

**Design and Development of a Context-Aware Collaborative Autonomous
Real-Time Vehicle Systems Framework**

by

Maria Joelma Pereira Peixoto

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Faculty of Engineering and Applied Science
University of Ontario Institute of Technology (Ontario Tech University)
Oshawa, Ontario, Canada
November, 2023

© Maria Peixoto, 2023

THESIS EXAMINATION INFORMATION

Submitted by: **Maria Joelma Pereira Peixoto**

Doctor of Philosophy in Electrical and Computer Engineering

Thesis title: Design and Development of a Context-Aware Collaborative Autonomous Real-Time Vehicle Systems Framework

An oral defense of this thesis took place on October 26, 2023 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Khalid Elgazzar
Research Supervisor	Dr. Akramul Azim
Examining Committee Member	Dr. Khalil El-Khatib
Examining Committee Member	Dr. Masoud Makrehchi
University Examiner	Dr. Richard Pazzi
External Examiner	Dr. Salimur Choudhury - Queens University

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

The increasing autonomy of intelligent systems, with applications extending from self-driving vehicles to home-based robots, has emerged as a critical area of focus in modern research. Yet, to acknowledge the full potential of these systems, numerous challenges must be addressed. This thesis encapsulates rigorous research resulting in eight scientific papers investigating autonomous systems' efficacy and efficiency. Our study proposes the Context-Aware Collaborative Autonomous Real-Time Vehicle Systems (CARVS) Framework and focuses on improving context awareness, simplifying remote task processing, and quantifying prediction uncertainty in Machine Learning (ML) algorithms. Our intention is to move forward the state-of-the-art in autonomous systems based on our findings as we investigate the employment of noise as a stimulus to boost agent exploration. We also address the development of mapping and task management systems for connected autonomous vehicles (CAVs) using edge, fog, and cloud computing. Furthermore, we study the quantification of uncertainty in ML algorithm predictions to describe their behaviours and decision-making mechanisms. This research provides valuable insights for the continuous improvement of autonomous learning and the ability to deal with uncertainties in dynamic and unpredictable environments, which could lead to greater acceptance of such systems.

Keywords: Autonomous Intelligent Systems; Machine Learning Algorithms; Context Awareness; Task Mapping and Management; Uncertainty Quantification.

Author's Declaration

I hereby declare that this thesis consists of original work which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize the University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

Maria Joelma Pereira Peixoto

Statement of Contributions

The results of my thesis research have been shared through the following publications:

1. **Peixoto, Maria J. P.** and A. Azim, “A collaborative and distributed task management system for real-time systems,” in *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, 2023, pp. 117–125.
DOI: [10.1109/ISORC58943.2023.00024](https://doi.org/10.1109/ISORC58943.2023.00024)
2. **Maria J. P. Peixoto** and A. Azim, “Design and development of a machine learning-based task orchestrator for intelligent systems on edge networks,” *IEEE Access*, vol. 11, pp. 33 049–33 060, 2023. DOI: [10.1109/ACCESS.2023.3263483](https://doi.org/10.1109/ACCESS.2023.3263483)
3. **Maria J. P. Peixoto**, A. Azim, J. Sheehan, and D. Timothy, “An intelligent traffic monitoring embedded system using video data mining,” in *2022 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, 2022, pp. 1–6. DOI: [10.1109/AIPR57179.2022.10092207](https://doi.org/10.1109/AIPR57179.2022.10092207)
4. **Peixoto, Maria JP** and A. Azim, “Improving environmental awareness for autonomous vehicles,” *Applied Intelligence*, vol. 53, no. 2, pp. 1842–1854, 2022.
DOI: <https://doi.org/10.1007/s10489-022-03468-6>
5. **Maria J.P. Peixoto** and A. Azim, “Using time-correlated noise to encourage exploration and improve autonomous agents performance in reinforcement learning,” *Procedia Computer Science*, vol. 191, pp. 85–92, 2021, The 18th International Conference on Mobile Systems and Pervasive Computing (MobiSPC),

The 16th International Conference on Future Networks and Communications (FNC), The 11th International Conference on Sustainable Energy Information Technology, ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2021.07.014>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S187705092101406X>

6. **Peixoto, J.P. Maria** and A. Azim, “Context-based learning for autonomous vehicles,” in *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, 2020, pp. 150–151. DOI: [10.1109/ISORC49007.2020.00033](https://doi.org/10.1109/ISORC49007.2020.00033)
7. **Peixoto, J.P. Maria** and A. Azim, “Explainable artificial intelligence (XAI) approach for reinforcement learning systems,” in *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC)*, [Accepted], 2023

Acknowledgements

I want to express my gratitude to my supervisor, Dr. Akramul Azim, for his wise guidance and unwavering support throughout my journey. His belief in my potential has been invaluable. I would also like to acknowledge the support of my colleagues in the RTEMSOFT lab, whose contribution has been instrumental in my success. Finally, I am grateful to the committee members for their insightful feedback.

