the MD, then the task is sent to that computing unit - which has its internal task queue. Conversely, if n_i is a resource on the ES, the task is sent to the radio interface and will eventually enter the ES' queue Q^{ES} - where the entry will also

contain the pair (n_i, m_i).

3.4.2 Performance Measures

We define a set of relevant performance measures that will guide our optimization rationale.

Task-Level: We define two key metrics at the task level: *task performance* and *task latency*. The former is primarily associated with the selection of the model used to complete the tasks. For instance, a more complex neural model will most likely achieve a better average performance (where the average is over the data samples acquired by the MD) compared to a lower complexity model. We then define $p_c(c, m)$ as the expected normalized task performance associated with a task of class c when model m is used.

Moreover, we denote h_i as the time at which task i is completed and define the task latency as $l_i = h_i - t_i$. We note that latency is the sum of multiple components, including queueing time, execution time, communication time (if the task is executed on the ES), and context switching. While it is indeed possible to abstract these components, in real-world systems they are determined by complex software-hardware effects and interdependencies that are difficult to capture using available modeling strategies, also due to the influence of temporal patterns of the system state.

System-Level: At the system level, we focus on *power consumption* at the MD. We define the instantaneous - normalized - power consumption time τ as w_τ. Intuitively, and similarly to the latency, the power consumption also is the result of a rather complex definition of system state and its pattern that makes it hard to use simple abstractions.

3.4.3 Optimization Objectives

The objective of the framework we propose is to jointly maximize the expected task performance

$$P = E\left[\sum_i p_{c_i}(c_i, m_i)\right], \tag{3.1}$$

while minimizing the expected task latency

$$L = E\left[\sum_i l_i\right] \tag{3.2}$$

and average power consumption

$$W = E\left[\int_\tau w(\tau) d\tau\right]. \tag{3.3}$$

The expectations are computed over realizations of the stochastic process describing the system behavior. We remark that due to the complexity of the system we resort to optimization and evaluation based on datasets collected experimentally.

3.5 System Architecture

In this section, we describe in detail the system we developed. An overview of the system is provided in Fig. 3.4. The software we developed has a modular structure and includes the following modules: task generator, dispatcher, and logger. Note that the latter is not only used to evaluate performance, but also to compose the real-time state used by the DRL agent to select the actions. Our task generator initiates a background process that generates a stream of tasks according to a particular distribution. Our dispatcher manages the reception of tasks and the execution of the tasks using torch libraries such as multiprocessing [59] and

cuda [21], when tasks are allocated locally, the dispatcher is able to perform loading (when required) and inference. If the tasks are allocated to the Edge, the dispatcher prepares and sends the message to be transmitted using the wireless interface of the MD. Our loggers are hardware customized, where we use the Tegra Utility [20] integrated within the Jetson platforms to obtain the state of the hardware.

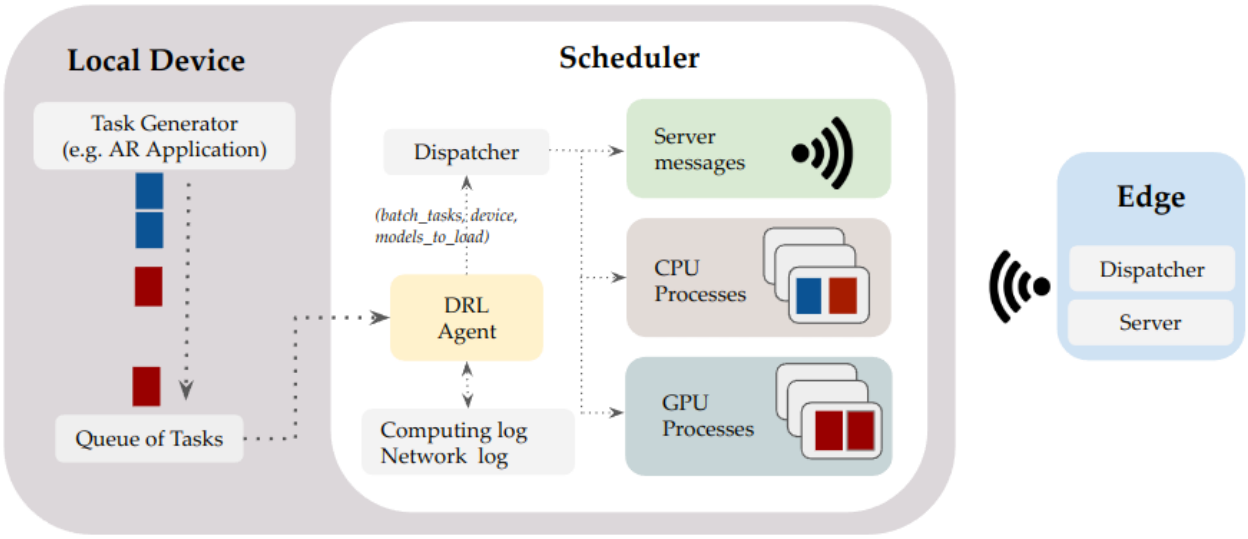


Figure 3.4: Schematics of the system implementation.

3.5.1 Computing Tasks

We developed a tasks generator module that defines a sequence of tasks to be dispatched. This module creates distributions of tasks based on three parameters: (1) the set of tasks, (2) the distribution of the time between tasks, and (3) the probability of switching from one task to another in the arrival sequence.

For the sake of simplicity, we focus our attention on computer vision, specifically on image classification and semantic segmentation. This choice is motivated by the availability of families of models designed for resource-constrained systems providing performance proportional to their complexity [86, 77]. We consider two families of models: EfficientNet

(which we could identify as m_0 and m_1 for effficientnet-b0 and efficientnet-b1) [77] for image classification and SegFormer (which we could identify as m_2 and m_3 for segformer-b0 and segformer-b1) [86] semantic segmentation. We use pre-trained models from the HuggingFace [19] repository.

In general, image classification models tend to have lower complexity compared to those designed for the much more difficult image segmentation task. This leads to different memory usage and, in general, task execution times for the two tasks. For instance, model m_1 or efficientnet-b1 has 19M parameters takes 0.3s in JN and 0.071s in JXA while m_3 or segformer-b1 with 50.3M parameters takes $0.5s$ in JN and 0.075s in JXA.

3.5.2 Task Dispatching

We implement the task dispatching process using batches, where a batch is a minimum number of tasks needed to initiate the allocation process. We use the multiple producer-consumer paradigms to establish communication between the task generator and the scheduler. The input queue is a process-safe data structure that shares data between processes allowing the tasks to be passed from the Task Generator module to the DRL agent. Once the tasks are received by the DRL agent, we obtain the most recent network and computing features. The computing and network logs are hardware-customized modules running in the background to provide an accurate state of the mobile device.

3.5.3 Hardware-Software Setup

The MD and ES hardware setup differ in capacity, but also in the OS implementation, which has implications for how we extract the real-time state of the hardware and the system operations. Our framework is built to run on multiple NVIDIA boards as MD and multiple

Desktop-Linux computers as ES (See Tables 3.1, 3.2).

The wireless antenna adapter used in the JN platform was used to transmit messages at 2.4GHz, the antenna gain is 5dbi. We use the same frequency to transmit messages from the JXA platform, however, the antenna gain is slightly higher (6dBi). We empirically observed a neglectable change in the RTT for the proposed scenarios.

	Mobile Device NVIDIA Jetson Nano	**Edge Server Laptop**
CPU	Quad-core ARM A57 @ 1.43 GHz	11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
GPU	128-core NVIDIA Maxwell™	NVIDIA GeForce RTX 3050 Ti
Memory	4GB 64-bit LPDDR4 25.6GB/s	32 GB DDR4 at 3200 MHz (2 x 16 GB), dual channel
AI Performance	472 GFLOPS	1.69TFLOPS
Network	Wifi 802.11 ac	Wifi 802.11 ac

Table 3.1: Hardware Settings I

	Mobile Device Jetson Xavier AGX	**Edge Server Jetson Orin**
CPU	8-core NVIDIA Carmel Armv8.2 64-bit	12-core Arm Cortex-A78AE v8.2 64-bit
GPU	NVIDIA Volta architecture with 512 NVIDIA cores	NVIDIA Ampere architecture with 2048 NVIDIA CUDA cores
Memory	64GB 256-bit, 136.5GB/s	32GB 256-bit, 204.8 GB/s
AI Performance	22TOP	275 TOPS
Wifi	Wifi 802.11 ac	Wifi 802.11 ac

Table 3.2: Hardware Settings II

3.6 Deep Reinforcement Learning Framework

As mentioned earlier, we optimize task allocation across the layers and resources of the system as well as the model used to complete the task. In the following, we define the

structure of decision-making, how the state is composed, and how we transpose the abstract performance measures we defined earlier into the real-world system we deployed.

We adopt a double deep Q network (DDQN) [82] algorithm as the core engine of our framework because it is a sequential decision-making problem. DDQN is model-free DRL that leverages the greedy policy implemented by deep Q Network (DQN) algorithms while reducing overestimation. In the DDQN algorithm, the greedy policy is evaluated using a second neural network. We refer the interested reader to [82] for a thorough description of the algorithm, which we summarize at a high level in Algorithm 1.

3.6.1 Decision Timing

We define a time step as the interval between a task arriving at the Q^{MD} (i.e. input queue) in the DRL agent and the result of the inference being obtained. The input queue is separated into a queue per resource available (Q^{R_i}) which enables asynchronous inference execution processes and consequently stores the individual real-time latency. This management of tasks facilitates the differentiation between the delay of the task and the total time in the time step. In fact, the time step is an upper bound of the latency.

3.6.2 State and Action space

The state space represents the available resources and interfaces in the MD, which is readily observable by the agent without the need to exchange information over the wireless link. In particular, we consider blocks of computing, task, model, and network features. We then define the global state as $S = \{Q^{MD}, C, T, M, N\}$ composed of 43 parameters, where Q^{MD} is the number of tasks in the input queue, C is the state of the computing resources in the MD, T, and M are the statistics of the recent tasks dispatched and the neural network

models recently used, and N is a set of features from the IP, TCP, and wireless interface. The specific variables used in the implementation are given in Section 3.7.

Actions are applied to batches of tasks, and the action space is $A = \{a_0, .., a_N\}$, where $a_i = (m_j, p_k)$ is the pair composed of the neural model to be used to perform inference and the resource (e.g. CPU, GPU) chosen to perform inference at the current batch of tasks. Therefore $N = M \times P$, where M is the maximum number of neural network models per task and P is the maximum number of processors that can be used based on the hardware specifications.

Algorithm 1 Multi-Layered System with Double DQN

Initialize Agent and Neural Network parameters: primary network Q_θ, target network $Q_{\theta'}$, and replay buffer$D$
Initialize Framework parameters: dispatcher configuration, task generator, computing feature logger, network feature logger processes
Initialize Environment parameters: action space, size of the batch of tasks, number of classes of tasks, the wireless connection configuration
for episode $e = 1, N$ **do**
  while (dispatched tasks < total tasks & size(input queue) > 0) **do**
    Select a random action a_t with probability ε.
    Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t, collect reward r_{t+1} and observe next state s_{t+1}
    Log the experience (timestamp, $s_t, rt + 1, s_{t+1}$)
    Store the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$
    if s is terminal state **then**
      break
    end if
  end while
end for
for each update state **do**
  sample $e_t = (s_t, a_t, r_{t+1}, st + 1)$ from $\mathcal{D}$
  Compute target Q value:
  $Q^*(s_t, a_t; \theta) \approx r_t + \gamma Q_\theta \left(s_{t+1}, \arg\max_{a'} Q'(s_t, a')\right)$
  Do gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
  Update target network parameters:
  $\theta' \leftarrow \theta$
end for
Terminate all processes initialized by the Framework

3.6.3 Reward Function

As indicated in Section 3.4, we define a multi-objective optimization objective that includes task delay, normalized task accuracy, and power consumption (measured over a time step). Our reward function reflects the objectives using a sigmoid function that penalizes the performance metrics to be outside a desired region. The overall performance is calculated as the sum of the weighted average of the normalized optimization objectives. This implies that the normalized accuracy is used as inverse since the more accurate, the smaller the value of the overall performance is and consequently, the reward is higher.

We compute the expectation of the performance as the average normalized accuracy of the task class based on the model selected. Since we use trained neural network models, we consider the documented accuracy to normalize the values in an interval [0,1] such that they are comparable. The latency is measured in real-time, we timestamp tasks at arrival time and retrieve the total time when the results are available to the MD. This latency is averaged among the tasks dispatched in each batch. Finally, power consumption is obtained as an instantaneous measure from the kernel variables. We use the interval duration to extract an approximate energy consumption within the time step. Energy consumption also is normalized.

Then, the overall reward function is defined as follows:

$$R_t = \begin{cases} S & \text{if resources available at time } t \\ -1 & \text{otherwise} \end{cases}, \tag{3.4}$$

where $S = 1/(1 + e^x)$ and $x = \sum_{i=1}^{N} w_i \dot{\delta}^*$ and δ represents the vector of the normalized performance metrics, in our case: end-to-end delay, accuracy and power consumption.

3.6.4 Architecture

The architecture of the proposed DDQN implementation relies on custom modules such as the environment, task generator, dispatcher, and loggers. The main characteristic of the DDQN algorithm is the experience replay and target network deployment to stabilize the optimization. In our case, to build the online neural network model we adopted a sequential structure starting with 3 layers of 1-dimensional convolutions and Relu as an activation function, a flattened layer, and 2 lineal layers with Relu as an activation function. Our inputs are state-action pairs, which is the 1-dimensional vector and the output is the Q-value that corresponds to the selected action.

3.7 Dataset

In order to train the neural network embedded in the DDQN, we collected a dataset that captures the experiences of our DRL agent in a complex environment with the aim to test and evaluate schemes build to optimize computing resources.

We implemented two types of policies in the experiments, the baseline policies which are CPU, GPU, and EDGE ONLY. The CPU policy refers to performing inference of all tasks generated on the CPU regardless of the context change, the same rule applies to the GPU ONLY and EDGE ONLY policies. The latter type of policy implemented is random which provides a non-biased sample of the action space. We log the state of the MD in the computer, network, tasks, and model features for the time the experiments ran. Finally, we log the output or reward of our system implementation. As part of modeling a complex environment, we use two scenarios composed of two hardware settings to show that variation in computing and wireless capacities enables variation in the optimal allocation policy. The hardware settings are listed in 3.1 and 3.2. The first setting uses an NVIDIA JN board as

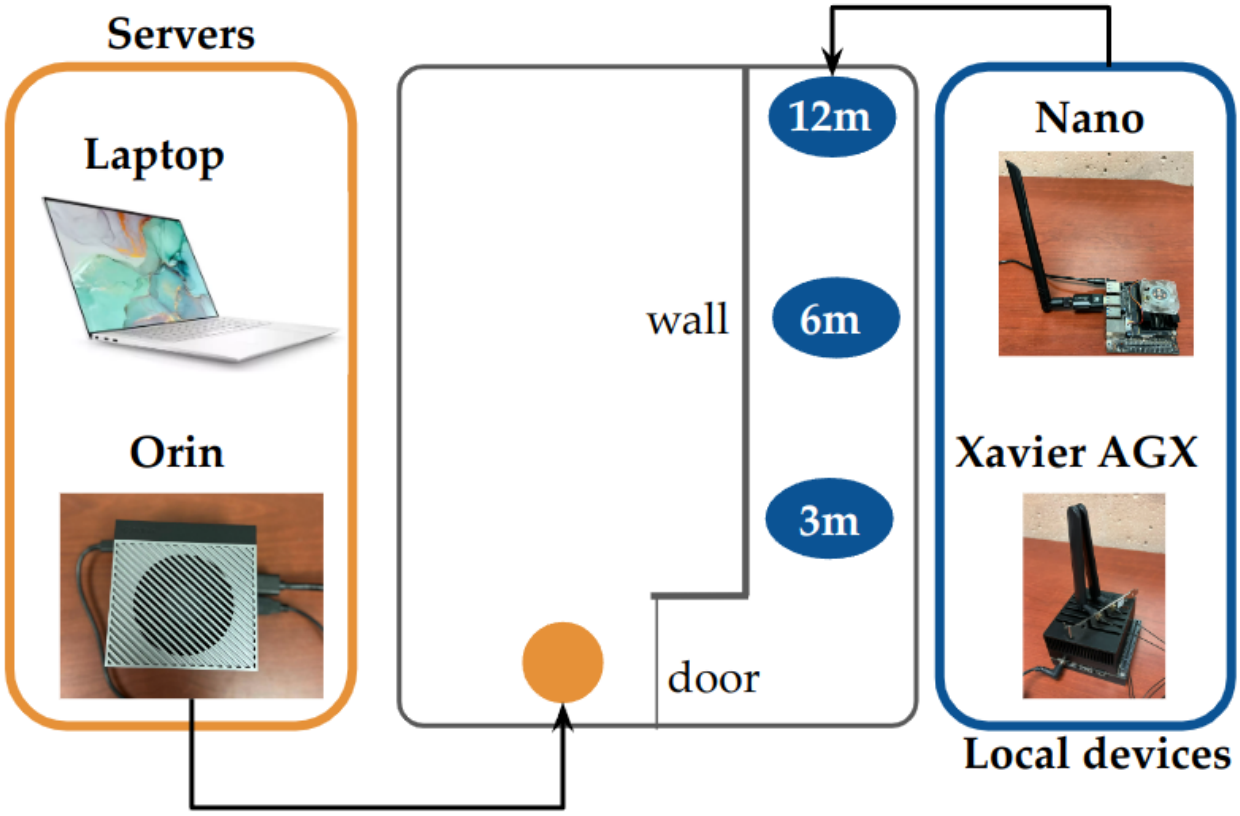


Figure 3.5: Experimental setup: Indoors space with a wall and a door between the Edge Server and the Mobile Device. The mobile device was located in a hallway at different distances from the Edge server.