Additionally, I want to thank my friends, siblings, and parents, Maria and José, for supporting and encouraging me to follow and always believe in my goals. Also, I am eternally grateful to the great love of my life, Andrei Bosco, for making this difficult journey smooth and full of love, kindness and care.

I also would like to thank the love and friendship of Azula, Vandinha, Carolina Maia and Hamilton de Vasconcellos in memoriam. Last but not certainly least, I thank my psychoanalyst, Walmy Silveira, for helping me understand and calm some voices in the madness of my mind.

*I must not fear.
Fear is the mind-killer.
Fear is the little-death that brings total obliteration.
I will face my fear.
I will permit it to pass over me and through me.
And when it has gone past, I will turn the inner eye to see its path.
Where the fear has gone there will be nothing. Only I will remain.*

(Litany Against Fear, Dune, Frank Herbert)

Table of Contents

Thesis Examination Information	ii
Abstract	iii
Author’s Declaration	iv
Statement of Contributions	v
Acknowledgements	vii
List of Tables	x
List of Figures	xi
List of Abbreviations	xiii
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	3
1.3 Contributions	5
1.4 Thesis Organization	8
Chapter 2 Background and Related Work	10
2.1 Context awareness for autonomous real-time vehicle systems	11
2.1.1 Context Awareness	11
2.1.2 Reinforcement learning	12
2.1.3 Gaussian Process	15
2.1.4 <i>Ornstein-Uhlenbeck</i> Process	15
2.2 Collaborative task management for autonomous real-time vehicle systems	16
2.2.1 Centralized approach	17
2.2.2 Distributed approach	19
2.2.3 Platooning	20
2.3 Learning and continuous monitoring for autonomous real-time vehicle systems	21
2.3.1 Uncertainty in events	21
2.3.2 Sensitivity analysis	23
2.3.3 Explainable Artificial Intelligence (XAI)	25
2.4 Overview of related works	29
Chapter 3 Proposed Context-Aware Collaborative Autonomous Real-Time Vehicle Systems Framework	33
3.1 Context Awareness Module	36
3.2 Collaborative Task Management Module	39
3.3 Learning and Continuous Monitoring Module	41

Chapter 4	Context awareness for autonomous real-time vehicle systems	43
Chapter 5	Collaborative task management for autonomous real-time vehicle systems	54
5.1	Centralized Approach	54
5.2	Distributed Approach	71
Chapter 6	Learning and continuous monitoring for autonomous real-time vehicle systems	84
6.1	Scenario 1	87
6.2	Scenario 2	98
Chapter 7	Experimental Results and Discussion	100
7.1	Context Awareness for Autonomous Systems	100
7.1.1	Tasks with empty city	106
7.1.2	Tasks with populated city	108
7.2	Task Management in Autonomous Systems	113
7.3	Learning and Performance Monitoring of Autonomous Systems	116
Chapter 8	Conclusion and Future Works	133
8.1	Conclusion	133
8.2	Future Work	135
References		138

List of Tables

Table 2.1	Summary of related work (Not Mentioned (N.M)=Not Mentioned)	29
Table 5.1	Details of features selected as input	58
Table 5.2	Defined characteristics for each of the applications used	61
Table 5.3	Performance evaluation with 10-fold cross-validation	63
Table 5.4	Machine Learning (ML) algorithms performance evaluation	72
Table 5.5	Features used as inputs by task mapper	72
Table 5.6	Tasks defined for the simulation	76
Table 5.7	General comparison of the analyzed approaches	78
Table 5.8	Analysis of the processing success percentage of each approach to CollisionPreventionTask	83
Table 6.1	Models performance for the binary classification problem	94
Table 6.2	Models performance for the multi-level classification problem	96
Table 6.3	Models performance for the regression problem	97
Table 6.4	Models performance for fire detection scenario	99
Table 7.1	Comparison with previously published works on hard exploration (Private Eye, Gravitar and Pitfall) and one score exploit (Seaquest) Atari games.	113
Table 7.2	Training details of evaluated agents	124
Table 7.3	Comparison of our approach and related works	132