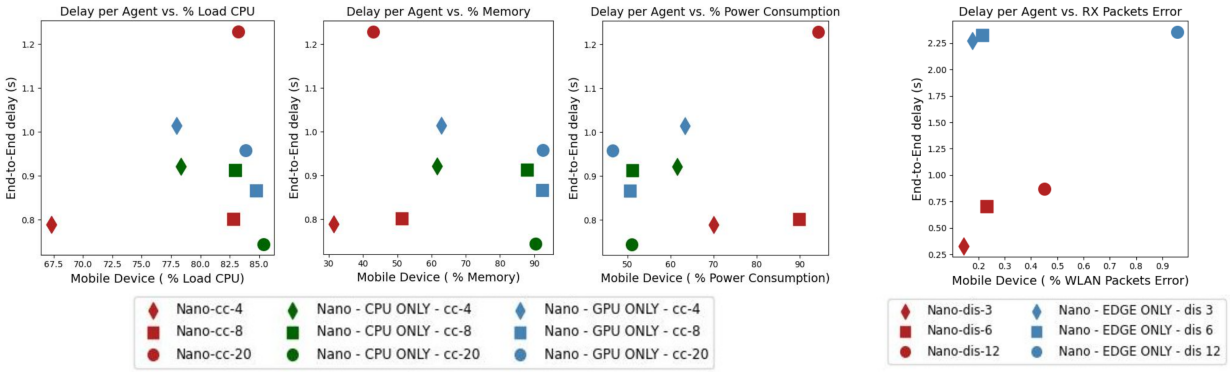


Figure 3.6: End-to-end tasks delay vs. state features for NVIDIA Jetson Nano. The left plots show mobile device features with the agents trained to adapt to the context changes. The right plot shows the wireless representative feature with the agents trained to adapt to the changes in distances

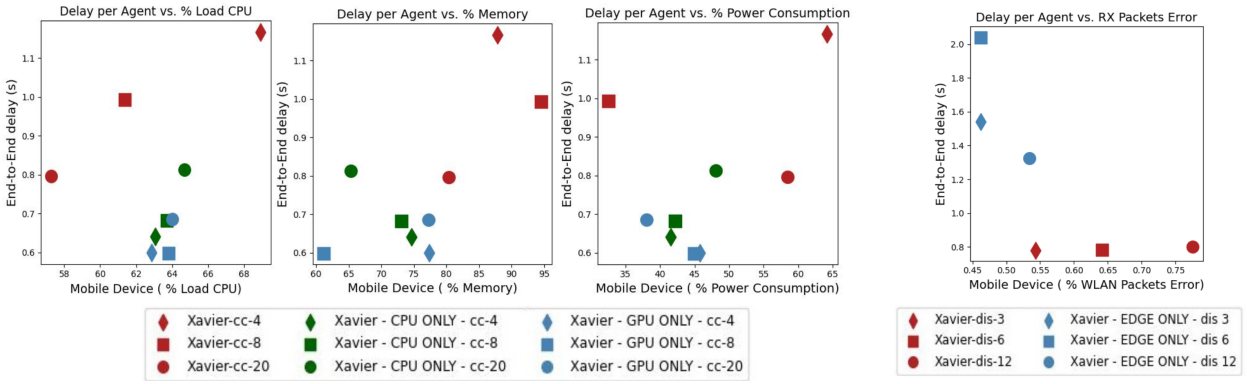


Figure 3.7: End-to-end tasks delay vs. state features for Jetson Xavier AGX. The left plots show mobile device features with the agents trained to adapt to the context changes. The right plot shows the wireless representative feature with the agents trained to adapt to the changes in distances

MD: an embedded computer with limited computing capabilities. This device is designed for simple machine-learning models. Moreover, the CUDA memory is shared with the CPU, which results in some limitations if we decouple the loading from the inference in machine learning tasks. The wireless interface is not integrated into this board, we have added a USB Wifi antenna adapter that facilitates wireless communication. In the first setting, we use a Laptop - Linux as the ES. This latter platform not only possesses a larger computing power but also uses a standard architecture. The second setting uses an NVIDIA JXA board as MD: an embedded platform that reduces task latency and memory bandwidth due to the zero-copy functionality active for the CUDA memory. For this board, we acquired a Wifi antenna, with a larger antenna gain than the JN. As ES, we use an NVIDIA Orin board, which has 10x the computing power and embeds inference accelerators [25]. We define experiments based on the following parameters: a hardware setting, a specific distance between the ES and the MD, a distribution of the heterogeneous tasks and a policy. We recorded information from 9 experiments per hardware setting and per policy (see details in Table 3.3). The dataset is composed of a collection of the DRL agent experiences per experiment, an experience contains the timestamp, the state, the chosen action, and the next state. The state is formed by the computing, tasks, model, and network features.

Parameter	Values
Hardware settings	Table 3.1 and Table 3.2
ES-MD distance	3, 6, and 12 mts
Tasks distributions	4, 8, 20 context changes
Policies	Random, EDGE ONLY, CPU ONLY, and GPU ONLY
Actions configuration (processors)	GPU, CPU and EDGE accordingly
Actions configuration (models)	Semantic segmentation: SegFormer B0 and B1 [86]. Image Classification: EfficientNet B0 and B1 [77]

Table 3.3: Experiment parameters

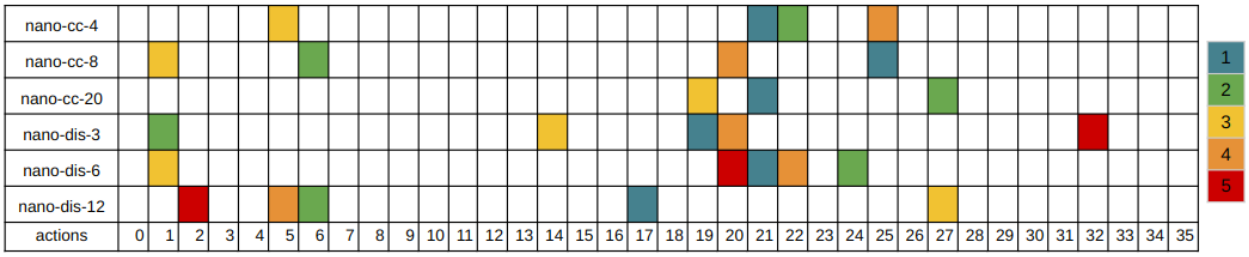


Figure 3.8: Ranking of actions per trained agent within scenario I

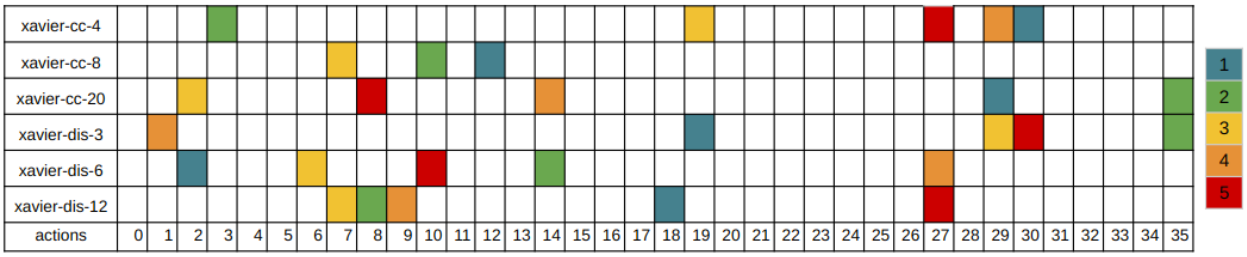


Figure 3.9: Ranking of actions per trained agent within scenario II

3.8 Results

Based on the dataset described in the previous section, we train the DRL agent and deploy it on the considered scenarios. We name the scenarios in terms of MD, policy, and frequency of context switching. For instance, the policy deployed by the agent in an environment when the context changes 4 times and uses the information of the NVIDIA JN is defined as *nano-cc-4*. The weights of latency, accuracy, and energy consumption we use in the training

phase are $[0.6, 0.3, 0.1]$, respectively.

In Fig. 3.6 and Fig 3.7, we show the average end-to-end task delay for selected state features of each hardware setting. The % of CPU load, memory, and power consumption are features of the state of the mobile device, while the % of lost WLAN packets corresponds to changes in link quality. Policies that allocate all the tasks to one processor emphasize the impact of context change, which tends to dominate the other delay components as we increase the degree of heterogeneity of the tasks. For instance, under the CPU ONLY policy, the overall latency in the various environments tends to saturate as the frequency of context switching increases. The GPU ONLY policy tends to consume less power and more memory compared to CPU ONLY policy in all environments. As expected, the EDGE ONLY policy suffers from the impairments of the wireless channel, especially as the MD-ES distance increases.

The policy adopted from the environment where the MD has an NVIDIA JN hardware setting is shown in Fig. 3.6 (nano-cc-4, nano-cc-8, and nano-cc-20). In Fig. 3.6, we observe that the DRL agent achieves smaller delays than the baseline policies – CPU ONLY and GPU ONLY – when the context change frequency is smaller than 20. This is because the DRL agent policy is prioritizing delay over power consumption of the MD. Further, the number of consecutive tasks of the same class is larger compared to the nano-cc-20 environment, and then there is more room for adaptation. Consequently, the improvement of the end-to-end delay when the context change is 8 is higher compared to 4. On the other hand, the DRL agent deployed in the environments where the distance between the MD and ES varies (nano-dis-3, nano-dis-6, nano-dis-12) show a superior reduction of the end-to-end delays (60% of the time) as the DRL agent policy is choosing to perform some tasks locally.

The policy obtained from the environment where the NVIDIA JXA is the MD, showed in table 3.2 (xavier-cc-4, xavier-cc-8, xavier-cc-20), only improves the end-to-end delay compared to the baseline policies – CPU ONLY and GPU ONLY – for the scenario when the context change occurs 20 times in a sequence of tasks. This is because the hardware is pow-

erful enough to repeatedly use complex models, and therefore the policy is prioritizing local computing with a stable delay for the low frequency context change scenario. Additionally, we note that the end-to-end delay achieved by our policy is 2.5x times smaller compared to the EDGE ONLY policy.

In figures 3.6 and 3.7, we show the relationship between the state of the MD and one of the metrics used to optimize the trained agent (e.g. end-to-end delay). Importantly, the actions are taken by the DRL agent to achieve such performance show that different policies apply to different scenarios. To illustrate this effect, we show the ranking of actions used in each of the policies adopted by the DRL agent (see Fig. 3.8 and Fig. 3.9). The most selected actions for variations in context change frequency for the NVIDIA JXA setting are a low complexity model in CPU for the image classification task and either high or low complexity in ES for the segmentation task; which shows that diversification of resources for tasks allocation enables performance improvements. This policy is reasonable due to the convergence in delay that occurs when the model is placed in the CPU instead of the GPU.

The policy obtained from the environment that varies the hardware of the MD and ES follows a pattern based on the context-change frequency; for instance, in the NVIDIA JN setting the most selected action for the scenario with the least context-change frequency is the same as the one with the highest context-change frequency (nano-cc-4 and nano-cc-20 respectively). This is the option with a high-complexity model for the image classification tasks and a low-complexity model for the image segmentation tasks placed in the CPU. Only the nano-cc-8 environment's first choice places the models in the GPU and the ES with low and high complexity respectively.

The policy obtained from the environment that varies in MD to ES distance (e.g. nano-dis-3, nano-dis-6, nano-dis-12) in NVIDIA JXA and NVIDIA JN implements different actions. NVIDIA JXA agents privilege actions that select resources onboard the ES. For instance, the second in the ranking of actions chosen in the NVIDIA JXA environment includes dispatching

all image classification tasks to the ES, where the high complexity is preferred in the closest distance and the low complexity is preferred when the distance between the mobile device and the ES increases. Conversely, the second choice in the ranking of the choices in the NVIDIA JN setting shows a preference for dispatching tasks locally where the GPU is used at the same time as the CPU. For actions 1, 24, and 6 (See Fig. 3.9) the image segmentation tasks are always dispatched to the GPU. This empirically shows that the policy is adapting to the changes in hardware in each scenario as well as to the changes in the environment.

3.8.1 Deep Learning Problem Formulation

We propose to use DDQN to optimize the tasks allocation. A model-free Deep RL that leverages the greedy policy implemented in DQN while reducing overestimation. In the DDQN algorithm, the greedy policy is evaluated using a network different from the one that estimates its value. Consequently the Q value given a state s, an action a and the parameters of the network θ is calculated as follows:

$$Q(s, a; \theta) = R + \gamma Q(s', argmax_{a'} Q(s', a'; \theta); \theta') \tag{3.5}$$

The action space is $A = \{a_0, .., a_n\}$ where $a_i = (m_j, p_k)$ which represents the model and the processor chosen to perform inference at the current batch of tasks. Therefore $n = M \times P$, where M is the maximum number of models per tasks and P is the maximum number of processors that can be used based on the hardware specifications. For instance, we consider the Edge Server as one powerful processor.

The state space $S = \{C, T, N\}$ represents the device condition in a time-step as a function of its computational capabilities, tasks features and network status. The reward function is

defined as follows:

$$R_t = \begin{cases} S & \text{if resources available at time } t \\ -1 & \text{otherwise} \end{cases} \tag{3.6}$$

Where $S = 1/(1+e^x)$ and $x = \sum_{i=1}^{N} w_i \dot{\delta}^*$, which represents the sum of the weighted average of the normalized optimization objectives (e.g. end-to-end delay, energy cost, and accuracy) during the time step.

3.8.2 Task Characterization

Two tasks belong to the same type only when the data that carries and the models needed to perform inference are valid e.g. the image classification tasks arrive in average every 3 seconds, the frame resolution is the same and the valid models are EfficientNet-B0 and EfficientNet-B4.

We model the tasks generation as distributions based on the number of new model loadings that can occur in a period of time. A new model loading occurs when the new tasks coming is different from the one that was just dispatched.

We considered tasks patterns of 200 tasks, in which we vary the frequency of the context change. For instance, if we have 4 context changes in a series of 200 tasks, we generate a different type of tasks every 50 tasks.

3.8.3 Environment

Our environment relies on the measurement of three independent set of features which are: (1) network features e.g. RTT, dropped packets, (2) computing features e.g. CPU usage, CPU-memory usage, and (3) tasks features e.g. size of the data carried, size of the model needed for each task. Additionally, restrictions in the computing capabilities availability as well as of the machine learning models are considered.

3.8.4 Framework Pipeline

We developed a framework to test our scheduler on scenarios in real-time for multiple platforms. The scheduler receives a queue of tasks as input and a trained RL agent uses a dispatcher to allocate the tasks among the processing units. Task allocation is either sequentially or parallelly based on the capabilities of the hardware we are using. An overview of the framework pipeline is shown in Fig 3.4.

We use the multiple producer-consumer paradigm to establish communication between the tasks generator and the scheduler. Once the tasks are generated and received by the DRL agent, we obtain the most recent network and computing features. The computing and network logs are hardware customized modules running in the background to provide an accurate state of the mobile device.

The type of tasks and models used to perform inference change based on the application that the mobile device is running. Subsequently, we have decoupled the tasks allocation decision-making from the execution of the tasks in the mobile device which facilitates the incorporation of heterogeneous tasks into the pipeline. The dispatcher manages the model allocation per processing unit as well as the inference per batch of tasks sent to the DRL agent.

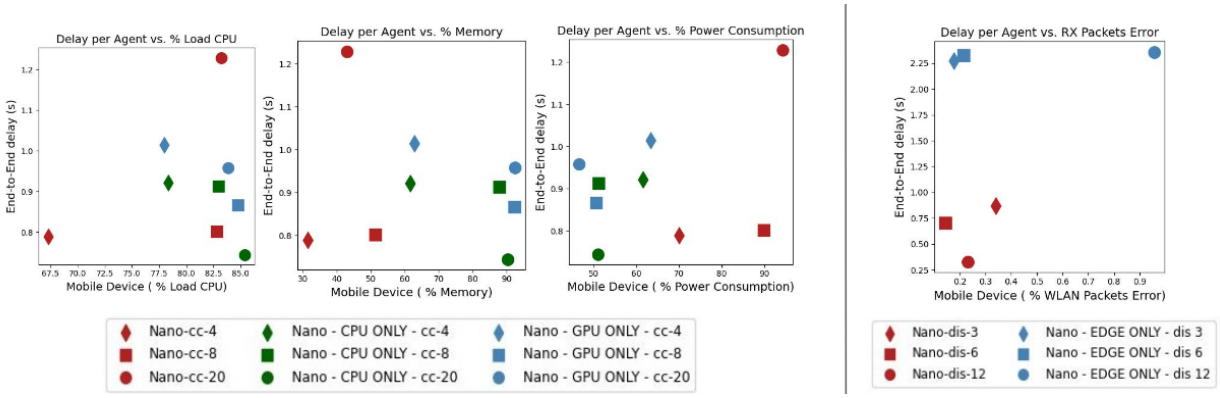


Figure 3.10: End-to-end tasks delay vs. state features for NVIDIA Jetson Nano. Left plots show mobile device features with the agents trained to adapt to the context changes. Right plot shows the wireless representative feature with the agents trained to adapt to the changes in distances

We use the dispatcher to load models and execute the respective inference based on the processor that has been selected from the DRL agent. In this regard, the DRL agent receives a queue of heterogeneous tasks and distributes a batch of tasks to one or multiple processing units though the dispatcher. The Edge server communication is also part of the dispatcher. We use a publisher-subscriber mechanism to send and receive messages from the Edge server which provides scalability for the servers and the clients. Since each type of tasks and model requires preprocessing data, the dispatcher also provides statistics per batch of tasks that has been dispatched.

The DRL agent actions indicate which model is used to execute the inference in a batch of tasks in the mobile device as well as in the Edge Server, therefore a Server object and an instance of the dispatcher also works at the Server side.

3.9 Experimental Setup and Data Collection

We developed a experimental setup to test the interoperability of the proposed scheme among servers and mobile devices. We used two scenarios with a set of servers and mobile devices

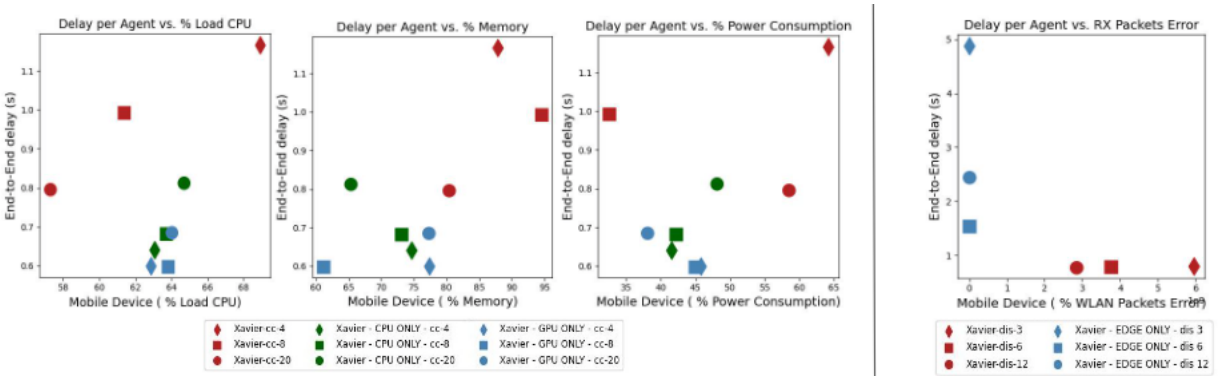


Figure 3.11: End-to-end tasks delay vs. state features for Jetson Xavier AGX.Left plots show mobile device features with the agents trained to adapt to the context changes. Right plot shows the wireless representative feature with the agents trained to adapt to the changes in distances

each (see Table 3.1 and 3.2). We located the mobile devices at different locations on a hallway indoors (see Fig. 3.5).

We collected a dataset based on computer vision tasks were we change the distribution od the tasks, and the distances among the Edge server and the mobile device. We ran a series of 200 tasks per experiment, tasks can be either image segmentation or image classification. The distributions of the tasks are based on the context change explained in section III.B. Our dataset has traces of 3 distributions of tasks for each context change in $CC = \{4, 8, 20\}$ and 3 different channel settings where the distance between the mobile device and the Edge server is 3, 6 and 12 mts respectively. To collect our dataset we ran a random agent that chooses actions from an action space made of 2 possible models, 2 possible type tasks and 3 potential processors.

3.10 Results

We investigate the tasks allocation problem for heterogeneous tasks in a multi-layered environment through a set of experiments that incorporate heterogeneity in the type of tasks, as

well as in the environment. We deployed our framework in two pairs of Edge Server - Mobile device configurations as shown in Table 3.1 and Table 3.2.

For the DDQN implementation, we use a sequential model with linear and convolutional layers with Relu activation functions. We have a state made of 43 input parameters where 11% belongs to the state of the mobile device, 18% are task related and the rest is part of the network monitoring.

We trained our agents based on three aspects: (1) the hardware configuration, (2) tasks distribution related to the frequency of context change and (3) the distance between the mobile device and the server which shows different channel settings. We consider two hardware configurations because one has better properties to perform multiprocessing which allows us to implement parallelism when multiple tasks are sent to different processors. We trained a total of 12 agents in an offline setting.

We evaluate our agents with the reward, accuracy achieved by the chosen actions and end-to-end task delay. We compared them with plausible policies such as CPU only, GPU only and EDGE only. The reward per batch of tasks dispatched reflects the priorities in the optimization as a weighted average. In our study we emphasised the end-to-end delay however we also consider the accuracy and power consumption as part of the weights, they are $[0.6, 0.3, 0.1]$ respectively. In Fig. 3.6 and Fig 3.7 we show the average end-to-end task delay for selected state features of each hardware setting. The % of CPU load, memory and power consumption are features of the state that capture the status of the mobile device while the % of the WLAN packets that are lost demonstrate the changes of the communication between the mobile device and the Edge Server.

Policies to allocate all tasks in one processor show a tendency that changes based on the metric that we use to measure performance. For instance all CPU ONLY policies tend to

converge to a delay the more context changes are in the distribution of tasks, to have more CPU usage while the all GPU ONLY policies tend to consume less power and more memory. On the other hand, EDGE ONLY policies show that the farther the Edge Server is, the more errors in the packets we find.