List of Figures

Figure 2.1	Centralized approach for remote workload processing	17
Figure 2.2	Over-the-air task processing in a platooning scenario	20
Figure 3.1	Context-Aware Collaborative Autonomous Real-Time Vehicle Systems (CARVS) Framework	33
Figure 3.2	Random walk simulation using python	37
Figure 3.3	Random walk result. Variability of the position x of the agent over time.	38
Figure 3.4	Distribution of the agents' displacement data during the random walk.	39
Figure 4.1	Pendulum-v0 task - training	47
Figure 4.2	LunarLander-v2 task - training	48
Figure 4.3	Humanoid-v2 task - training	49
Figure 4.4	Pendulum-v0 task - evaluation	51
Figure 4.5	LunarLander-v2 task - evaluation	52
Figure 4.6	Humanoid-v2 task - evaluation	53
Figure 5.1	Edge orchestrator workflow (RSU = Roadside Units/ CBS = Cellular Base Station)	55
Figure 5.2	Our one-stage proposed architecture	57
Figure 5.3	(S1 = first-order Sobol index, ST = total Sobol index) - Variance-based sensitivity analysis to define 'Selected features'.	58
Figure 5.4	Two-stage architecture proposed in [37]	60
Figure 5.5	Comparison of our proposal with the proposal presented in [37]	64
Figure 5.6	Number of tasks that failed during simulation in EdgeCloudSim	66
Figure 5.7	Orchestration algorithm overhead during simulation in EdgeCloudSim	67
Figure 5.8	Network delay of approaches during simulation in EdgeCloudSim	68
Figure 5.9	Algorithm simulation time duration in EdgeCloudSim	69
Figure 5.10	Average QoE for the number of vehicles	70
Figure 5.11	Tasks distributed percentage on each server by analyzed approach	79
Figure 5.12	Percentage of successfully completed tasks in total and by server	80
Figure 5.13	Percentage of successfully completed tasks according to their criticality on the AGX Edge Server	81
Figure 5.14	Percentage of successfully completed tasks according to their criticality in Tesla Fog Server	81
Figure 5.15	Percentage of successfully completed tasks according to their criticality on the DGX Cloud Server	82
Figure 6.1	Conformal predictor workflow	85
Figure 6.2	Comparison of the traffic network used in the simulation with the real map of Toronto	88
Figure 6.3	Proposed approach to estimate uncertainty in events	89
Figure 6.4	Sobol analysis to indicate priority input for binary classification problem	93

Figure 6.5	Sobol analysis to indicate priority input for multi-level classification problem	95
Figure 6.6	Sobol analysis to indicate priority input for regression problem . .	97
Figure 6.7	Sobol analysis to indicate priority input for fire detection problem	98
Figure 7.1	Agent Asynchronous Advantage Actor Critic (A3C) model	102
Figure 7.2	Specification of the A3C agent connected to the CARLA simulator	103
Figure 7.3	Tasks performed with empty city	109
Figure 7.4	Tasks performed with populated city	110
Figure 7.5	Comparison between distributed and centralized approaches . . .	114
Figure 7.6	Success Rate by Processing Units	115
Figure 7.7	Task Success by Criticality	117
Figure 7.8	Task Success Analysis	118
Figure 7.9	Abstraction of our proposal Double Deep Q-Network (DDQN) with uncertainty	121
Figure 7.10	Simulation environment	123
Figure 7.11	Scoring performance comparison between approaches with and without uncertainty	123
Figure 7.12	Details of the explainability screen in our proposal	125
Figure 7.13	Explanation of the behaviour and actions of the DDQN agent in the obstacle avoidance scenario	128
Figure 7.14	Explanation of the behaviour and actions of the DDQN with uncertainty agent in the obstacle avoidance scenario	130
Figure 7.15	Agent error explanation	131

List of Abbreviations

A2C Advantage Actor Critic

A3C Asynchronous Advantage Actor Critic

A3C-DP Asynchronous Advantage Actor Critic with Disturbed Policy

AI Artificial Intelligence

AL-DQN Advantage Learning - Deep Q-Network

CARLA Car Learning to Act

CARVS Framework Context-Aware Collaborative Autonomous Real-Time Vehicle Systems Framework