The trained agents for the NVIDIA JN hardware settings showed in table 3.1 (nano-cc-4, nano-cc-8 and nano-cc-20) achieve better delays when the context change is less than 20. The end-to-end delay improves from 0.9 seconds to 0.8 seconds when the context change is 8 and this gets higher for the distribution when the context changes 4 times. On the other hand, the agents trained to adapt to the changes in distance between the mobile device and the edge server (nano-dis-3, nano-dis-6, nano-dis-12) show a superior reduction of the end-to-end delays (60% of the time). The trained agents for the NVIDIA JXA hardware settings showed in table 3.2 (xavier-cc-4, xavier-cc-8, xavier-cc-20) only achieve an efficiency in delays for the scenario when the context change occurs 20 times in a sequence of tasks. However, some costs are reduced such as the % of CPU Load and Power Consumption for the case when the context changes 8 times. Additionally, the end-to-end delays are better (2.5x times) for each of the trained agents xavier-dis-3, xavier-dis-6, xavier-dis-12) compared to the EDGE ONLY policy.

In figures 3.6 and 3.7 we show the relationship between the state of the mobile device and one of the metrics used to optimize the trained agent (e.g. end-to-end delay). However, the actions taken from the trained agents to achieve such performance also show that different policies apply to different scenarios.

We show the ranking of actions used in each of the trained agents (see Fig. 3.8 and Fig. 3.9). The actions preferred in first place for variations in the context change for the NVIDIA JXA setting are a low resolution model in CPU for the image classification task with either high or low resolution model in EDGE for the segmentation task; which shows that GPU is

not preferred for this setting as a first option which seems reasonable due to the convergence in delay that occurs when the model is placed in the CPU instead of the GPU.

The policy chosen by the Xavier agents is similar when we change the hardware; in the NVIDIA JN setting the preferred action for the scenario with the least changes of context is the same of the one with the most changes of context (nano-cc-4 and nano-cc-20 respectively) is the one with high resolution model for the image classification tasks and low resolution model for the image segmentation tasks placed in the CPU. Only the nano-cc-8 agent first choice places the models in the GPU and the EDGE with low and high resolution respectively.

The actions preferred by the agents trained to adapt to the distance changes (e.g. nano-dis-3, nano-dis-6, nano-dis-12) in Xavier and Nano differ from each other. Xavier agents prefer actions that include the EDGE, for instance the second in the ranking of actions chosen by the Xavier agents (35,14,8) include dispatching all image classification tasks at the EDGE, where the high resolution is preferred in the closest distance and the low resolution is preferred when the distance between the mobile device and the Edge server increases. On the contrary, the second choices in the ranking of the Nano agents show a preference of dispatching tasks locally where the GPU is used at the same time as the CPU. For the actions 1, 24, and 6 the image segmentation tasks are always dispatched at the GPU. This empirically shows that the policy is adapting to the changes of hardware in each scenario as well as to the changes in the environment.

3.11 Discussion

In this chapter, we presented a framework for the control of task routing across the layers and computing units of edge computing systems. Different from most prior work, we resort

to the use of a real-world deployment to capture important effects, such as context switching, that greatly impact performance in systems characterized by streams of heterogeneous tasks. We train a DRL agent to control the allocation of tasks to the system devices and computing units in response to a complex definition of the state that incorporates features observable by the mobile device and demonstrate that the agent applies different policies to different hardware settings, task arrival settings, and other general features of the system.

A mobile device is equipped with processing units such as CPU and GPU as well as with wireless interfaces. Such device is required to perform inference either locally or by offloading the inference to a nearby Edge Server. We propose to optimize task allocation using the DDQN algorithm with a multi-objective function that considers the the end-to-end delay, energy cost in the mobile device and accuracy achieved with the selected model.

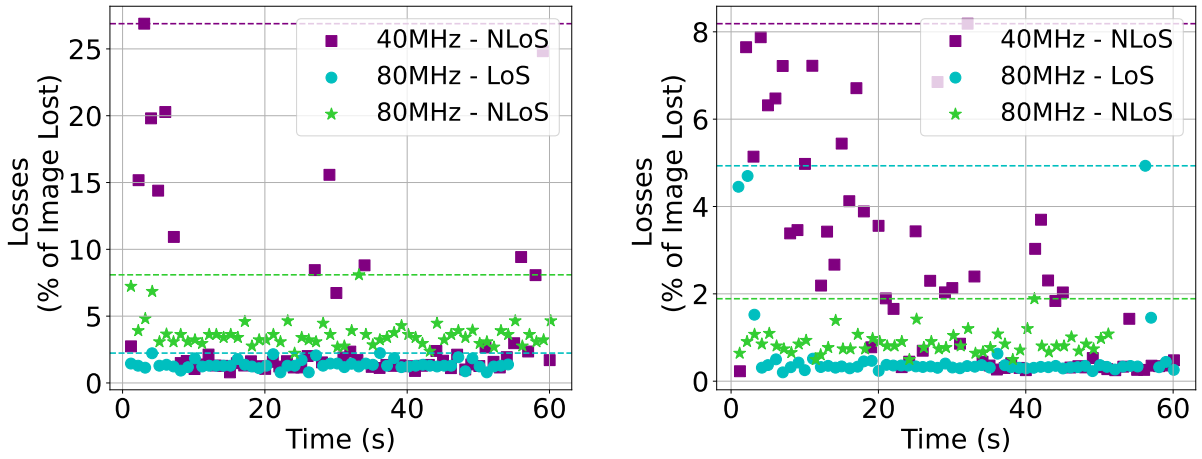
We consider on-device tasks generation with heterogeneous and computationally high-intensive which is common for complex applications such as AR. Therefore the selection between performing inference on-device or on the Edge server is non-stationary. We have formulated the tasks allocation optimization as a Deep RL problem. Deep RL algorithms are used to optimize sequence of decisions for high-dimensional states. This technique maximizes the expected reward of the sampled states given a sequence of decisions chosen. The interaction between the agent and the environment is trained to learn the MDP (Markov Decision Process) that characterizes our scenario.

Chapter 4

Cross-Layered-Communication Logic: Resilient Wireless Communication in MECS

4.1 Introduction

Future mobile applications are expected to rely heavily on machine learning algorithms for their operations, typically in complex deep neural networks (DNN). To make some examples, AR or autonomous navigation rely on computer vision tasks such as image segmentation, object detection, and classification, which are computation-intensive and delay-sensitive [3]. Due to the limited computing capabilities that are typically available to mobile devices (MED), in MECS the tasks are offloaded to a remote device, the ES [33, 48, 17]. Although offloading can decrease MED's energy consumption, as well as the execution time, the offloading process requires the transmission of the input data to the ES. In the case of computer vision, the input data often have a large size, and their timely and reliable transfer over



(a) 1x1 streams

(b) 4x4 streams

Figure 4.1: Packet loss using 40MHz and 80MHz within two outdoor scenarios: with and without obstacles.

wireless channels poses considerable challenges to communication and network systems.

The above issue is further exacerbated in applications where MEDs operate in challenging propagation environments, such as ground rovers autonomously processing visual information to autonomously navigate in urban and indoor settings, or airborne drones with extreme mobility characteristics [18]. In such scenarios, unreliable communications degrade core performance metrics, and make data rate erratic [47], consequently gravely affecting task performance. We note how these applications impose fine-grain performance constraints connected to individual data points (e.g., a bound on per-image latency or quality), rather than requirements on average performance metrics such as throughput.

Current literature adopts two main approaches in MECS to address this challenge. Both are based on limiting the amount of data to be transmitted from MED to the ES.

Approach one: partial offloading [33, 1, 89, 44, 11] is one of the approaches for maintaining the required QoS in challenging channel condition. Partial offloading involves dividing the data processing and task execution between ESs and the MEDs. In poor channel conditions, the MEDs choose to partially rather than fully offload the data to the ES.

Approach two: Some solutions propose to compress input data and or intermediate features [55, 46, 65], for instance in the context of "split" CNN architectures. This results in a smaller amount of data transmitted over the wireless channel, and thus a reduced sensitivity to poor or highly erratic channel conditions.

The above approaches focus on the modification of the computing pipeline, leaving the transmission layer unaltered. Conversely, many communication centered approaches focus on traditional traffic and network-based perspectives ignoring the important challenges and opportunities connected to the semantic structure of the data and computing tasks.

Focusing on modern MU-MIMO communication technologies, our rationale takes as a starting point fine grain characteristics of task-level performance. We first illustrate the characteristics of the problem at hand with some preliminary tests on a real-world system (see the detailed description in Section 4.4. Fig. 4.1 show patterns of per-image packet loss for different transmission configurations (number of parallel data streams and channel bandwidth) both in LoS and NLoS settings. The erratic nature of these fine-grain patterns is apparent, as well as their dependency on the specific environment and transmission configuration. On the one hand, larger resource usage – a larger number of streams and bandwidth – can reduce both average and variance of per-image PDR (as illustrated in Fig. 4.2). On the other hand, a larger amount of resources used by one user can impair the ability of the system to support a large number of users operating in the same region.

In the context of perception for robotic platforms, our overarching objective is to create a context- and task-aware predictive logic capable to dynamically and semantically control MU-MIMO resource usage to achieve task specific objectives. The proposed framework – SOAR – reasons at the granularity of individual data points to be delivered to the ES. Specifically, SOAR selects transmission configurations predicted to achieve a per-image packet loss – corresponding to a task specific performance – in a window of future images while minimizing resource usage based on observable features drawn from system components such as

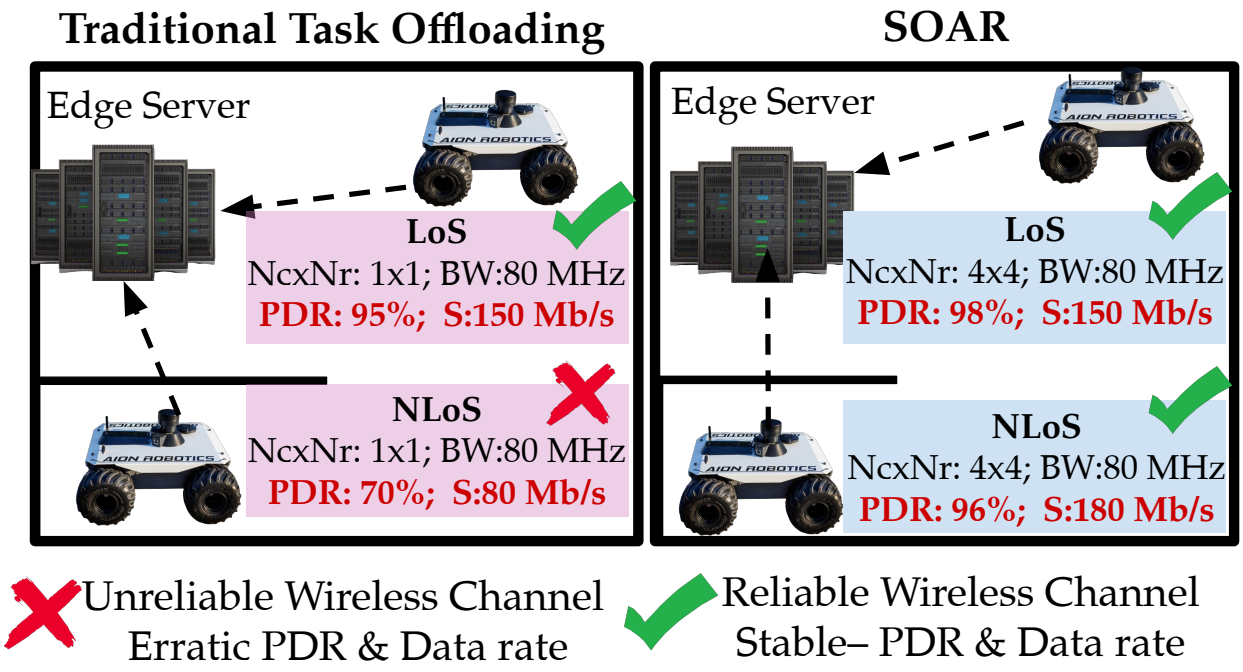


Figure 4.2: SOAR vs traditional task offloading approaches. Here, N_c: no. of the transmit antenna, N_r: no. of the receive stream, BW: bandwidth, PDR: packet delivery ratio, and S: data rate of the wireless channel.

telemetry, network interface and application.

The engine of SOAR is an innovative context-aware distributional deep reinforcement learning (DDRL) agent [22] that embeds a multi-branched neural network. A first neural selector processes general features to detect the context and select a context-specific model (branch) in a pre-trained set. The branch process a specialized set of features to output the distribution of a value function based on a composite performance/resource usage cost function. We remark how traditional DRL agents focus on the estimation of the expectation of a value function conditioned on an action, without consideration for its variance. In the context of mission-critical systems, this may lead to catastrophic outcomes. Conversely, DDRL focuses on the distribution of the value function given the action, thus allowing more sophisticated reasoning – together with more robust learning.

Summary of Novel Contributions

• In the context of robotic applications offloading computer vision tasks over MU-MIMO Wi-Fi (IEEE 802.11ac) channels, we perform an in-depth analysis of the relationship between system-level features of MECS from logical blocks such as application, network, and telemetry, the characteristics of image transfer at the fine-grain temporal scale, and task performance.

• We develop a novel semantic, predictive and context-aware controller – SOAR – that dynamically configures MU-MIMO transmission parameters to meet packet loss ratio objectives set at the granularity of individual image/task while minimizing channel resource usage. The controller is task aware, meaning that the packet loss threshold is set to the specific computer vision task. To make an example, as demonstrated in our results, a satisfactory performance in image classification can be achieved with a larger loss of information compared to semantic segmentation, where individual pixels are classified. The SOAR framework is context-specific as the controller and its input features are dynamically adapted to the operating context – e.g., a rover navigating in a line-of-sight (rural open space) or non line-of-sight (urban with buildings) environments – to boost control effectiveness.

• At the core of the SOAR framework, we explore the use of an innovative multi-branch distributional deep reinforcement learning agent. The core idea is to create context-specific agents, that process specialized set of features maximizing their performance in the environment their are trained for. A low-complexity neural selector analyzes a small set of features to extract the context and select the agent. We show that such an approach considerably improves the ability of the agents to predict patterns of per-image packet loss compared to an non-specific predictor. The agents take the form of distributional deep reinforcement learning algorithms, that evaluate the distribution of aggregate rewards rather than their expectation. This approach, which to the best of our knowledge is almost completely unex-

plored in communication systems, boosts the robustness of the controller.

• We perform an extensive data collection campaign to inform both the construction and the evaluation of SOAR with a Linux workstation as ES and a rover as MED in 2 different environments with 4 different communication setups. We consider data offloading at different FPS– 1, 5, 15, and 30 FPS for each of the setups. Additionally, we develop a framework to synchronously capture the features from network, application and MED telemetry blocks.

• We present results that evidence SOAR's framework context adaptation based on real-world dynamics represented by the application, network, and telemetry features. SOAR's DDRL context agent achieves a state-of-the-art level performance for two types of tasks: instance segmentation and object (image) classification, while reducing the resources utilization by 35% for NLoS and 40% LoS within an offloading deadline constraint of 20ms. We incorporated various strategic policies to out DDRL agent, including short-sighted (myopic) and long-term approaches, to assess the predictability of the LoS context relative to the NLoS context. Our findings indicate that the myopic policy yields superior performance in LoS context compared to NLoS context.

4.2 Related Work

As discussed earlier, task offloading in MEC systems suffers from the volatile nature of wireless channels, as well as time-varying workload patterns. To mitigate this issue, approaches explored in the literature include resource allocation frameworks, distributed computing solutions and the dynamic reconfiguration of computing pipelines between MED and ES. The reader is referred to excellent survey papers on this topic such as [70, 83, 40, 2]. In the literature, the task execution in ES is often referred to as full offloading, while distributing the task execution among MEDs and ESs is termed as partial offloading. One of the key

metrics to consider is the amount of data that needs to be transferred over the wireless channel, which is influenced by the computing component deployed at the mobile side, as well as application-dependent characteristics (e.g., input data size).

Most recent literature in MECS focuses on optimizing fundamental metrics such as energy consumption, bandwidth usage, and computational resources used in reliable channel conditions. To mention a few, Guo et al. [32]. Xiao et al. [85], and Fresa et al. [30] developed frameworks to maximize task accuracy in full offloading settings. Other contributions, such as Matsubara et al. [52], Lakew et al. [45], and Zhao et al. [92] explore partial offloading solutions to optimize the balance between end-to-end delay, energy consumption, and task performance using techniques such as split computing, early exit, and data compression [54].

To make a example, dynamic pipeline configuration has been studied in [13], where the authors propose to use a mixture of features such as telemetry, network, and application to predict the ESs that maximize the probability of successful task execution. The proposed approach consists of learning a task offloading policy based on a model-free Markov Decision Process that estimates the cardinal number of each ES that reliably performs the offloaded task. The impact of wireless channel conditions in MECS has been studied by [39], and [63]. Specifically, in [39], authors analyze how the noise in a LTE wireless communication system impacts the performance of offloading image segmentation tasks. Additionally, in [63], authors propose a denoising step that takes the transmitted image as input and returns a denoised image processed by the trained computer vision algorithm at the ES. The denoising step consists of a transformer-based multi-stage denoiser network trained for 5G communications noise with a UAV as the MED, improving the performance in object detection and segmentation within multiple changes in the 5G wireless communication parameters.

We then note how most task-aware algorithms mostly focus on the computing workload, while communication-oriented contributions are primarily centered on traditional traffic-based reasoning. In stark contrast, our contribution uniquely connects the semantic of

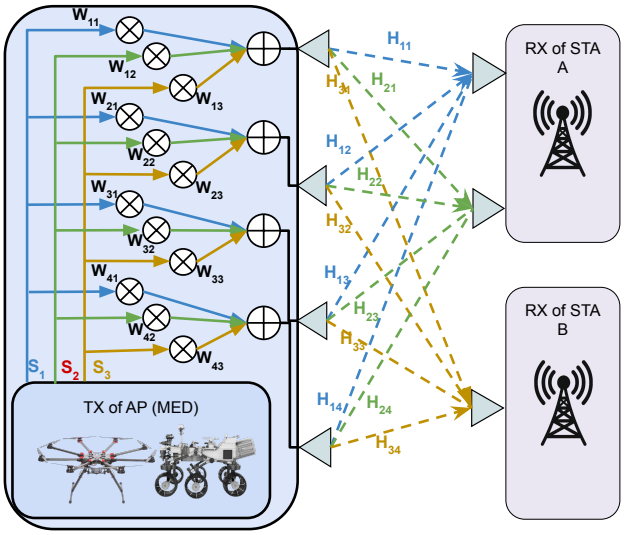


Figure 4.3: Example of a 4×3 MU-MIMO system.

the tasks and their fine-grain performance to the dynamic adaptation of communication parameters.

4.3 Preliminary Experiments: MU-MIMO in MECS

4.3.1 A Walkthrough of MU-MIMO Wi-Fi Systems

Wi-Fi is the technology of choice for plug-and-play connectivity [64]. As it is currently one of the most pervasive and widely adopted wireless technologies, it is also a viable choice for reliable task offloading [61, 5]. The AP integrated within MED enables MD to communicate simultaneously with multiple stations (STAs) attached to one or more ESs or to parallelize multiple streams to the same stations (STAs), reliably and efficiently leveraging beamforming through steering multiple transmission streams[34]. Wi-Fi adopts OFDM to allow transmission in K partially overlapping orthogonal sub-channels. The input bits are organized into groups–termed as OFDM samples, each group containing a certain number of bits. K of such OFDM samples a_k are grouped into OFDM symbols $a = [a_{-k/2}, ..., a_{(k/2)-1}]$ [37, 38].