CAVs Connected Autonomous Vehicles

CBS Cellular Network

CEP Complex Event Processing

DDQN Double Deep Q-Network

DeepCS Deep Code Search

DQN Deep Q-Network

EDFHC Earliest Deadline First and High Criticality

GPU Graphics Processing Unit

ICP Inductive Conformal Predictor

IPM Improved Probabilistic Model

KNN k-Nearest Neighbors

MAPE Mean Absolute Percentage Error

MDP Markov Decision Process

MEC Multi-access Edge Computing

ML Machine Learning

ML_CP Machine Learning with Conformal Predictor

MLP MultiLayer Perceptron

MSE Mean Squared Error

N.M Not Mentioned

NPM Normal Probabilistic Model

OTA Over-The-Air

OU Ornstein-Uhlenbeck

PER Prioritized Experience Replay

POMDP Partially Observable Markov Decision Process

PPO Proximal Policy Optimization

PU Processing Unit

QoE Quality of Experience

ReLU Rectified Linear Unit

RL Reinforcement Learning

RLzoo Reinforcement Learning Zoo

RSU Roadside Unit

SAC Soft Actor Critic

SARFA Specific and Relevant Feature Attribution

SDE Stochastic Differential Equation

SOTA State-Of-The-Art

SUMO Simulation of Urban MObility

SVM Support Vector Machine

TCP Transductive Conformal Predictor

VM Virtual Machine

WAN Wide-Area Network

WLAN Wireless Local-Area Network

XAI Explainable Artificial Intelligence

Chapter 1. Introduction

1.1 Motivation

The ability of intelligent systems to operate independently and adapt to their surroundings, known as autonomy, has become an increasingly important area of research in recent years. This is crucial for their effectiveness in various applications, such as self-driving vehicles and household robots. As the use of these autonomous agents grows, it is essential for them to have a comprehensive understanding of their operating environment and context [1]. Despite this, there are still obstacles to overcome before these systems can reach their full potential.

One of the challenges in autonomous systems is context awareness. For a system to be truly autonomous, it must not only comprehend its surroundings but also adjust to them [2]. This requires ongoing and efficient exploration of the environment, which can be aided by introducing noise [3], [4]. This enables the system to be less confident in its actions, allowing it to explore its surroundings more thoroughly.

Efficient task management and mapping are essential elements for autonomous systems to succeed [5]. As these systems become more complex, they are required to perform a diverse range of tasks, from autonomous driving and navigating in unstructured environments to complex interactions with humans in social situations. The ability to manage and prioritize these tasks is necessary for the overall performance and practicality of these systems.

Another point is learning and performance monitoring, vital for the effective operation of autonomous systems, which constantly deal with uncertainties arising from the dynamics and unpredictability of environments [6]. These systems must be able to continuously learn from experience and adapt based on their performance, dealing with uncertain situations, allowing them to improve over time and respond effectively to new challenges.

In this way, this dissertation proposes to investigate these problems, seeking to contribute to the advancement of state-of-the-art autonomous systems by developing new techniques and approaches for context awareness, task management and mapping, and learning and performance monitoring based on uncertainty. We will facilitate the development of more effective and efficient autonomous systems, as well as inform and guide the process of autonomous learning, paving the way for its broader adoption in various applications and scenarios.

This research aims to develop real-time autonomous vehicle systems as the transportation industry moves towards complete autonomy. One of the critical reasons for pursuing this research is the real-time decision-making challenges that autonomous vehicles face, which require processing and interpreting a vast amount of data from various sensors and the need for the vehicle to communicate with other vehicles and infrastructure. We assume that the environment is composed only of autonomous vehicles. However, it can also be integrated into cars with human drivers to help them make better decisions and improve traffic efficiency. Systems must work well for autonomous and human-driven vehicles.

1.2 Problem Statement

Context awareness is crucial for making efficient and effective decisions in real-time autonomous systems. In order to become context-aware, the system must interact with the environment to understand its characteristics and functioning. [Reinforcement Learning \(RL\)](#) is used to ensure this continuous interaction and learning, where the agent is rewarded or punished for each action that affects the environment [7].

During the autonomous systems training in [RL](#), the agent can repeat the same action to have an acceptable reward. However, another motion may produce a much greater reward, and the agent will never know if it is choosing only what it has already learned in the previous explorations. For this reason, the agent must explore its environment a lot during training to get to know it well and better choose the actions during the testing phase. There are numerous approaches to stimulate the exploration of agents during the training phase, one of which is noise insertion [3], [8], [9]. Nonetheless, [RL](#) exploration still has many unexplored challenges and research opportunities. As in real life, we have several external factors when learning something new. The [RL](#) agent must go through interferences and noises that diminish its certainty in action at the training moment and make it explore more options.

Considering that some specific systems, especially connected autonomous vehicles, have limited processing capacity locally, they can take advantage of the resources on remote servers that can support processing their workload in many situations. The challenge is to develop a task management system that can handle the dynamic nature

of the environment and allocate tasks remotely, comparing and contrasting the real-time performance of centralized and distributed approaches while considering factors such as task priority and resource availability. Accordingly, we propose a concise one-stage ML-based task mapper and scheduler to solve the limited onboard processing capacity and minimize the response time for intelligent vehicles. The task mapper counts on the support of ML to estimate the service time to upload and process a task on the edge, fog or cloud. Based on this estimation, the best remote server with the lowest estimated service time will be chosen. Also, the task mapper receives numerous input features and cannot use all of them to predict the service time, as it would cause a processing overload. Therefore, we use sensitivity analysis to decide which input features of the model are decisive for forecast the required service time.

As Artificial Intelligence (AI) technology advances and impacts various industries like healthcare, agriculture, and transportation [10], it becomes essential to understand how these systems make decisions to establish trust and understanding. Complex event processing systems create events that stem from other low-level primitive events. These derived events result from rules that match patterns relevant to generating this complex event [11]. Therefore, we need to measure the level of uncertainty linked to complex events to avoid potential harm. Then, to increase transparency and comprehensibility for humans, AI explanation approaches have been developed. One way to ensure that individuals understand the system's limitations is to measure and disclose the model uncertainty related to AI predictions [12]. This method will help demonstrate that machine learning systems are not always completely accurate

and how much they can be trusted.

Thus, we evaluate the effectiveness of our proposed machine learning models by measuring them against several metrics to determine how closely the model's decisions mimic those of a human driver. These metrics include [Mean Squared Error \(MSE\)](#), which measures the accuracy of the model's predictions regarding a car's movements like acceleration and braking. A low [MSE](#) score indicates the vehicle can navigate efficiently and safely, with minimal chances of unexpected movements. Metrics such as accuracy and sensitivity are used to assess the vehicle's ability to detect objects and respond to them appropriately, such as determining when to halt for a pedestrian or another vehicle. High accuracy ensures that the autonomous vehicle will not confuse a tree for a traffic sign, while high sensitivity guarantees that it will not miss a pedestrian. In summary, these metrics serve as a report card for the autonomous vehicle, accurately assessing how well it is learning and performing the tasks required for safe and efficient driving.

1.3 Contributions

To address the challenges mentioned above, the main contributions of this research work are summarized below.

- **Researching, designing, implementing, and evaluating techniques that improve exploration and recognition of the environment. This leads**

to better context awareness for decision-making by autonomous agents.

We aim to research the behaviour of additive Gaussian noise and [Ornstein-Uhlenbeck \(OU\)](#) [13] noise to encourage an autonomous agent to be further stimulated to explore more action options efficiently during its training. Our motivation comes from the assumption that agents are more vulnerable when they have limited knowledge of the possible actions in the environment, which is frequently subject to changes.

- **Development and evaluation of a mapping and offloading technique that uses machine learning to transfer incoming tasks to the appropriate remote server based on expected processing time, required computation, criticality levels, and available resources.**

We have a vehicle list $V = \{v_1, v_2, \dots\}$, and each vehicle v generates several tasks with a deadline, criticality, and size $\tau = \{D, C, SZ\}$. Each vehicle has a task mapping $T_i = \{\tau_{i,1}, \tau_{i,2}, \dots\}$ that checks the availability of resources R from other vehicles, from the nearest roadside units and from the cloud to decide where to send a task τ to execute on a remote server S . Each server has a task scheduling responsible for managing a circular queue Q of processing tasks. Considering the first task τ_1 of this queue, we have a task with a short deadline and a high criticality that starts to be processed. Meanwhile, the circular queue is being reorganized by the task scheduler as new tasks with new deadlines, criticalities, and sizes arrive. So, the second task τ_2 to be processed also has a short deadline, high criticality and is smaller than task τ_1 . Thus, the task scheduler considers

that if the deadline of task τ_2 is smaller than that of task τ_1 ($\tau_{2D} < \tau_{1D}$) and the size of task τ_2 is smaller than that of task τ_1 ($\tau_{2SZ} < \tau_{1SZ}$), the task τ_2 starts to be processed while τ_1 is on hold, back in the queue Q again. After processing task τ_2 , task τ_1 returns to be processed. Then, the remote server S returns the processed task τ_2 to the requesting node.