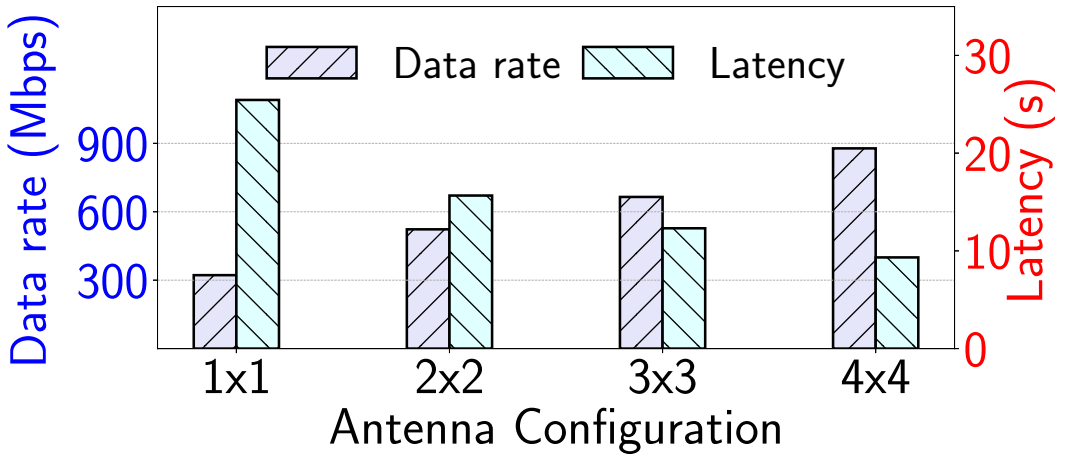
The digitally modulated symbols are simultaneously transmitted over K sub-channels.

$$s_{tx}(t) = e^{j2\pi f_c t} \sum_{-K/2}^{(K/2)-1} a_k e^{j2\pi kt/T} \tag{4.1}$$

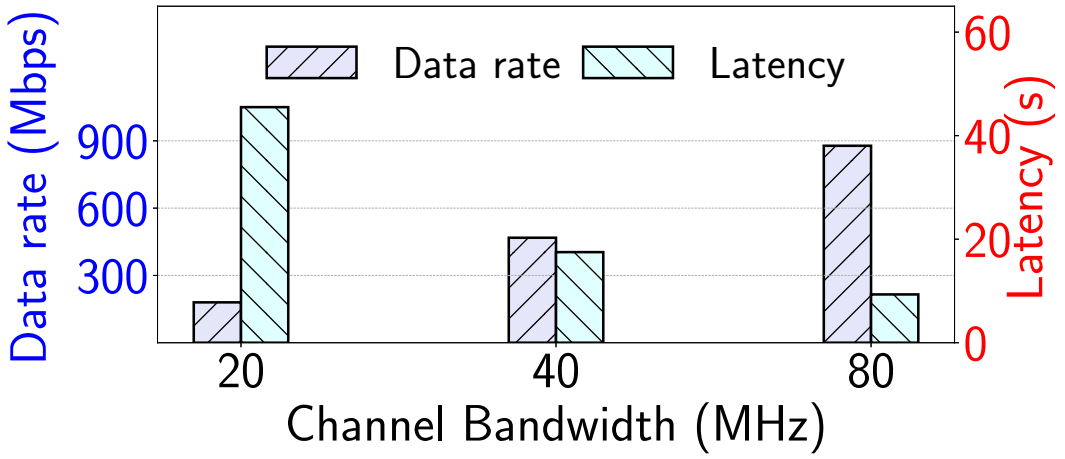
Equation (4.1) expresses the transmitted signal whereas f_c is the carrier frequency and $T = 1/(\Delta f)$ is the symbol time with Δf being the sub-channel spacing. To improve the signal quality, the transmitter performs beamforming to steer the transmission streams toward the intended receiver. To perform beamforming, multiple signal streams are combined at the transmitter by steering the weights $\mathbf{W}$. $\mathbf{W}$ is derived from the CFR matrix $\mathbf{H}$, which is estimated for every OFDM sub-channels. The obtained $\mathbf{H}$ is of dimension $\mathbf{K} \times \mathbf{M} \times \mathbf{N}$, where $\mathbf{M}$ and $\mathbf{N}$ are the number of transmit and receive antennas. At the receiver (beamformee), the beamformed signals are retrieved from the fact that $[\mathbf{H}]_{\bar{l},\bar{i}} \times [\mathbf{W}]_{l,i} = 0$ where $\bar{l} \neq l$ or $\bar{i} \neq i$. Fig. 4.3 presents a 4×3 MU-MIMO system where AP (beamformer) with 4 antennas transmitting to two different stations (STAs): STA A and STA B, having two and one receive antennas enabled respectively. The transmission signal $s_{tx}(t)$ from the beamformer bounces off different physical objects of the environment, and $\mathbf{P}$ different copies of the signal are received by the beamformees (STA A and STA B). If the signal is transmitted from $m \in \{0, 1, ...M-1\}$ antennas and received by $n \in \{0, 1, ...N-1\}$ antennas the CFR matrix $\mathbf{H}$ is represented equation 4.2 where $\mathbf{A_p}$ and $\tau_\mathbf{p}$ are the attenuation and delay experienced by path $\mathbf{P}$.

$$\begin{aligned} H_k(n) &= A_k(n) e^{j\phi k(n)} \\ &= \sum_{P=0}^{P-1} A_p(n) e^{-j2\pi(f_c + k/T)\tau_p(n)} \end{aligned} \tag{4.2}$$

The beamformer (AP) derives **W** from the **H** matrix to steer the transmission streams to



(a) Data rate and latency at different antenna configurations (80 MHz)



(b) Data rate and latency at different bandwidths (4×4 system)

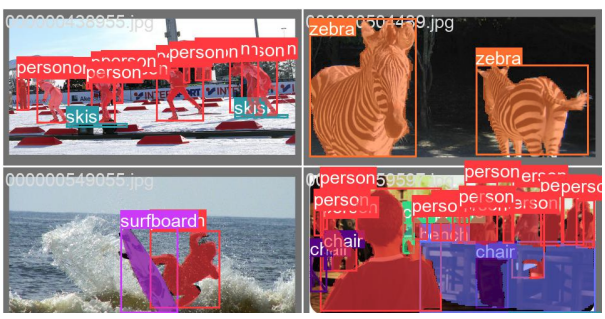
Figure 4.4: Data rate and latency at different antenna configurations and bandwidths.

enhance the power towards single or multiple beamformees (STA) simultaneously. This helps to improve the signal strength, resulting in improving the data rate and the transmission delay. Another benefit of MU-MIMO offloading is that the limited capacity of any individual streams does not degrade the overall network performance significantly.

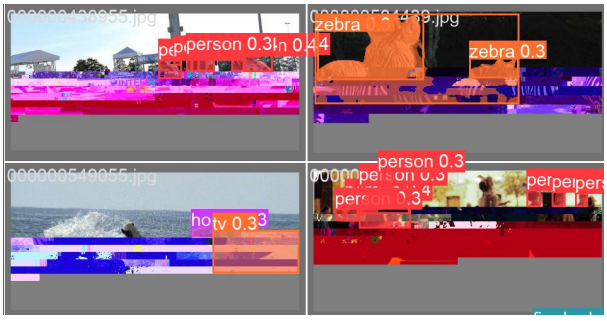
The overall link quality, including transmission data rate, packet drop, and transmission latency, depends on the number of transmissions and reception streams, available bandwidth,

and the number of stations (STAs) [88]. We present preliminary results showing variations in data rates and latency given different antenna configurations and channel bandwidths in Fig. 4.4. From the results, we can see higher data rates and lower latency with higher antenna configurations and bandwidth usage – which establishes an expected direct proportionality between resource usage and average performance. However, we note how average performance is just one aspect of the problem, as task offloading in mission-critical systems necessarily requires guarantees on individual data point (an image in our setting) latency.

Importantly, a higher number of transmission streams and larger bandwidth usage increase power consumption significantly, which causes the system to drain out faster, while also impairing the ability of the system to support multiple users [74]. Thus, it is essential to configure the MU-MIMO system to maintain task-level reliable communication links and optimize resource and energy consumption simultaneously. However, both **(i)** the, potentially extremely, erratic behavior of the channel in the context of robotics and vehicular applications, and **(ii)** the extreme requirements of such applications, it very challenging to find a transmission configuration that is effective. On the contrary, providing robust task offloading while minimizing resource usage requires predictively, dynamically, and opportunistically adapting transmission parameters at the level of individual tasks, in a task-aware fashion. This leads to the formulation of the control problem and resolution engine as detailed in the next sections.



(a) True labels and masks



(b) Predicted labels and masks

Figure 4.5: True labels and masks and Predicted labels and masks (YOLOv8) for datasets with 70% packet loss.

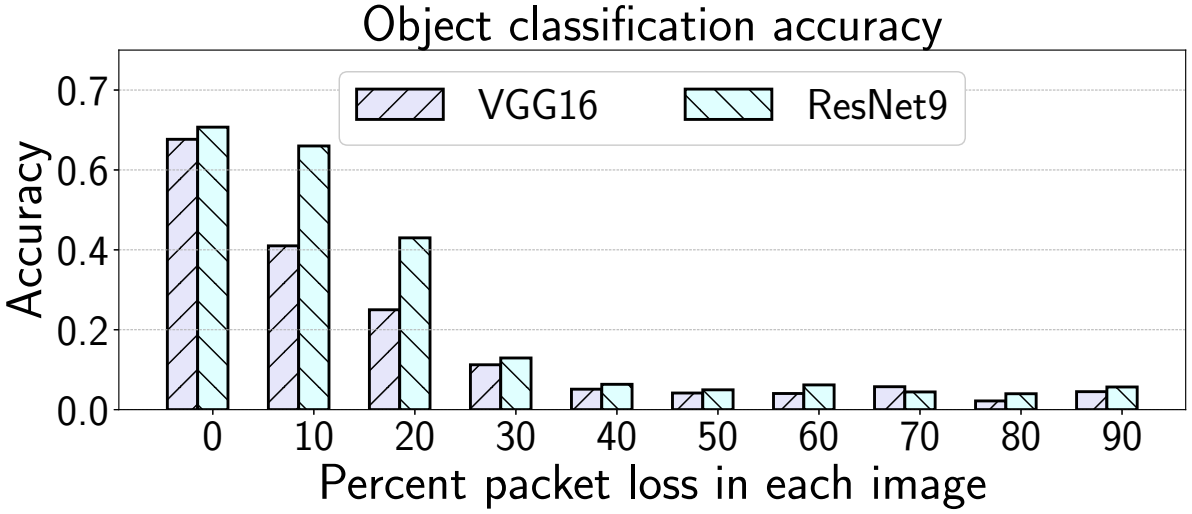


Figure 4.6: Object (Image) classification accuracy at different percentages of packet losses with CIFAR 100 dataset.

4.3.2 Computer Vision Task Performance with Packet Loss

We now demonstrate the how the relationship between task semantics, task performance and packet loss considering two popular ML based computer vision tasks– (i) object classification and (ii) instance segmentation. From the results, it is clear that due to the different semantic targets of the tasks and structure of the models the tolerable amount of packet loss per image is a function of the specific task, as well as of the neural model used. This consideration can be used to tune resource utilization to the task.

Object (Image) classification with packet loss

In object (or image) classification, a neural model categorizes an image into a particular label. A set of numerical features such as edges and corners are considered in the training phase to determine the performance of a class prediction. The accuracy measures the ratio of correct prediction where true positives and true negatives are considered in the numerator. In a well-balanced dataset, a score near 1 corresponds to a better performance. Fig. 4.6

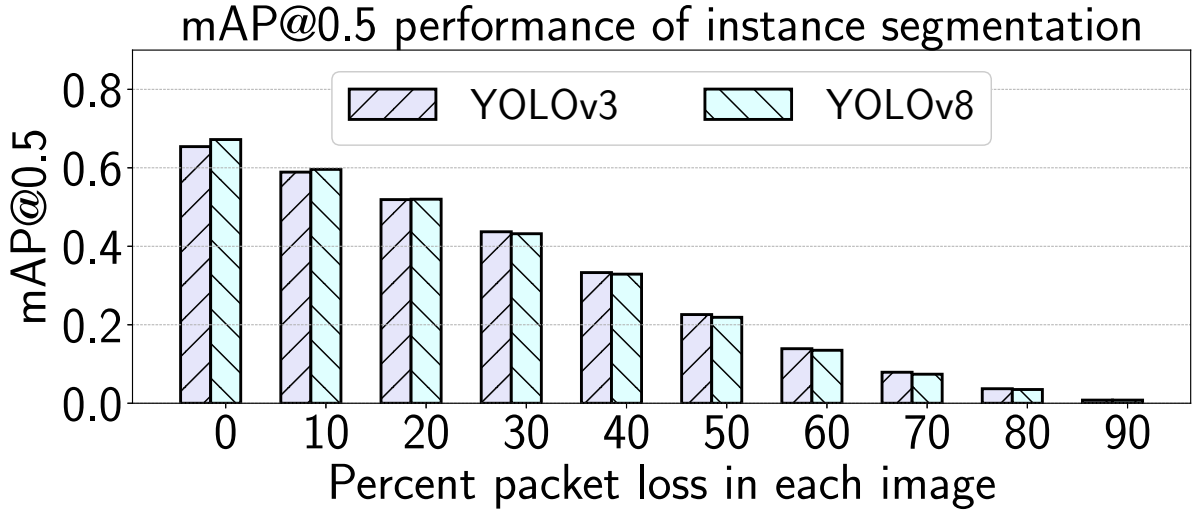


Figure 4.7: mAP@0.5 performance of instance segmentation at different percentages of packet losses.

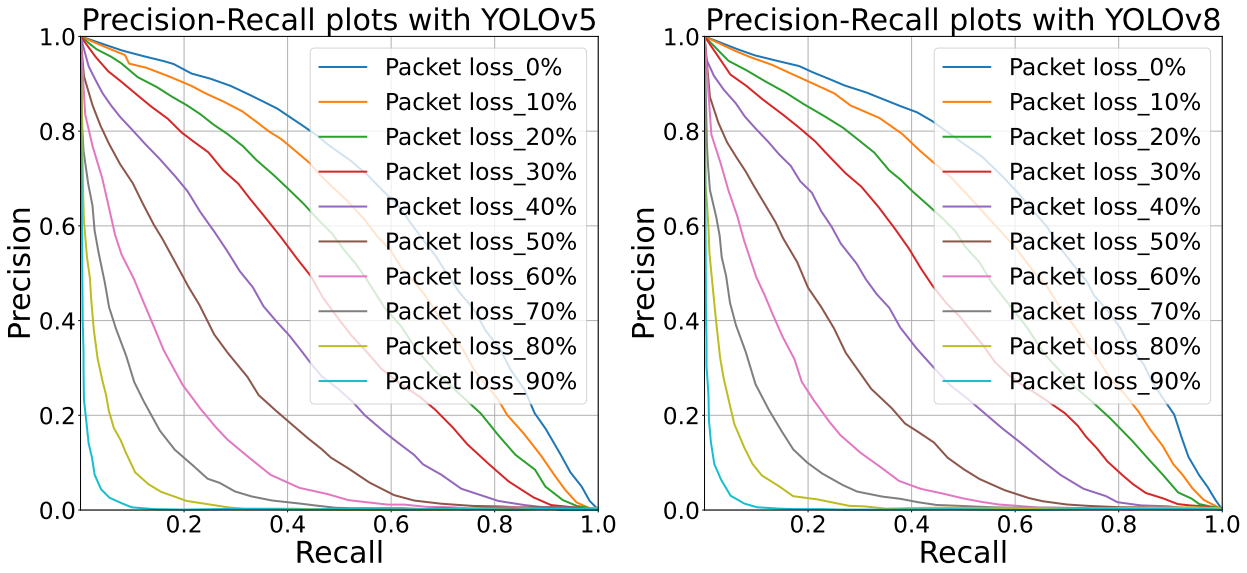
shows object classification accuracy obtained using the VGG16 [75] and ResNet9 [35] models trained and tested on the CIFAR 100 dataset [43] as a function of the packet loss affecting the input image. The performance of both models decays drastically when the percentage of packet loss in an image is above 30%. The VGG16 architecture [75] is composed of a uniform arrangement of convolution layers, this differs from the ResNet9 architecture [35], which is reflected in the performance variation, for instance, an image with 10% of loss downgrades the accuracy by half compared to the performance with no losses.

Instance segmentation with packet loss

In instance segmentation, a model identifies objects of interest and classifies each object's individual pixels. The intersection over the union (IoU) measures the extent of overlap between the predicted bounding boxes and the ground truth bounding boxes. An IoU of 0.5 indicates that there is at least a 50% overlap between the predicted, ground truth, bounding boxes and the pixel-level segmentation. mAP at 0.5 IoU (mAP@0.5) computes the AP for each class of the object and then takes the mean across all classes. AP accounts for both

precision (how many of the predicted objects are correct) and recall (how many of the actual objects were detected). A higher mAP@0.5 score indicates better performance, with scores approaching 1 indicating near-perfect localization and categorization. It reflects the model's ability to detect object boundaries and accurate pixel-wise labeling.

Fig. 4.7 and Fig. 4.8 depict the mAP@0.5 performance and PR curve respectively of instance segmentation with coco [49] validation dataset for different percentages of packet loss in every image with two SOTA models: YOLOv5 [42] and YOLOv8 [41]. mAP@0.5 performance is 0.67% when there is no packet loss in the images which degrades to 0.219% and 0.008% when the packet losses are 50% and 100% respectively with YOLOv8. Fig 4.5 shows the true label, true masks, and corresponding predicted labels (YOLOv8), predicted segmentation masks (YOLOv8) for a sample of datasets with 70% of packet loss respectively.



(a) YOLOv5 (b) YOLOv8

Figure 4.8: PR curve of instance segmentation for different percentages of packet loss with YOLOv5 and YOLOv8

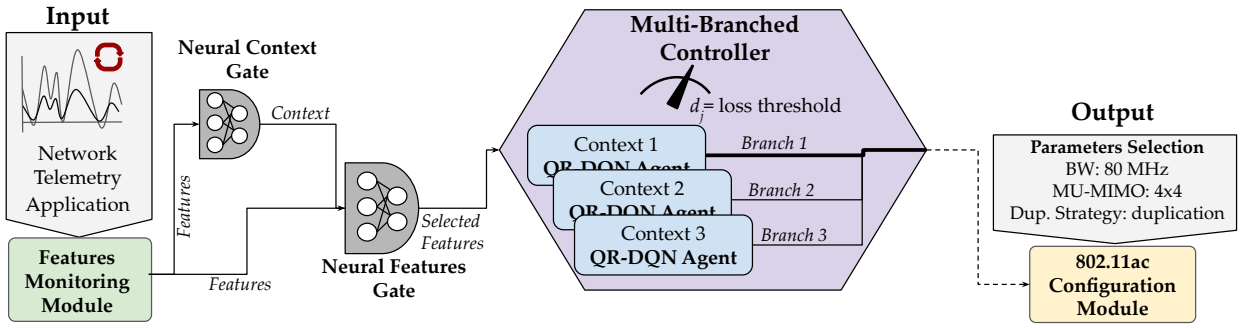


Figure 4.9: Dataflow of the SOAR system: We process a set of features from a diversity of context traces to optimize the wireless configuration based on the ML task requirements.

4.4 SOAR Task Offloading System

4.4.1 Problem Formulation

We consider a MECS composed of a MED and multiple ESs. The MED is connected to the ESs through a IEEE 802.11ac [37] MU-MIMO network, where AP and stations (STAs) correspond to the MED and ESs, respectively. We emulate a real-time task offloading system, where the image frames from benchmark datasets are offloaded from MED (AP) to the ESs (stations (STAs)) with a predefined frame rate. Our primary objective is to optimize wireless resource usage subject to deadline and task performance constraints. We denote the system control parameters as X_{b_i,p_i,r_i}, a vector composed of three key variables: the bandwidth $b_i \in B = \{x_1, \ldots, x_n\}$ MHz, the number of parallel streams in the antenna configuration $p_i \in A = \{1 \times 1, \ldots, n \times n\}$, and the packetization strategy used per device $r_i \in Y = \{y_1, \ldots, y_n\}$. We define two packetization strategies, they are duplication and parallelization; in duplication, a packet is duplicated as much as the number of streams in the configuration such that a 4×4 antenna configuration has 4 copies of a packet whereas parallelization implies that each stream transmits a unique packet. We define the task performance requirements (D_i) and map them to a target packet loss ratio per image as $D_i \in F = \{F_{type_1}, ..., F_{type_n}\}$. Each task has a latency constraint corresponding to a deadline T, which is the maximum time given to the system to complete the task and corresponds to

a maximum time to deliver the information to the ES.