- **Design and evaluation of uncertainty measurement in predicting events by ML-based approaches. Furthermore, the use of this quantified uncertainty as a technique to help explain the actions of autonomous agents and understand their behaviour.**

We are considering an autonomous system \mathcal{S} that interacts with an environment E with a set of factors $F = F_1, F_2, \dots, F_m$ influencing the perception of the system according to a set of available decisions $D = D_1, D_2, \dots, D_n$. Given that information, we have the uncertainty u definition as $u : K \rightarrow \mathbb{R}$, where K represents a set of all possible levels of knowledge that the system can have about the environment, and \mathbb{R} denotes the set of real numbers. Thus, the greater the knowledge, the lesser the uncertainty: for all $K_i, K_j \in K$, if $K_i < K_j$, then $u(K_i) > u(K_j)$.

Due to the related above, we propose incorporating uncertainty into [Explainable Artificial Intelligence \(XAI\)](#) for autonomous systems combining Bayesian deep learning and uncertainty-aware planning for interpretable and transparent decision-making processes. Thus, we aim to implement visual indicators that clearly show how the autonomous agent perceives its environment. Further-

more, these indicators should not only identify the factors that influence the agent’s decision-making process but also demonstrate the impact of uncertainty on its choices. Therefore, the indicators must provide absolute clarity.

1.4 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 offers an introduction to context definition, RL, Gaussian process, OU process, uncertainty in events, sensitivity analysis, XAI, centralized and distributed approach, and platoon. Additionally, we present the key findings of the related works relevant to this research. Chapter 3 proposes the Autonomous Connected Real-Time (ACT) Framework. This framework consists of three central cores, which are described in detail in the following chapters: Context Awareness for Autonomous Systems, Task Management in Autonomous Systems, and Learning and Performance Monitoring of Autonomous Systems. In Chapter 4, we discuss the concept of context awareness for autonomous systems. We explain the benefits of using noise to encourage exploration and provide experiments highlighting the advantages of incorporating this noise in training autonomous agents. In Chapter 5, we discuss the challenges of managing tasks in autonomous systems. We thoroughly examine centralized and distributed approaches to task mapping. Chapter 6 explains how to measure uncertainty in predicting events for autonomous systems. This uncertainty can help track the learning progress of these systems and provide insight and explanations into their actions and behaviour.

Chapter 7 outlines the primary outcomes and debates regarding the three main components that constitute our ACT Framework. Chapter 8 presents the conclusions, limitations and plans for future work.

Chapter 2. Background and Related Work

In this chapter, we will explore important concepts and definitions that are essential for understanding and discussing the research presented in this thesis. Our research is divided into three main areas, each with its own background information and primary works related to our study. These areas are:

1. Context awareness for autonomous real-time vehicle systems: This core addresses how autonomous systems perceive their surroundings and utilize contextual information to make decisions.
2. Collaborative task management for autonomous real-time vehicle systems: This core focuses on tasks generated by autonomous systems and how these systems handle them, considering factors such as deadlines, resource limitations, and latency.
3. Learning and continuous monitoring for autonomous real-time vehicle systems: After autonomous systems have acquired the necessary contextual knowledge for appropriate decision-making and the ability to manage and allocate their tasks effectively and efficiently, this core considers how we can ensure that these systems take uncertainties present in the environment into account for decision-making. Furthermore, it emphasizes the importance of ensuring that these decisions align with the system's understanding of the environment and meet human expectations.

2.1 Context awareness for autonomous real-time vehicle systems

This section discusses the context and contextual information regarding Chapter 4 - Context Awareness for Autonomous Systems. We also cover the concepts of [RL](#), Gaussian, and [OU](#) processes, which introduce uncertainty into the autonomous agent. We also present the primary descriptions of related works.

2.1.1 Context Awareness

In order to understand context, we refer to the definitions of Dey [\[14\]](#) and Coutaz [\[15\]](#). According to Dey [\[14\]](#), context refers to any information that can help describe the situation of a person, place, or object that is relevant to the interaction between a user and an application. This information also includes the user as well as the application itself. Therefore, a context-aware system, as defined by Dey [\[14\]](#), utilizes context to provide valuable information and/or services to the user, with the value being based on the user's intentions.

According to [\[15\]](#), context refers to a collection of details that includes objects, information, functions, and roles of entities, as well as the relationships and situations they are involved in. This information is crucial for systems to operate effectively in ever-changing environments.