We, then, denote the objective function as the sum of the cost of the control variables using the following resources consumption model adapted to MU-MIMO network configuration:

$$E_t = f_0 + f_1(b_t) + f_2(p_t) + f_3(r_t) \tag{4.3}$$

In the above equation, each f_i represents shape the raw resource usage to express the relevance of each component and the relationship between the increase usage of each component and the cost. We define the performance constraint as the probability of having the minimum number of packets d_j successfully received within the target time T. The probability M_j, defined as

$$M_j = P(d_j > D_j|\kappa), \tag{4.4}$$

is set based on application reliability demands.

We define our optimization problem as follows:

$$\begin{aligned} &\min \sum_{i}^{n} X_{b_i,p_i,r_i} \\ &\quad \text{s.t.} \\ &\qquad M_j \quad \forall j \in \{\text{type}_1, \ldots, \text{type}_n\} \\ &\qquad \kappa \leq T \\ &\qquad \kappa > 0 \\ &\qquad f_0, f_1, f_2, f_3 > 0 \end{aligned} \tag{4.5}$$

4.4.2 System Overview and Algorithm

To solve the problem defined in the previous section, we propose the semantic control framework SOAR. The SOAR framework is composed of four modules (see Fig. 4.9): (i) the Data Collector, (ii) the Context Detector (iii) the Feature Selector, (iii) the control agent.

The *Data Collector* module merges state-related data from the MED, including sensor readings, telemetry, wireless communication metrics, and application-specific features. The inputs for this model encompass telemetry data—such as accelerometer, gyroscope measurements, network metrics – including transmitter and receiver signal strengths, data rates, and application-specific parameters such as frames per second and task type. This data is normalized and partitioned into 1-second intervals to form the foundation for subsequent predictive analysis.

The *Context Detector* module takes the data coming from the Data Collector module and outputs the context. We consider each context as a class and train a classifier to identify the context from a subset of the raw data collected. The MLP model developed for this purpose is composed of 5 sequential layers with a ReLU activation function at the last layer to map to two different classes representing the contexts/scenario: LoS and NLoS in our testing as explained later.

In the *Feature Selector* module, a random forest algorithm is utilized to discern a context-specific subset of features that optimally predict future packet losses. The selection of these features is contingent upon the context– differentiating between LoS and NLoS scenarios, and is thus connected to the context detector module.

The *Control Agent* module integrates the contextual features and task requirements derived from the *Feature Selector*. This module is responsible for choosing a configuration based on the objective and constraints of the optimization problem. A specialized agent is trained for

different contexts and task performances, customizing a QR-DQN algorithm (see Algorithm 2).

Algorithm 2 SOAR Framework Algorithm

Input: $\sigma \rightarrow$ data, $\xi \rightarrow$ task requirements, q $\rightarrow$ quantiles
Output: $\chi_{b,p,r} \forall \{b \in B_{\phi,\xi}, p \in P_{\phi,\xi}, r \in R_{\phi,\xi}\} \rightarrow$ configs

```
1:
2: φ =ContextDetector(σ)                                  # Context Classifier
3:
4: ft = GetFeatures(φ)                                    # Feature Selector
5: m = Env(ft, φ, q, ξ, σ)                                # Initialize Environment
6: ag = load(φ, ξ)                                        # Load the trained agent
7: for e in episodes do
8:     s_t, _ ← reset(ag)
9:     while True do
10:        procedure PREDICT(ag)                          # QR DQN Prediction
11:
12:            Q(s_{t+1}, a_{t+1}) = Σ_j q_j θ_j(s_{t+1}, a_{t+1})
13:            a* ← argmax_{a_{t+1}} Q(s_t, a_{t+1})
14:            Tθ_j ← r + γθ_j(s_{t+1}, a*), ∀j
15:            return Σ_{i=1}^{q} E_j[H(Tθ_j − θ_i(s_t, a)]
16:
17:        end procedure
18:        s_{t+1}, r, done ← m.step(a*)
19:
20:    end while
21: end for
```

4.4.3 Distributional Deep Reinforcement Learning

We instantiate our controller as a Distributional Deep reinforcement learning agent whose reward combines the objective function and constraints of our optimization problem. As all reinforcement learning algorithms, our solution formally refers to Markov Decision Processes, defined by the tuple $(S, \mathcal{A}, R, P, \gamma)$, which are generally used to model complex environments and dynamically optimize decisions conditioned on a perceived state. $\mathcal{A}$ and S are the action and state spaces, R is the reward function associated with the state-action combination. $P(s_{t+1}|s_t, a_t)$ is the probability to transition from state s_{t+1} from the current state s_t and an

action chosen a. $\gamma \in (0, 1]$ is a discount factor that represents the impact of future rewards on the optimization. A policy $\pi(\cdot|s)$ associates each state $s \in S$ to a distribution over the action space A.

Although, standard RL algorithms successfully optimize the returns on policies defined as $G^\pi = \sum_{t=0}^{\infty} \gamma^t R_t$ – the sum of discounted rewards over a trajectory following the policy π; this approach leads to a general disregard of the rewards from states whose probability of occurrence is low, as well as completely neglect the variance of the long-term reward given an action. Moreover, directly learning the expectation of a long-term reward function generally requires a large number of samples.

Herein, we propose to adopt a distributional reasoning, and specifically use distributional RL algorithms, which explicitly consider the uncertainty in the long-term outcome of actions. This class of algorithms has been shown to improve the performance of the controller in many ways, from enabling faster learning to improving generalization. While DDRL has seen a limited investigation in the context of robotic navigation, its use in the optimization of wireless networks is almost completely unexplored.

We take as starting point the distributional DRL model-free algorithm called QR-DQN [22]. The QR-DQN algorithm is based on the DQN (Deep Q Network)[57], a standard DDRL algorithm where the expected return is the q-value of a state-action combination for an environment $\mathcal{E}$ and it is calculated as follows:

$$Q^*(s, a) = \mathop{\mathbb{E}}_{s_{t+1} \sim \mathcal{E}}[R_t + \gamma \max_{a+1} Q^*(s_{t+1}, a_{t+1})|s, a] \tag{4.6}$$

In QR-DQN, the q-value is expanded to a distribution of quantiles per action. The algorithm defines a quantile distribution Z that maps each state-action pair (s, a) to a uniform

probability distribution supported on $\theta_i(s,a)$ as follows:

$$Z_\theta(s,a) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\theta_i}(s,a) \tag{4.7}$$

where δ_z denotes a Dirac at $z \in \mathbb{R}$ Then, following this formulation, in [22] it is proven that given a quantile $\tau \in [0,1]$ the minimizer for $F_z^{-1}(\tau)$ is the quantile regression loss calculated as follows:

$$L^\tau(\theta) = \mathop{\mathbb{E}}_{\widetilde{Z}\sim Z}[\rho_\tau(\hat{Z}-\theta)], \text{where} \tag{4.8}$$

$$\rho_\tau(u) = u(\tau - \delta_{u<0})\forall u \in \mathbb{R} \tag{4.9}$$

where δ_u denotes a Dirac in u.

To solve our optimization problem, we define our action space $\mathcal{A} = B \times A \times Y$ as the number of possible wireless configurations for bandwidth, number of streams, and packetization strategy. Our state space S is defined by the context which represents a subset of the telemetry, network, and application features. The reward function is denoted as follows:

$$R = \begin{cases} \sum_{i=0}^{n}\lambda_i f(X_{b_t,p_tp,r_t}) + \lambda_{n+1}d_j + \frac{1}{1+e^x} & \text{if } d_j > D_j \\ \sum_{i=0}^{n}\lambda_i f(X_{b_t,p_tp,r_t}) + \lambda_{n+1}d_j & \text{otherwise} \end{cases} \tag{4.10}$$

Different from any existing DDRL algorithm, we train a set of context-specific agents, and use the context detector to select the agent, thus building the first multi-branched examplar of DDRL. Notably, each agent will use a constext-specific set of features as input to maximize its performance while minimizing its complexity.

4.5 Dataset and Setup

We build a dataset focused on monitoring the state of the system composed by MED and the ES within two different contexts for robotics applications. The dataset aims to provide insights into the correlation between the controllable parameters of the wireless channel such as the number of streams, the bandwidth used, the packetization strategy, and the inherent characteristics of the environment and application.

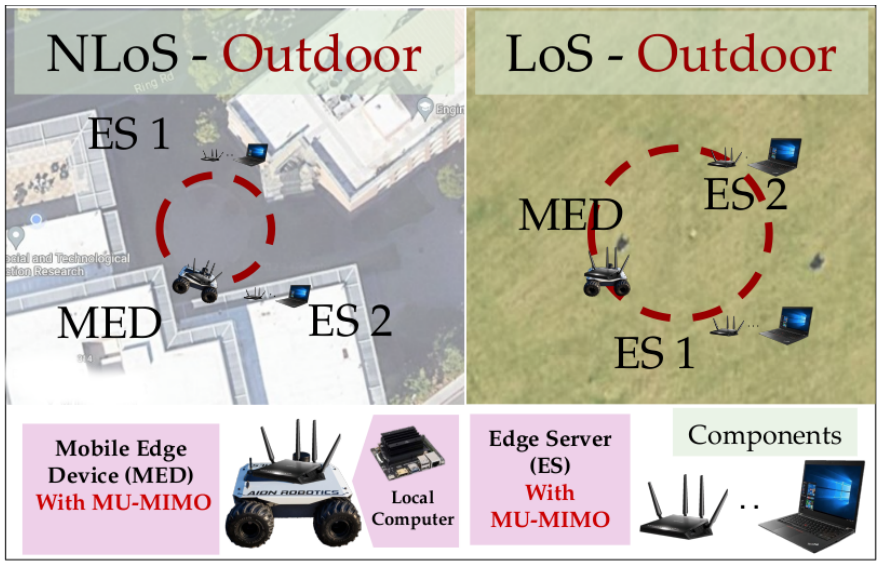


Figure 4.10: NLoS and LoS contexts for SOAR framework.

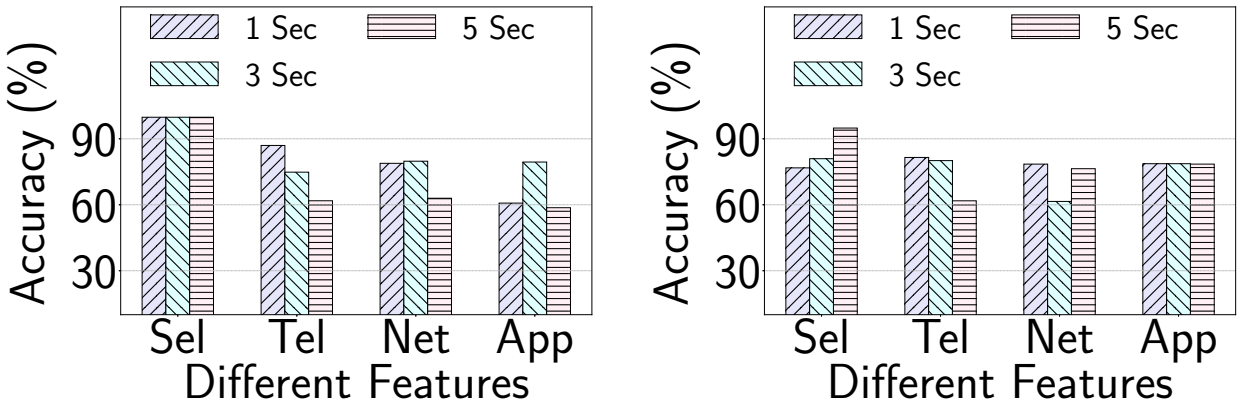
We collected features from two different scenarios: LoS and NLoS. For the LoS scenario, we have deployed our equipment in an isolated park area with no interference coming from infrastructure or other mobile devices. Alternatively, for the NLoS scenario, we use an urban environment with buildings and wireless infrastructure hindering packet transmission.

In both scenarios, the MED follows a circular trajectory (with a $\sim$ 12m radius, see Fig. 4.10). Each experiment is characterized by completing twice the same trajectory with specific configuration parameters, for instance, a fixed number of streams used to transmit data, a specific bandwidth, the transmitted frames per second, and the number of ES that the MED transmits its images.

We augmented our dataset through the implementation of multiple deadlines for the images, which correspond to different packet loss patterns. We drop all packets received after the deadline. We leverage the multiple streams of communication based on the packetization strategy to extend the dataset.

The hardware employed for the MED includes a custom-designed rover equipped with a Pixhawk [56] processor with Ardupilot as a ground control system. Within the rover's onboard control system, a Jetson Nano[16] microcomputer is deployed to oversee the execution of critical tasks, including telemetry management, network operations, and packet logging. To facilitate the telemetry logging process, we integrated the functionalities of the DroneKit library, which enables the recording of sensor measurements for subsequent analysis. The hardware used for the ES is a laptop with a GPU that could process the transmitted image at their end.

In the pursuit of performing 5G data transmission, particularly for utilizing MU-MIMO technology, we established a LAN infrastructure by interconnecting multiple NetGear routers. Each MED and ES has its NetGear router to transmit images. To modify wireless configuration for our experimental objectives, we installed a Linux-based operating system designed for embedded devices, known as OpenWRT which enables us to change the bandwidth used as well as the number of streams for the wireless transmission. Our network configuration is comprised of a mobile AP which is our MED attached to the rover, alongside two stationary stations (STAs) which are our ESs that transmit a rate of images per second using a non-reliable network protocol (UDP).



(a) LoS scenario

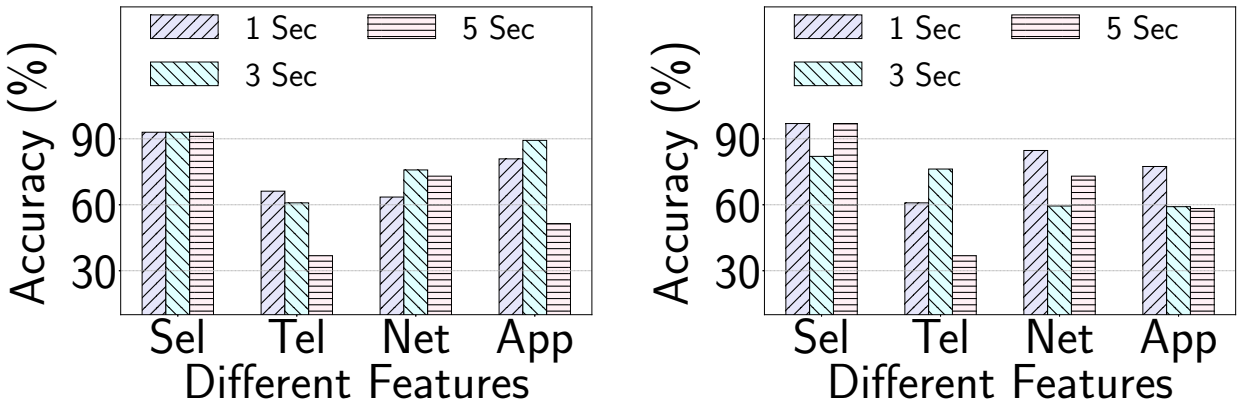
(b) NLoS scenario

Figure 4.11: DNN-filtered, telemetry, network and application features vs accuracy for segmentation task

4.6 Training and Evaluation

We implement custom machine learning models in multiple components of the SOAR framework, particularly in the *Feature Selector* and *Context Detector* modules. In the following, we describe the implementation details, the evaluation of our approach, and an analysis of the DDRL approached used in the SOAR algorithm.

To build the *Feature Selection* module, we train a random forest classifier to select the useful features as it is mentioned in section 4.4.2. Our tests show that the optimal set of features is heavily dependent on the context; for instance, in the LoS context: fps, X_{accel}, Y_{accel}, distance, and signal strength are the relevant features while in the NLoS context: X_{gyro}, Y_{gyro}, X_{accel}, Y_{accel}, distance and Rx bitrate perform better toward packet loss prediction. Additionally, we test the efficiency of the selected features compared to the network, application, and telemetry feature blocks used individually by implementing binary classifiers. These classifiers predict whether a percentage of packet loss exceeds a threshold within a time window (in seconds). Figures 4.11a - 4.12b show how the features selected by the random forest classifier outperform individual features for a packet loss prediction in windows



(a) LoS scenario

(b) NLoS scenario

Figure 4.12: DNN-filtered, telemetry, network and application features vs accuracy for classification task

of 1, 3, and 5 seconds respectively.

In the *Context Detector* module, we train a sequential 5-layer model with ReLU and Softmax activation functions and binary cross entropy loss to identify the two contexts based on our dataset number of contexts. We use 10% of our data and a 5 cross-validation split to avoid overfitting, after training we achieve 91.43% of accuracy.

We implement a DDRL-based algorithm to solve the optimal wireless configuration dynamically over a MED trajectory considering the erratic channel behavior observed in a real-world environment. We developed our environment to support the selected features per context as well as the possible actions based on the real-world wireless configurations available in our hardware setup. Concretely, the deployment of the QR-DQN agent was using Stable-Baselines 3 framework [67]. We train an agent per context with 100 quantiles as a tuned parameter. We use our collected dataset as the state space, and showcase the performance of the agents for two computer vision tasks relevant to robotics applications: instance segmentation and image classification. As shown in section 4.3 the performance of a task is downgraded as a function of a portion of the image received or the packet loss. Similarly, the wireless configuration and the application deadline requirement play a role in the packet