In this research, the term “context” refers to all the information that defines the state of an autonomous system at a specific time and place. For instance, if we consider an autonomous vehicle, its context would include its location, speed, and the conditions of the road it is driving on. It would also encompass information about other agents present in the same environment, such as other vehicles, road signs, and weather conditions that may affect the vehicle’s state.

In [16], the authors demonstrate a lane-keeping assist scenario where an autonomous vehicle is trained and tested using deep reinforcement learning. The study categorizes the environment into two types: discrete actions and continuous actions. However, the research does not examine how well the trained agent interacts with other agents that are also in the same environment. Additionally, the study does not showcase the agent’s performance in environments other than the one it was trained in, which would provide insight into the agent’s capability to handle and adjust to different contexts.

2.1.2 Reinforcement learning

RL is related to the [Markov Decision Process \(MDP\)](#). In the [MDP](#), the action space A can be discrete ($A = 1, \dots, 100$) or continuous ($A = [-1, 1]$). For both the discrete and continuous action spaces, an agent in the state $s_t \in S$ acts in $\in A$, receives a reward $r(s_t, a_t) \in R$ and moves to the next state $s_t + 1$.

A [Partially Observable Markov Decision Process \(POMDP\)](#) is a generalization

of the MDP in which the current system state is not necessarily known. Instead, the decision maker remembers the decisions taken and the observations noticed over time, and tries to use that information to take the next decision. It is possible that, for example, instead of the “current system state”, a probability about the states is maintained while the decisions are taken [17]. This model is already being used to solve various problems, such as robot navigation, elevator controls, military applications, medical diagnostics, and education [18].

RL has long struggled with the issue of exploration in different environments. As a result, many studies have looked into ways to encourage exploration during agent training. One such study [19] explored combining an actor-critical reinforcement learning approach with a trajectory optimization model-based method, which optimizes exploratory noise by creating a trajectory from the current state to ideal future states. This is done using value functions learned by an RL algorithm. However, projecting images into a latent embedding space can be challenging and may result in the loss of relevant information.

A POMDP [17] is defined as a tuple: $(S, A, T, R, \Omega, O, \gamma)$, where:

- S is a set of states in which the process can be part of;
- A is a set of actions that may be executed in different decision moments;
- $T : S \times A \times S \mapsto [0; 1]$ is a function that returns the probability of the system going to state s' , given that it was at state s and the executed action was a ;

- $R : S \times A \mapsto \mathbb{R}$ is a function that returns the cost (or reward) for taking a decision when the process is at state s ;
- Ω is a set of observations that are obtained on every decision moment;
- $O : S \times A \times \Omega \mapsto [0, 1]$ is a function that returns the probability of an observation being verified, given the state s and last executed action.
- $\gamma \in [0, 1]$ is the discount factor.

In reality, it is not always possible to be fully aware of situations. Our agent operates contextually by gathering and interpreting information from the environment. To make informed assumptions, the agent must compare and contrast past events to identify patterns and make the best decisions. Therefore, it requires a memory state that determines which information to use at each timestep.

Exploration efficiency is a topic that has attracted the attention of countless researchers who seek to encourage their autonomous agents to experiment with global exploration strategies at higher levels. In that recent research [20], the authors conducted a study that shows that the combination of fast and slow rewards in different contexts, including those with noisy inputs, can be a promising method for high-level exploration in reinforcement learning. The authors prove that noise combined with other mechanisms helps the agent explore more efficiently and robustly. The evaluation of that work does not include real-world tasks, and it only considers video game environments.

A few other works seek to improve exploration through the diversification of states. Skew-fit [21] makes a weighted probability distribution of actions based on maximum entropy so that rare states receive higher weights. Thus the authors seek to encourage agent exploration by maximizing entropy, not combining it with other mechanisms.

2.1.3 Gaussian Process

According to the paper [22], a Gaussian process can be defined by a collection of random variables that follows a normal Gaussian distribution. Thus, the following Equation (2.1) may be used to define it:

$$G_{noise} \sim GP(m(x), K(x, x')) \quad (2.1)$$

Where $m(x)$ is the function that informs the mean at any point in the input space, and $K(x, x')$ represents the function that defines the covariance between the points. The mean can be any value, and the covariance matrix must be positive.