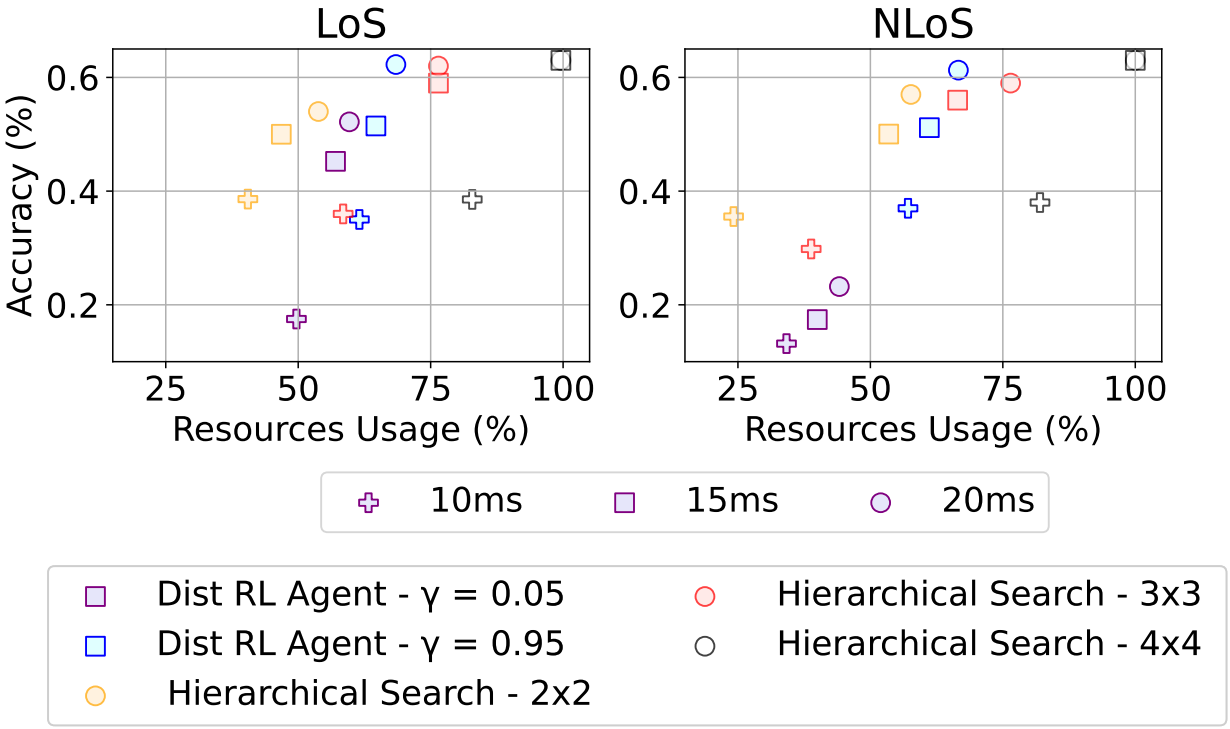


Figure 4.13: Image Classification performance within LoS and NLoS contexts

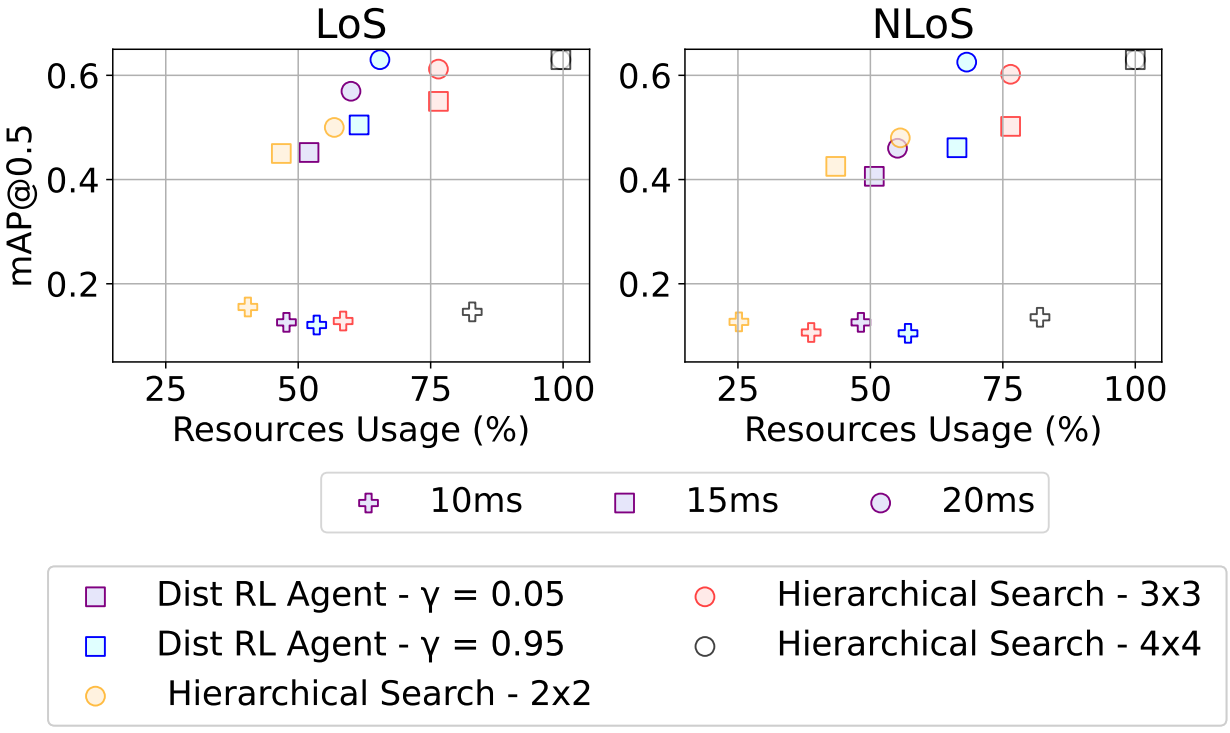


Figure 4.14: Instance Segmentation performance within LoS and NLoS contexts

loss.

As part of the evaluation of the DDRL agents, we train agents for two types of strategies: myopic and long-term. These policies are differentiated by the values assigned to the discount factor γ. The myopic policy corresponds to $\gamma = 0.05$, indicating a short-sighted approach where immediate rewards are prioritized over future gains. Contrarily, a higher γ value of 0.95 emphasizes future rewards. To evaluate the performance of these strategies, we implement a hierarchical search algorithm, which assumes that the antenna configuration is fixed, e.g. 2×2 within the trajectory followed by the MED. The search finds the minimum amount of resources (determined by the bandwidth and packetization strategy) that the MED can use to get the minimal number of packet loss per second. We repeat the experiment for 2×2, 3×3, and 4×4 antenna configurations and 3 different deadline requirements: 10, 15 and 20 ms. The resource usage calculation is based on the components of the possible wireless configuration which are the bandwidth, number of streams, and packetization strategy. Each component spends a percentage of the resources, in this way, the maximum resources expenditure in a wireless configuration is done by the 4×4 antenna configuration, 80 MHz bandwidth, and duplication as packetization strategy. From these variables the prioritization goes as the hierarchical search, being the antenna configuration the parameter that is more influential in the resource utilization, followed by the bandwidth and the packetization.

The classification task results for LoS and NLoS are shown in Fig. 4.13. We calculate the accuracy obtained from the relationship between the packet loss and the performance observed in 4.6 for the ResNet9 [35] model. From Fig. 4.13 we notice that within a 20 ms deadline, 3×3 and 4×4 fixed wireless configurations achieve the maximum accuracy of the ResNet9 [35] model in both contexts with their respective costs in the resources usage when the hierarchical search is implemented. Regarding the strategies of each agent, the long-term policy outperforms the accuracy of the myopic policy for both contexts while the latter maintains a lower resource usage. The myopic policies compared among contexts

demonstrated a better performance in the LoS context – our intuition is that this is due to the lower complexity environment compared to the NLoS. Additionally, we observe that the LoS agent reaches its best performance using 35% less resources than the fixed wireless configuration, and the NLoS agent reaches its best performance using 40% less resources than the fixed configuration.

Segmentation task results for LoS and NLoS are shown in Fig 4.14. We calculate the performance of the mAP@0.5 given the packet loss based on the numbers obtained in Fig. 4.7, specifically using the function for the YOLOv8 [41] model. Further, in Fig. 4.14 the maximum accuracy of the YOLOv8 [41] model is reached – similarly to the classification task – within the 20 ms deadline and using a 3 and 4 streams fixed configuration. Contrary to the classification task, the myopic policy achieves a better mAP as the performance degrades gradually with the packet loss compared to the classification task. We also observe that the mAP remains above 0.5 for the 15ms deadline. Resources usage is reduced in both contexts, moreover for a certain number of resources 60%, SOAR can use smaller deadlines (thus decrasing the overall latency) without drastically degrading the performance of the tasks.

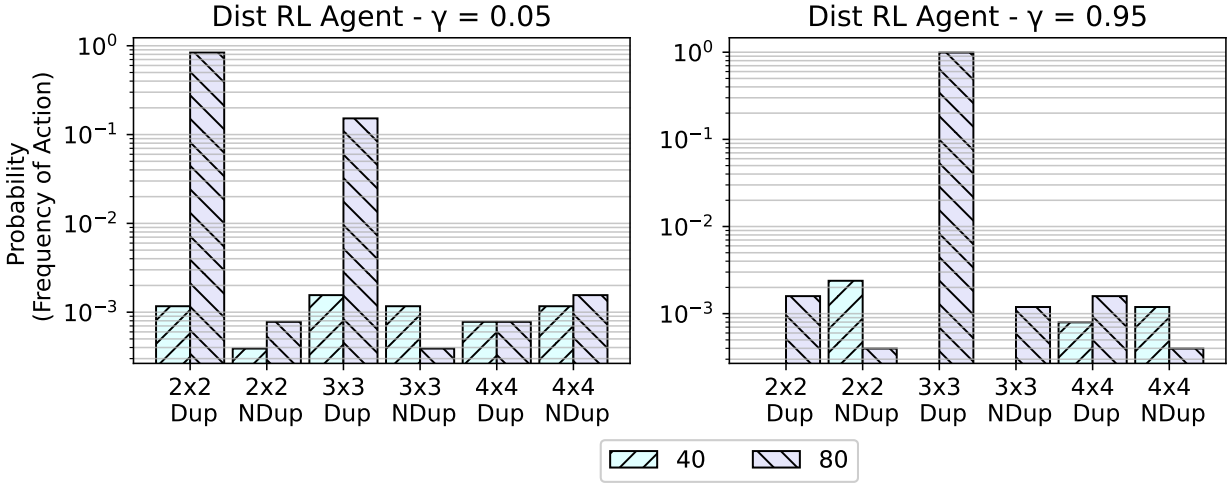


Figure 4.15: Action distribution for 20ms for different γ parameters in NLoS context. Dup and NoDup stand for each packetization strategy implemented

Our QR-DQN algorithm produces policies that optimize the reward distribution; this is

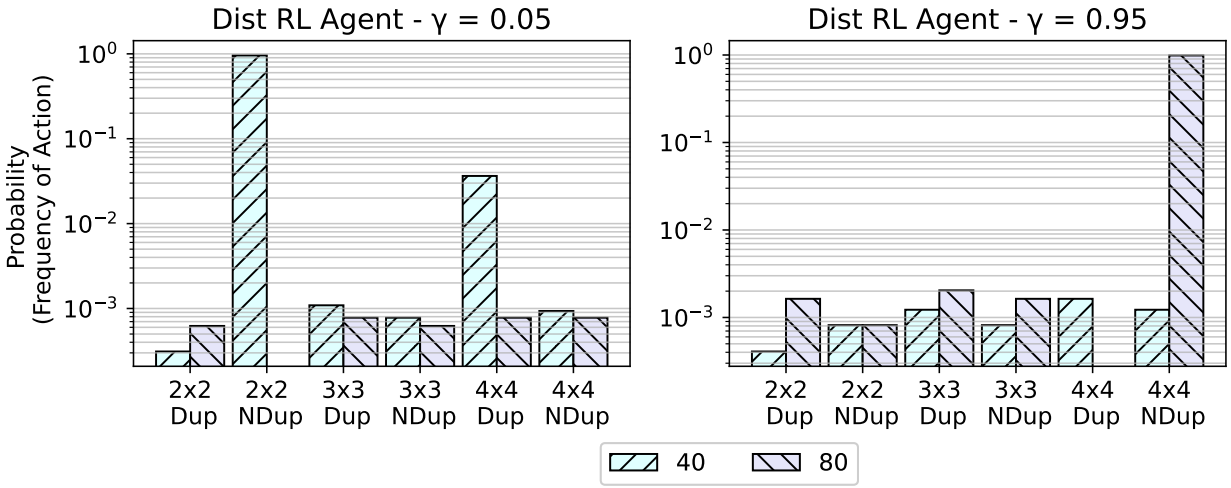


Figure 4.16: Action distribution for 20ms for different γ parameters in LoS context. Dup and NoDup stand for each packetization strategy implemented

reflected in the distribution of the action. We run experiments of 500 episodes to obtain Fig. 4.16 and 4.15 that illustrate the action distribution of the DDRL agents per context. In both contexts, the myopic policy produces a broader range of actions, the duplication of packets is used more often in the NLoS scenario which implies an increased resource consumption in this context. This aligns with the increased complexity associated with the NLoS context as compared to the LoS one. In terms of bandwidth preferences, there is a clear distinction: the 80 MHz bandwidth is predominantly selected in NLoS contexts, whereas the 40 MHz bandwidth is more frequently utilized in LoS situations.

4.7 Discussion

Our research delved into the task offloading problem within the scope of implementing robotic applications where the mobility of the MED, the communication channel, and the ML tasks requirements in real-time play an important role. The primary objective of our study was to provide reliable communications through understanding the semantics among contexts in the real-world to minimize the resource utilization associated with the parameters of the

multiple-input multiple-output wireless configurations for specialized tasks.

To address this challenge, we introduced the SOAR framework, which is a reliable context-aware wireless configuration solution tailored for resource optimization in MECS. To rigorously assess the performance of SOAR, we assembled a comprehensive dataset comprising transmitted images through a MU-MIMO channel within a context (NLoS and LoS) trajectory. This dataset was generated using a lightweight, non-reliable data protocol (UDP), and it encompassed various features including application, telemetry, and wireless configuration.

Our analysis involved the implementation and development of multiple data gates and modules to extract features with predictive capabilities, and to identify the context to which the features belong. Furthermore, we extended our dataset by incorporating diverse packetization strategies and imposing task deadlines feasible in the real-time robotic applications realm.

The results demonstrated that SOAR can identify contexts using real-world data through feature selection and context detector gate implemented within its design. SOAR's DDRL context agent maximizes the reward distribution in a horizon which is reflected in the adaptability of its policy to the tasks performance while reducing the resources utilization over the MED trajectory. Specifically, SOAR achieves a state-of-the-art level of task performance using 35-40% of resources when compared to a fixed MU-MIMO wireless configuration optimization for an offloading deadline of 20ms within both contexts.

Chapter 5

Higher-Layered-Computing Logic: Adaptive Inference for Modular Pipelines in MECS

5.1 Introduction

The proliferation of machine-type communications (MTC) has been driven by the advancement of AI in areas such as connected vehicles [81], industrial automation [6], smart cities [4], and agriculture [60]. Traditionally, the applications for MTC like object detection, segmentation, etc., have been executed locally on end devices through local inference. It [84] offers real-time response, but also requires high computational requirements and increased power consumption, leading to higher capital expenditure (CAPEX) and reduced device runtime. This may not be scalable in size, weight, power, and cost (SWaP-C) constrained environments, such as factories, defense, and aerospace. Recent studies have explored remote inferencing/full offloading [15] to address these drawbacks, where resource-constrained

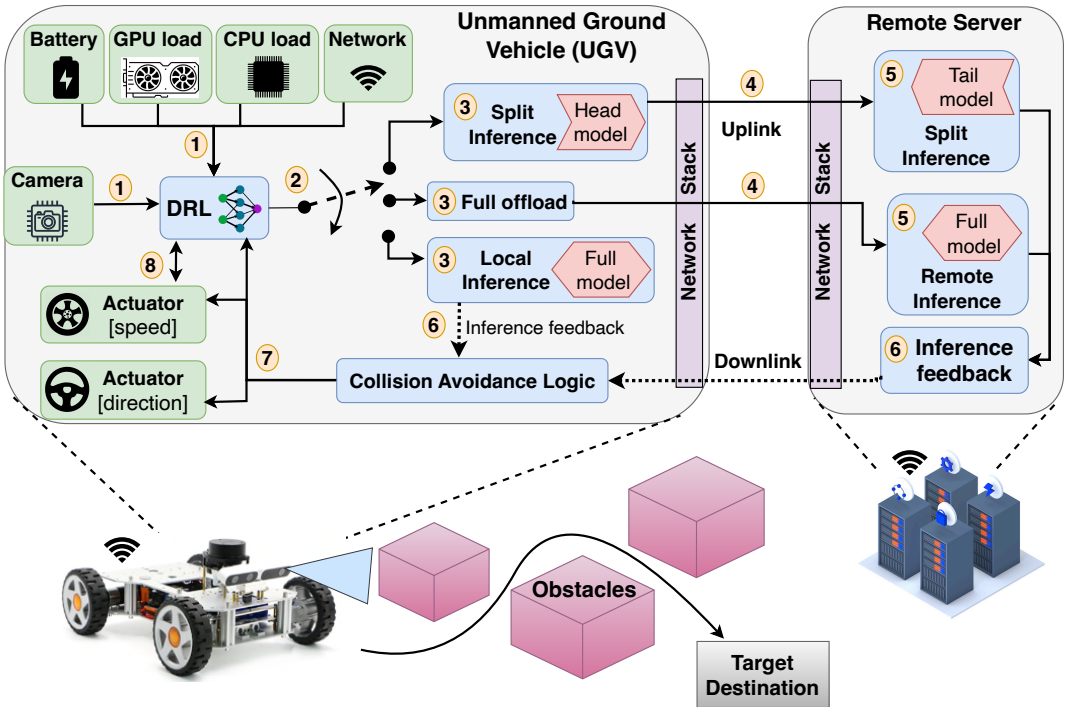


Figure 5.1: The CAMTEC framework utilizes the UGV system information to learn the context of the machine task performance. It then orchestrates the selection of the optimal inferencing method (local, remote, or split), and UGV velocity for collision-free navigation.

devices offload sensor data to remote servers equipped with high computational capabilities for inference operations. However, this requires high bandwidth utilization, hindering scalability in limited spectrum scenarios and potentially increasing CAPEX for access to a larger spectrum. Split inference [51, 8, 53] has been proposed as an alternative solution aiming to achieve a middle ground between local and remote inferencing by partitioning the neural network model and executing different parts on the end device (the head model) and the server (the tail model). Split inferencing can be beneficial for autonomous devices constrained by

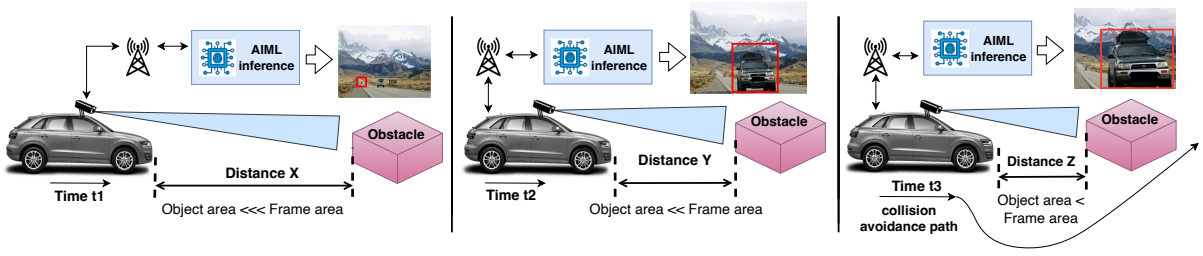


Figure 5.2: Example scenario for quantifying the vision-based collision avoidance (CA) quality of service threshold in terms of identified object-to-image frame area ratio

SWaP-C in dynamic network conditions, since it requires lower computational complexity on the end devices compared to local inferencing, and lower bandwidth compared to full offloading.

• **Motivation and novelty:** Existing MTC applications [87] lack the intelligent logic that understands the effect on the machine task performance of robots and UGVs due to local/remote/split inferencing decision in dynamic network conditions. Our work addresses these drawbacks by proposing the Context-aware Adaptive Inference for Machine Task Execution in Complex Environments (CAMTEC). As shown in Fig. 5.1, CAMTEC orchestrates the activation of the most beneficial inferencing method (local/remote/split) for any off-the-shelf ML model for jobs such as object detection and recommends to any off-the-shelf object detection based navigation controller, the UGV speed to ensure a zero collision trajectory and maintain the UGV navigation (machine task) precision. To enable this orchestration, CAMTEC learns the effect of wireless network, computation, and energy on machine task precision from the context of the environment conditions, network performance, the available compute and energy thresholds, and the ML model precision (e.g., inference confidence level).

• **Proposed Work:** In the considered scenario, the camera-equipped UGV with SWaP-C constraints runs CAMTEC on top of off-the-shelf object detection AI/ML models and navigation controller algorithms, to execute vision-based obstacle avoidance and reach a target destination in dynamic network conditions. While capable of performing some level of local inference for immediate tasks, the UGV strategically offloads computationally demanding AI/ML tasks to a nearby edge server based on CAMTEC's understanding of the environment, compute, and wireless performance. This approach balances the need for real-time responsiveness with the ability to handle complex computations. The edge server, upon receiving the sensor/tensor data, executes remote inference or collaborates with the UGV on split inference tasks and returns crucial inference feedback. This feedback influences the

vehicle navigation controller, which, in conjunction with CAMTEC, determines the optimal speed of the vehicle to safely navigate obstacles and reach its target destination. This ensures that the UGV optimizes resource utilization and adapts to changing environmental and/or operational demands to successfully execute a machine task in constrained environments.

The main contributions of this chapter are as follows:

- We implement an off-the-shelf machine learning model (YOLOx) trained to do vision-based object detection and classification for local, remote, and split inferencing on a commercial off-the-shelf (COTS) UGV and remote server, for navigation in a cluttered environment as the machine task. Through empirical experiments we showcase the performance limitations of local, remote and split inferencing, individually.
- We design and implement a DRL-based adaptive inference and UGV speed controller process at the heart of CAMTEC, using multi-modal sensor data from the UGV. For this purpose, we modify and use a Proximal Policy Optimization (PPO) agent that learns from the compute, energy, network, and the YOLOx inference performance metrics and generates the updated decision for inference modality in real-time, along with the updated speed of the UGV, for the execution of the machine task with desired precision. We set the machine task precision threshold as zero collisions during navigation.
- We explain the building blocks of CAMTEC, and through extensive experiments we showcase its performance in a cluttered environment in dynamic network and UGV system conditions. We show how our proposed approach ensures the machine task precision through better resource management, compared to the baseline ideal scenarios.

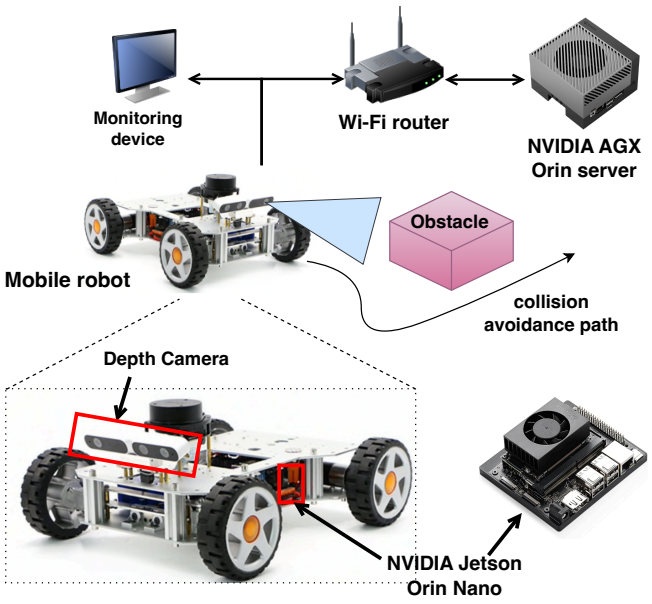


Figure 5.3: The UGV equipped with a stereo depth camera sensor and NVIDIA Jetson Orin Nano local compute module interacts with the NVIDIA AGX Orin remote server via a WiFi link.

5.2 Problem Statement and Optimization

5.2.1 Problem Statement

We consider a scenario where the UGV follows a predefined trajectory that includes obstacles along its path. In this context, a collision is defined as the point at which the distance between the UGV and an obstacle approaches zero. The proposed vision-based navigation system estimates obstacle distance by leveraging the inverse relationship between distance and bounding box size, conditioned on the obstacle's class. A potential collision, denoted by φ, is defined to occur when the condition $|p_i - th| < d_o$ is satisfied, where p_i represents the object-to-frame area ratio of the i-th object, th is the minimum area threshold for collision avoidance, and d_o is the minimum detection distance required to ensure safe navigation.

To optimize the overall performance of the navigation to a target destination, we propose a multi-objective cost function that drives the policy of the DRL algorithm by incorporating

trajectory speed optimization and collision avoidance while considering device and network resource constraints. The cost function and its constraints are defined as follows:

$$\begin{aligned} \min \quad & \left[\alpha \left| S_n - \frac{1}{R} \sum_{r=0}^{R} S_r \right|^2 + \beta \left| p_i - th \right|^2 \right] \\ \text{subject to} \quad & |p_i - th| < d_o, \\ & \eta_{cpu} < C_{MAX}, \eta_{gpu} < G_{MAX}, \\ & \kappa < B, \zeta < E. \end{aligned} \tag{5.1}$$

where $i \in C$, the set of obstacle classes. S_n represents the target average speed of the robot at interval n, and $\frac{1}{R}\sum_{r=0}^{R} S_r$ is the average speed observed on the trajectory up to time T. The coefficient α determines the importance of minimizing deviations from the target average speed. The second term quantifies the collision risk by measuring the difference between the object-to-frame ratio p_i of a detected object and the collision threshold th, with β acting as a weighting factor for this risk. The constraints represent the operational capacity of the system during the execution of navigation tasks, providing a framework to ensure efficiency and safety. For example, $|p_i - th| < d_o$ ensures that object-to-frame ratio p_i, associated with an object of class $i \in C$ (the set of all obstacle classes), remains within a safe threshold distance d_o to avoid collision. The CPU utilization η_{cpu} must remain below the maximum allowable limit C_{MAX} to maintain system performance. Similarly, GPU utilization, η_{gpu}, must not exceed G_{MAX}. The total bandwidth usage κ must not exceed the available network capacity B, and the energy consumption ζ is constrained by the device energy capacity E.

5.2.2 Deep Reinforcement Learning Optimization

We propose a DRL framework that interfaces with off-the-shelf object detection and navigation algorithms for adaptive local/remote/split inferencing and speed control. This frame-

work employs Proximal Policy Optimization (PPO) algorithm [71] for sample-efficient and stable policy updates. PPO improves upon Trust Region Policy Optimization (TRPO) by constraining policy shifts using a clipping mechanism, ensuring balanced exploration and stability. The DRL environment is composed of the state space $\mathcal{S}$, which represents key system metrics such as device battery level, CPU and GPU utilization due to inference or navigation processes, available network bandwidth, and server workload. The action space, $\mathcal{A}$, is defined by the possible speeds and inference modes (local, remote, and split). The PPO agent optimally learns a policy $\pi(a|s))$ to dynamically select actions based on the system's current state. The reward function $r(s, a)$is formulated to incentivize optimal system behavior by balancing application performance metrics while accounting for system state dynamics, including bandwidth constraints. It integrates the parameter f, which measures the navigation performance, demonstrating the deviation of the current speed and threshold from their respective targets, scaled by weights α and β, as in Equation 5.1. Our reward function is computed as:

$$R = \begin{cases} -1, \text{ if constraints in eq. 5.1 are not met,} \\ w_f \cdot f + w_e \cdot \Delta_E + w_b \cdot \Delta_B + \dfrac{w_l}{\text{lat}}, \text{ otherwise,} \end{cases} \tag{5.2}$$

where w_f, w_e, w_b, and w_l are configurable weights for the deviation penalty, energy consumption, bandwidth optimization, and latency minimization, respectively. This flexible design allows the agent to prioritize different objectives based on the environment's requirements. Incorporating these weights alongside the optimization in Equation 5.1 facilitates convergence toward feasible states constrained by our dataset, thereby improving convergence to optimal performance. This approach is generalized for any reinforcement learning environment, enabling consistent application across diverse domains without environment-specific modifications. We utilized the PPO implementation from Stable Baselines3 [67] to fine-tune our optimization problem and developed a custom environment compatible with the OpenAI Gym interface.

5.3 Preliminary Experiments

5.3.1 Machine Task Precision (MTP)

For autonomous navigation in cluttered environments, we quantify MTP via the object-to-image frame area ratio, denoted as p and illustrated in Fig. 5.2. This metric is calculated as the ratio between the detected object's bounding box area and the total image frame area (width $\times$ height). The parameter p serves as a normalized, scale-invariant threshold for vision-based collision avoidance, where higher values correspond to increased proximity and collision risk. This normalization ensures consistent performance evaluation across diverse camera resolutions and object dimensions while maintaining computational efficiency for real-time navigation applications requiring immediate collision risk mitigation.

5.3.2 Experiment Setup

The testbed shown in Fig. 5.3 comprises four main COTS components: a Rosbot PRO UGV [68] equipped with a camera, an NVIDIA Jetson AGX Orin [62] Edge Server, a dual-band configurable WiFi router [79] for connecting the devices, and a remote monitoring system. For local computation, the UGV is equipped with an NVIDIA Jetson Orin Nano processing unit as local processor, a Wi-Fi Adapter Wireless Dual Band 5 GHz 867 Mbps + 2.4 GHz 300 Mbps to transmit/receive data, a stereo camera. The UGV also includes an STM32F103RC microcontroller for UGV control, powered by a 22.2 V 5000 mAh battery.

5.3.3 Split Computing Models in Constrained Devices

Object detection models produce three primary outputs: bounding boxes for spatial localization, confidence scores indicating detection reliability, and class labels for categorization.

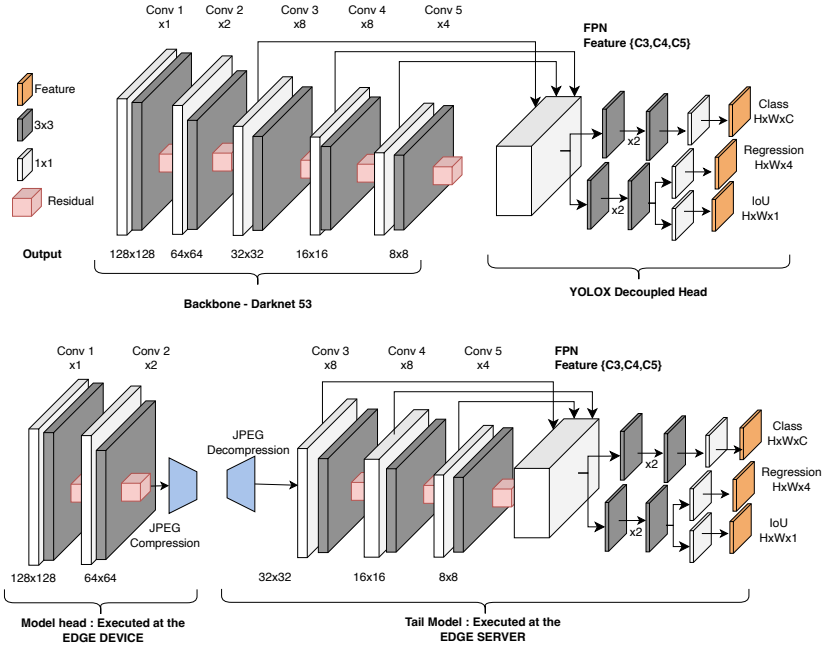


Figure 5.4: Inference Split Model Architecture. **Top:**YOLOX implementation of YOLOv3 with Darkent-53 as backbone. **Bottom: Inference Split Model implemented in the CAMTEC framework.**

Traditional methods such as R-CNNs, SSDs, and anchor-based approaches enhanced accuracy but introduced significant computational and memory demands, limiting their effectiveness on resource-constrained devices. To address these limitations, the YOLO (You Only Look Once) series [28] achieves real-time performance by framing bounding-box prediction as a regression task rather than discrete classification. Specifically, YOLOv3 [28] integrates the Darknet-53 neural network as its backbone, consolidating initial layers into a single streamlined feature extraction path (as depicted in Fig. 5.4). This architectural trait is particularly suited to split inference, aligning with our goal of optimizing computational resources for real-time applications. For this reason, we specifically utilize the Darknet-53 backbone from YOLOX models [31], which further refine the YOLO framework by removing anchor boxes and employing a decoupled head design for improved accuracy and efficiency.

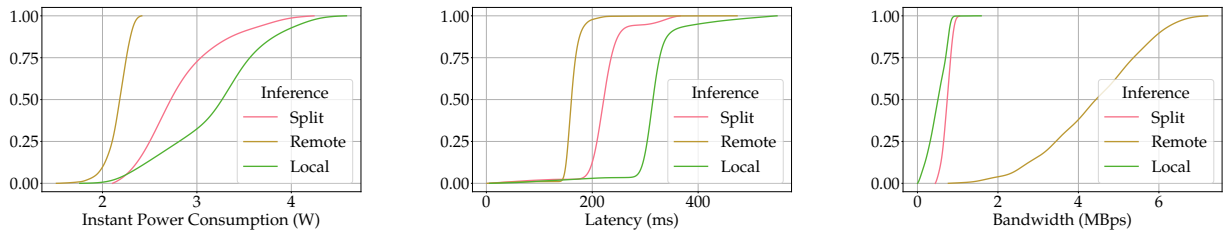


Figure 5.5: Cumulative Distribution Function (CDF) of key performance features: (a) Instant Power Consumption, (b) End-to-End Latency, (c) Bandwidth. Data were collected from an autonomously navigated trajectory at multiple speeds.

5.3.4 Observation

In our preliminary experiments, we evaluated three metrics: power consumption of the UGV, latency of the inference, and bandwidth usage during controlled robot trajectories with a fixed obstacle. We utilized a YOLOx detection model fine-tuned on the COCO dataset [49] with 47.7 mAP [31], processing 640×480 pixel frames. We identified the threshold parameter p for successful obstacle avoidance in an indoor cluttered space at speeds of 0.05, 0.10, and 0.15 m/s across three inference modes. As shown in the cumulative distributions in Fig. **??**, split inference outperforms local inference by reducing power consumption, and latency through partial offloading to servers. Although remote inference achieves the lowest consumption of power and latency of the inference, it requires significantly higher bandwidth, underscoring the trade-off between computational efficiency and network resource demands in edge computing systems.

5.4 CAMTEC Framework Description

To enable context-aware adaptive inference, we built CAMTEC over a WiFi-based wireless network, and it has the flexibility to be deployed over other network protocols as well (e.g., 5G). We provide a detailed workflow of the CAMTEC framework, referring to Fig. 5.1.

Step 1: A suite of multi-modal sensors provide the state space representation to the DRL algorithm. The battery, CPU, GPU, and WiFi modules provide the real-time data corresponding to the energy, compute, and network states. The camera captures the real-time environment in front of the UGV. The captured images are stored in a Last-In-First-Out (LIFO).

Step 2: The DRL module processes detection results alongside critical system metrics, including latency, Collision Avoidance (CA) state, computing energy consumption, and network performance. These inputs enable the agent to determine the optimal inference configuration and the vehicle's linear speed. The agent's decision-making process is designed to balance computational efficiency with adaptive responses to dynamic environmental conditions while optimizing for reliable navigation and maintaining a target speed. This proactive decision-making ensures that the vehicle follows its intended trajectory efficiently in the absence of immediate obstacles, supporting seamless and adaptive navigation in complex scenarios.

Step 3: The frames from the camera are processed by the YOLO object detection model, in either remote, local, and split inference modes, as recommended by the DRL module.

Steps 4,5: For instances where the DRL module chooses remote or split inference, the camera/tensor data are transmitted in the uplink to the remote server via WiFi.

Step 6: The inference results, which are provided as feedback to the UGV, include the identified obstacles and their positions, the Object-to-Frame Area Ratio (p) and the Collision Avoidance (CA) state. The CA state determines the UGV navigation commands, specifying the required maneuver (e.g., turning left, right, or adjusting the trajectory) to ensure safe and efficient obstacle avoidance.

Step 7: The Collision Avoidance logic processes the CA state to compute navigation commands, determining the required linear and angular velocities for effective obstacle avoidance.

The CA state is governed by two key parameters: (1) the p-value, which quantifies the distance to a predefined threshold based on the size of the object within the vehicle's field of view, and (2) the center position of the detected object. The system transitions between states, such as executing a left-turn motion for a specific number of iterations with predefined angular velocities, to a correct trajectory state. The correct trajectory state allows the vehicle to replicate its pre-CA movements, ensuring smooth re-alignment with the planned navigation path while maintaining overall operational efficiency and stability.

Step 8: The calculated velocities are transmitted to this node, which executes physical motion commands for the UGV. This ensures real-time adjustments to navigate around obstacles and maintain an efficient trajectory toward the target.

5.5 Performance Evaluation

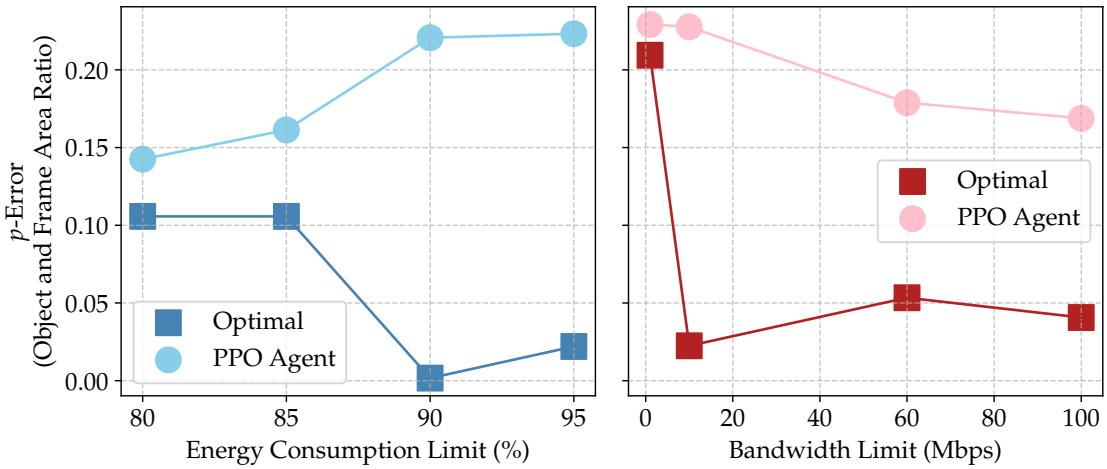


Figure 5.6: p-error for Optimal Trajectory vs. PPO Agent within battery and bandwidth constraints

To evaluate CAMTEC, we calculated the optimal changes in inference execution required to achieve an average trajectory speed while avoiding collisions. Battery level and bandwidth constraints were established as baseline parameters to assess their impact on navigation

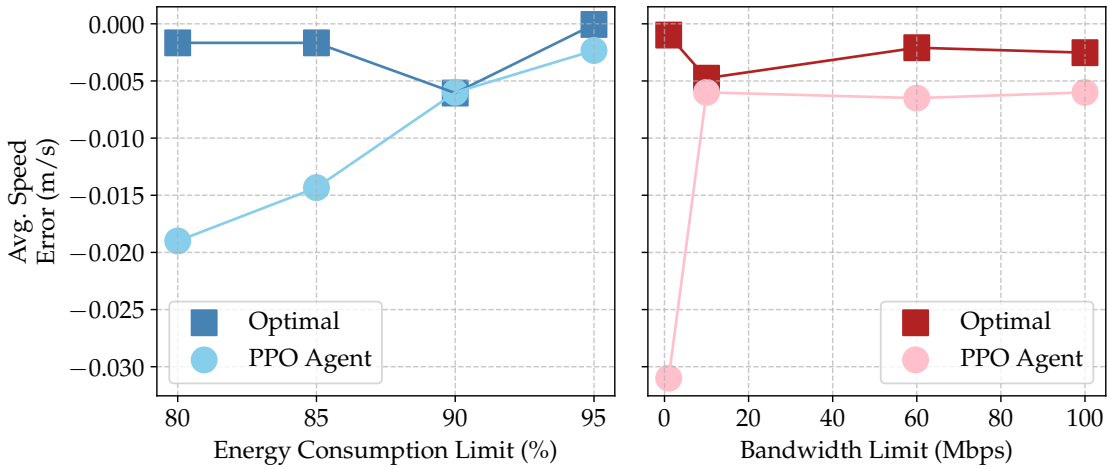


Figure 5.7: Average speed error for Optimal Trajectory vs. PPO Agent within battery and bandwidth constraints.

performance and resource consumption. From the constructed dataset, the navigation performance, denoted by f, was optimized under constraints ensuring a minimum battery level of 95%, 90%, 85%, and 80%, after reaching the target destination. We evaluate our PPO agent using average speed error and p-error, comparing its decisions to an optimal trajectory that selects the best speed and inference mode at each next interval, defined as longer than the inference time. Average speed error measures the deviation in chosen speeds, while p-error is the average p-value of detected objects per interval compared to a predefined threshold. Fig. 5.6 and 5.7 illustrate the two primary factors contributing to application performance: p, the object-to-frame area ratio, and the average speed. The results demonstrate that the policy implemented by the PPO agent closely aligns with the optimal values under the defined battery and bandwidth constraints. Notably, the PPO agent exhibits a conservative approach to both metrics—speed and p—particularly when resources are limited. The DRL agent chooses more remote/split inference modes based on network capacity, and the RTT latency causes the p-error to increase slightly even with reduced vehicle speed (as seen in Fig. 5.6). This behavior results in the PPO agent achieving performance closer to the optimal solution when more resources are available. For example, with a maximum constrained battery of 20%, allowing up to 80% utilization, and a bandwidth limit of 100 Mbps, which ap-

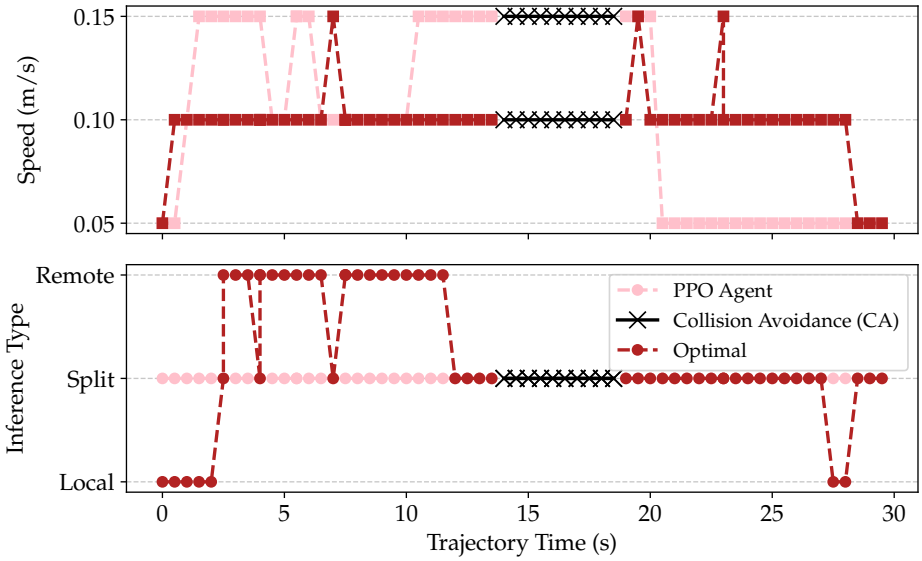


Figure 5.8: Sample trajectory under a bandwidth constraint of 1 Mbps and an energy consumption limit of 90%.

proximates an unlimited small network scenario, the agent performs more efficiently because the PPO agent executes a lower speed to keep a safe distance to the obstacle.Moreover, the optimal solution reveals that increased bandwidth availability leads to reduced UGV speed. This is attributed to the latencies introduced by remote and split inference, which may be the only viable options when battery levels are critically low or CPU/GPU loads are high (as seen in Fig. 5.7). In contrast, higher battery levels enable faster trajectories, as the system can afford higher energy expenditure for performance improvements. In Fig. 5.8, we illustrate a representative sequence of actions taken by the PPO agent along a sample trajectory.

5.6 Discussion

The proposed framework, CAMTEC, addresses the critical challenge of optimizing overall machine-task performance, moving beyond the isolated improvements of individual ML models. Specifically designed for vision-based autonomous navigation, CAMTEC employs a

context-aware approach that integrates real-time system state understanding based on the computational resources, energy availability, and network conditions. Our PPO agent implemented in the framework shows how the UGV dynamically transitions between speeds as well as inference modes – local, remote, and split – effectively distributing the computational load. This distribution enhances autonomous navigation performance, particularly under bandwidth and battery constraints. Experimental validation on a ground vehicle testbed demonstrates CAMTEC's ability to navigate environments, achieving robust performance with an error range of 0% to 30% from the optimal expected trajectory. These results demonstrate CAMTEC's adaptability and practicality in managing machine-task execution in resource-constrained environments.

Chapter 6

Conclusions

This dissertation presents and validates a novel, semantics-driven framework for cross-layer resource management in Mobile Edge Computing Systems (MECS). The main thesis is that using semantic awareness, which we identify as the contextual value of information to task outcomes, as part of the control and joint optimization of computation and communication resources, can significantly improve end-to-end application performance. This approach contrasts with conventional methods that rely on optimizing decoupled, system-level metrics such as average latency or buffer occupancy.

The core contribution is a three-tier hierarchical control architecture where each layer is governed by a semantic Deep Reinforcement Learning (DRL) agent. At the lower layers, a computing controller allocates resources across the mobile device and edge server, explicitly modeling context switching overhead to meet per-task latency and performance goals. A cross-layer communications controller tunes wireless parameters and packet duplication strategies to mitigate packet loss, optimizing for reliability under strict latency and task-level performance. At the higher layers, a coordination controller selects inference modalities and other complex task parameters, unifying the management of computation and communica-

tion to ensure robust performance for complex, closed-loop tasks.

The efficacy of these frameworks was demonstrated on dedicated hardware testbeds using mobile platforms. The integrated systems consistently achieved superior task-level accuracy or precision for real-time deadlines set below 30 milliseconds of inference for the lower-layered and cross-layered work, and enhanced communication reliability under mobility, thereby confirming the direct link between semantic resource allocation and task-level performance.

This work establishes a vertically integrated control paradigm where semantics unifies the management of computation, communication, and task-level performance. It proposes a refocusing from optimizing isolated metrics to the goal of ensuring the timely and reliable delivery of task-critical information. Future research directions include extending these frameworks to multi-server topologies, incorporating formal safety guarantees, developing aware scheduling, and designing semantics compression schemes. These developments further articulate the underlying principle of this dissertation: semantics as the unifying foundation for intelligent, cross-layer control.

Bibliography

[1] A. Acheampong, Y. Zhang, X. Xu, and D. A. Kumah. A review of the current task offloading algorithms, strategies and approach in edge computing systems. *Computer Modeling in Engineering & Sciences, 134 (1)*, pages 35–88, 2023.

[2] M. Ahmed, S. Raza, M. A. Mirza, A. Aziz, M. A. Khan, W. U. Khan, J. Li, and Z. Han. A Survey on Vehicular Task Offloading: Classification, Issues, and Challenges. *Journal of King Saud University-Computer and Information Sciences*, 34(7):4135–4162, 2022.

[3] M. Y. Akhlaqi and Z. B. M. Hanapi. Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions. *Journal of Network and Computer Applications*, 212:103568, 2023.

[4] I. F. Akyildiz and H. Guo. Wireless communication research challenges for extended reality (XR). *ITU Journal on Future and Evolving Technologies*, 3(1):1–15, 2022.

[5] F. Alam, A. Toosi, M. Cheema, C. Cicconetti, P. Serrano, A. Iosup, Z. Tari, and A. Sarvi. Serverless Vehicular Edge Computing for the Internet of Vehicles. *IEEE Internet Computing*, 2023.

[6] C. Bai, P. Dallasega, G. Orzes, and J. Sarkis. Industry 4.0 technologies assessment: A sustainability perspective. *International journal of production economics*, 229:107776, 2020.

[7] T. Baker, M. Asim, H. Tawfik, B. Aldawsari, and R. Buyya. An energy-aware service composition algorithm for multiple cloud-based iot applications. *Journal of Network and Computer Applications*, 89:96–108, 2017.

[8] A. Bakhtiarnia, N. Milošević, Q. Zhang, D. Bajović, and A. Iosifidis. Dynamic split computing for efficient deep edge intelligence. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[9] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

[10] A. Bhat and S. Dasadhikari. Nvidia xavier documentation, 2018.

[11] J. Bi, H. Yuan, K. Zhang, and M. Zhou. Energy-minimized Partial Computation Offloading for Delay-Sensitive Applications in Heterogeneous Edge Networks. *IEEE Transactions on Emerging Topics in Computing*, 10(4):1941–1954, 2022.

[12] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou. Helicon: Orchestrating low-latent & load-balanced virtual network functions. In *IEEE ICC*, pages 353–358, 2022.

[13] D. Callegaro, M. Levorato, and F. Restuccia. Seremas: Self-resilient mobile autonomous systems through predictive edge computing. In *2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2021.

[14] D. Callegaro, Y. Matsubara, and M. Levorato. Optimal task allocation for Time-Varying edge computing systems with split DNNs. In *2020 IEEE Global Communications Conference: Selected Areas in Communications: Internet of Things and Smart Connected Communities (Globecom2020 SAC IoTSCC)*, Taipei, Taiwan, Dec. 2020.

[15] L. Caronti, K. Akhunov, M. Nardello, K. S. Yıldırım, and D. Brunelli. Fine-grained hardware acceleration for efficient batteryless intermittent inference on the edge. *ACM Transactions on Embedded Computing Systems*, 22(5):1–19, 2023.

[16] S. Cass. Nvidia makes it easy to embed ai: The jetson nano packs a lot of machine-learning power into diy projects - [hands on]. *IEEE Spectrum*, 57(7):14–16, 2020.

[17] F. Chai, Q. Zhang, H. Yao, X. Xin, R. Gao, and M. Guizani. Joint Multi-task Offloading and Resource Allocation for Mobile Edge Computing Systems in Satellite IoT. *IEEE Transactions on Vehicular Technology*, 2023.

[18] Y. Chang, Y. Cheng, U. Manzoor, and J. Murray. A Review of UAV Autonomous Navigation in GPS-Denied Environments. *Robotics and Autonomous Systems*, page 104533, 2023.

[19] A. Community. Hugging face: The ai community building the future.

[20] N. Corporation. Tegrastats utility, 2019.

[21] P. Cuda. Torch cuda.

[22] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional Reinforcement Learning with Quantile Regression, 2017.

[23] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[24] P. Dinh et al. Demystifying Resource Allocation Policies in Operational 5G mmWave Networks. In *IEEE WoWMoM*, pages 1–10, 2022.

[25] M. Ditty. Nvidia orin system-on-chip. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–17. IEEE Computer Society, 2022.

[26] M. Eisen, M. M. Rashid, K. Gatsis, D. Cavalcanti, N. Himayat, and A. Ribeiro. Control aware radio resource allocation in low latency wireless control systems. *IEEE Internet of Things Journal*, 6(5):7878–7890, 2019.

[27] M. S. Elbamby, C. Perfecto, C.-F. Liu, J. Park, S. Samarakoon, X. Chen, and M. Bennis. Wireless edge computing with latency and reliability guarantees. *Proc. of the IEEE*, 107(8):1717–1737, 2019.

[28] A. Farhadi and J. Redmon. Yolov3: An incremental improvement. In *Computer vision and pattern recognition*, volume 1804, pages 1–6. Springer Berlin/Heidelberg, Germany, 2018.

[29] R. A. Fezeu et al. An In-Depth measurement analysis of 5G mmwave PHY latency and its impact on End-to-End delay. In *International Conference on Passive and Active Network Measurement*, pages 284–312. Springer, 2023.

[30] A. Fresa and J. P. Champati. Offloading algorithms for maximizing inference accuracy on edge device under a time constraint. *arXiv preprint arXiv:2112.11413*, 2021.

[31] Z. Ge. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[32] H. Guo, J. Liu, and J. Lv. Toward intelligent task offloading at the edge. *IEEE Network*, 34(2):128–134, 2019.

[33] K. F. Haque, F. Meneghello, M. E. Karim, and F. Restuccia. SAWEC: Sensing-Assisted Wireless Edge Computing. In *22nd International Conference on Pervasive Computing and Communications (PerCom 2024)*, 2024.

[34] K. F. Haque, M. Zhang, F. Meneghello, and F. Restuccia. Beamsense: Rethinking wireless sensing with mu-mimo wi-fi beamforming feedback. *arXiv preprint arXiv:2303.09687*, 2023.

[35] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[36] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng. Multi-hop cooperative computation offloading for industrial iot–edge–cloud computing environments. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2759–2774, 2019.

[37] IEEE Computer Society, LAN/MAN Standards Committee. Ieee standard for information technology– telecommunications and information exchange between systemslocal and metropolitan area networks– specific requirements–part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications–amendment 4: Enhancements for very high throughput for operation in bands below 6 ghz. *IEEE Std 802.11ac-2013 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae-2012, IEEE Std 802.11aa-2012, and IEEE Std 802.11ad-2012)*, pages 1–425, 2013.

[38] IEEE Computer Society, LAN/MAN Standards Committee. IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: Enhancements for High-Efficiency WLAN. *IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020)*, pages 1–767, 2021.

[39] H. E. Ilhan, S. Ozer, G. K. Kurt, and H. A. Cirpan. Offloading deep learning empowered image segmentation from uav to edge server. In *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pages 296–300. IEEE, 2021.

[40] A. Islam, A. Debnath, M. Ghose, and S. Chakraborty. A Survey on Task Offloading in Multi-Access Edge Computing. *Journal of Systems Architecture*, 118:102225, 2021.

[41] G. Jocher. Yolo by ultralytics. `https://github.com/ultralytics/yolov5`, 2023. Accessed: 2022-08-12.

[42] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, K. Michael, TaoXie, J. Fang, imyhxy, Lorna, Z. Yifu, C. Wong, A. V, D. Montes, Z. Wang, C. Fati, J. Nadar, Laughing, UnglvKitDe, V. Sonck, tkianai, yxNONG, P. Skalski, A. Hogan, D. Nair, M. Strobel, and M. Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Real-time Instance Segmentation. `https://doi.org/10.5281/zenodo.7347926`, Nov. 2022. Version v7.0, Publisher: Zenodo, doi: 10.5281/zenodo.7347926.

[43] A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-100 (Canadian Institute for Advanced Research, 100 classes). Technical report, CIFAR-100, 2009.

[44] D. S. Lakew, N.-N. Dao, S. Cho, et al. Adaptive Partial Offloading and Resource Harmonization in Wireless Edge Computing-assisted IoE Networks. *IEEE Transactions on Network Science and Engineering*, 9(5):3028–3044, 2022.

[45] D. S. Lakew, A.-T. Tran, N.-N. Dao, and S. Cho. Intelligent Self-Optimization for Task Offloading in LEO-MEC-Assisted Energy-Harvesting-UAV Systems. *IEEE Transactions on Network Science and Engineering*, pages 1–14, 2024.

[46] N. G. Larrakoetxea, J. E. Astobiza, I. P. López, B. S. Urquijo, J. G. Barruetabeña, and A. Z. Rego. Efficient machine learning on edge computing through data compression techniques. *IEEE Access*, 11:31676–31685, 2023.

[47] F. Li, X. Wang, Z. Wang, J. Cao, X. Liu, Y. Bi, W. Li, and Y. Wang. A Local Communication System Over Wi-Fi Direct: Implementation and Performance Evaluation. *IEEE Internet of Things Journal*, 7(6):5140–5158, 2020.

[48] H. Li, P. Zheng, T. Wang, J. Wang, and T. Liu. A Multi-Objective Task Offloading based on BBO Algorithm Under Deadline Constrain in Mobile Edge Computing. *Cluster Computing*, 26(6):4051–4067, 2023.

[49] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015.

[50] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE communications surveys & tutorials*, 19(3):1628–1656, 2017.

[51] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh. Distilled split deep neural networks for edge-assisted real-time systems. *IEEE Internet of Things Journal*, 9(1):700–713, 2022.

[52] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato, and F. Restuccia. Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing. *arXiv preprint arXiv:2201.02693*, 2022.

[53] Y. Matsubara, S. Fujii, and Y. Sakagami. Split computing and early exiting for deep learning applications: Survey and research challenges. *arXiv preprint arXiv:2103.04505*, 2021.

[54] 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):1–30, 2022.

[55] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt. Supervised compression for resource-constrained edge computing systems. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, jan 2022.

[56] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *2011 IEEE International Conference on Robotics and Automation*, pages 2992–2997, 2011.

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

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[59] P. Multiprocessing. Multiprocessing package - torch.multiprocessing.

[60] Next Generation Alliance. Video Codecs for 6G Machine-Type Communications: Improving the Quality of Critical Application Roles.

[61] L. Niu, X. Chen, N. Zhang, Y. Zhu, R. Yin, C. Wu, and Y. Cao. Multi-Agent Meta-Reinforcement Learning for Optimized Task Scheduling in Heterogeneous Edge Computing Systems. *IEEE Internet of Things Journal*, 2023.

[62] NVIDIA Corporation. Jetson agx orin for next-gen robotics. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/`. Accessed: 2024-02-11.

[63] S. Ozer, E. Ilhan, M. A. Ozkanoglu, and H. A. Cirpan. Offloading deep learning powered vision tasks from uav to 5g edge server with denoising. *IEEE Transactions on Vehicular Technology*, 2023.

[64] K. Pahlavan and P. Krishnamurthy. Evolution and Impact of Wi-Fi Technology and Applications: A Historical Perspective. *International Journal of Wireless Information Networks*, 28:3–19, 2021.

[65] Y. Park, U. Gim, and M. J. Kim. Edge storage management recipe with zero-shot data compression for road anomaly detection, 2023.

[66] P. Popovski, Č. Stefanović, J. J. Nielsen, E. De Carvalho, M. Angjelichinoski, K. F. Trillingsgaard, and A.-S. Bana. Wireless access in ultra-reliable low-latency communication (urllc). *IEEE Transactions on Communications*, 67(8):5783–5801, 2019.

[67] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[68] Roboworks. *Rosbot User Manual v.20250409*. Roboworks, Barangaroo, NSW, Australia, Apr. 2025. Version 20250409.

[69] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, and S. Papavassiliou. Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 195:108177, 2021.

[70] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, and S. Papavassiliou. Task Offloading in Edge and Cloud Computing: A Survey on Mathematical, Artificial Intelligence and Control Theory Solutions. *Computer Networks*, 195:108177, 2021.

[71] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[72] P. Schulz and e. al. Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture. *IEEE Communications Magazine*, 55(2):70–78, 2017.

[73] C. Shu, Z. Zhao, Y. Han, and G. Min. Dependency-aware and latency-optimal computation offloading for multi-user edge computing networks. In *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2019.

[74] P. Silva, N. T. Almeida, and R. Campos. A comprehensive study on enterprise wi-fi access points power consumption. *IEEE Access*, 7:96841–96867, 2019.

[75] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[76] H. Tabani et al. Improving the efficiency of transformers for resource-constrained devices. In *24th Euromicro Conference on Digital System Design*, pages 449–456. IEEE, 2021.

[77] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[78] Z. Tang, W. Jia, X. Zhou, W. Yang, and Y. You. Representation and reinforcement learning for task scheduling in edge computing. *IEEE Transactions on Big Data*, 2020.

[79] TP-Link Corporation. Archer ax55 — ax3000 dual band gigabit wi-fi 6 router. `https://www.tp-link.com/us/home-networking/wifi-router/archer-ax55/`. Accessed: 2024-02-11.

[80] T. X. Tran, K. Chan, and D. Pompili. Costa: Cost-aware service caching and task offloading assignment in mobile-edge computing. In *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2019.

[81] U.S DoT. Saving Lives with Connectivity: A Plan to Accelerate V2X Deployment.

[82] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[83] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin. A Survey and Taxonomy on Task Offloading for Edge-Cloud computing. *IEEE Access*, 8:186080–186101, 2020.

[84] H. Wang, B. Kim, J. Xie, and Z. Han. Energy drain of the object detection processing pipeline for mobile devices: Analysis and implications. *IEEE Transactions on Green Communications and Networking*, 5(1):41–60, 2021.

[85] H. Xiao, C. Xu, Y. Ma, S. Yang, L. Zhong, and G.-M. Muntean. Edge Intelligence: A Computational Task Offloading Scheme for Dependent IoT Application. *IEEE Transactions on Wireless Communications*, 21(9):7222–7237, 2022.

[86] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021.

[87] T. Zeng, A. Ferdowsi, O. Semiari, W. Saad, and C. S. Hong. Convergence of communications, control, and machine learning for secure and autonomous vehicle navigation. *IEEE Wireless Communications*, 31(4):132–138, 2024.

[88] Y. Zeng, P. H. Pathak, and P. Mohapatra. A first look at 802.11 ac in action: Energy efficiency and interference characterization. In *2014 IFIP Networking Conference*, pages 1–9. IEEE, 2014.

[89] S. Zhang, H. Gu, K. Chi, L. Huang, K. Yu, and S. Mumtaz. Drl-based Partial Offloading for Maximizing Sum Computation Rate of Wireless Powered Mobile Edge Computing Network. *IEEE Transactions on Wireless Communications*, 21(12):10934–10948, 2022.

[90] T. Zhang, Z. Li, Y. Chen, K.-Y. Lam, and J. Zhao. Edge-cloud cooperation for dnn inference via reinforcement learning and supervised learning. In *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 77–84. IEEE, 2022.

[91] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri. Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1270–1278. IEEE, 2019.

[92] Z. Zhao, K. Wang, N. Ling, and G. Xing. Edgeml: An automl framework for real-time deep learning on the edge. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, pages 133–144, 2021.