2.1.4 Ornstein-Uhlenbeck Process

The *Ornstein-Uhlenbeck* [23] process is responsible for generating temporally correlated noise with zero means, and it is defined according to the following [Stochastic Differential Equation \(SDE\)](#) [24]:

$$OU_{noise} \sim \alpha(\mu - X_t)dt + \sigma dW_t \quad (2.2)$$

dW_t is the Brownian motion scaled by volatility σ , which also controls the amount of noise. Also, $(\alpha \geq 0)$, μ and $(\sigma > 0)$ are parameters. $X_t \in \mathbb{R}$ and for each t the values X_t follow a normal distribution. We define μ as the mean of the process (usually 0), α is the speed (how fast α varies with noise) of the mean reversion scaling the distance between X_t , and μ .

The study “Using the Ornstein-Uhlenbeck Process for Random Exploration”, developed in [25], proposes using the OU process to generate random exploration. The paper disposes of an approach to creating active agent orientation through the state space by sampling speed instead of displacements. Although the proposal presented also uses the OU process to encourage random exploration, those researchers do not evaluate the difference between the Gaussian and OU noises, and the experiments need more empirical tests. Another similar work is “Noisy Networks for Exploration” [26], but the authors evaluate the proposal only in-game environments and tasks.

2.2 Collaborative task management for autonomous real-time vehicle systems

In this section, we discuss the centralized and distributed approaches along with the vision of the platoon. All these concepts are related to task management in

autonomous systems, covered in depth in Chapter 5.

2.2.1 Centralized approach

Figure 2.1 presents a centralized architecture in that each intelligent system sends its workload to be processed in a digital infrastructure (cloud, fog or edge), which updates the tasks those systems are supposed to perform in real-time.

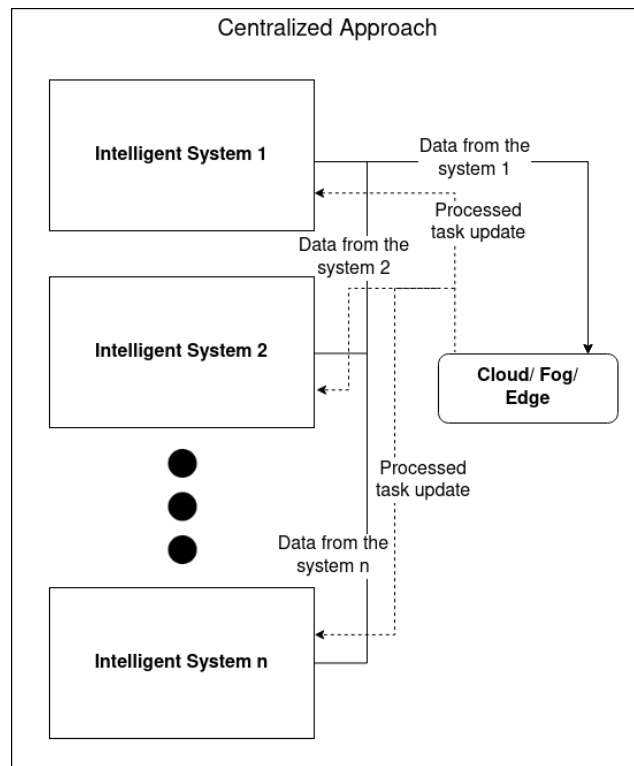


Figure 2.1: Centralized approach for remote workload processing

The features of fog and edge computing have attracted extensive interest from academia and industry, as they significantly reduce the network latency, concentrating computing services closer to consumers [27]. On the other hand, cloud servers have unrestrictedly more extensive processing and storage capacity than in the edge and fog

[27]. Therefore, our system requires low network load, low execution and response times, and occasionally high processing power. Due to these characteristics, it is necessary to decide on the processing target device based on the characteristics of each task and the moment it arrives at the orchestrator, as we must consider vehicular network context situations. Thus, this study intends to prioritize edge usage due to previously listed advantages. Also, a recent study [28] shows that edge-offloaded services can emit less CO₂, which is highly relevant considering the current climate change situation on our planet.

The paper [29] is a work that uses fuzzy logic as a basis for its study. The authors use fuzzy logic to propose an edge computing infrastructure that must orchestrate the workload coming from mobile devices. Furthermore, the investigation in [29] declares that fuzzy logic is responsible for orchestration actions that consider the requirements of the network, computation, and tasks to decide where to execute the tasks in their proposal.

Although several works [29]–[32] use fuzzy logic as a basis, the difficulties in establishing rules correctly, the need to perform numerous simulations and tests, and also the fact that there are no precise mathematical definitions [33] are disadvantages that need to be pointed out. Due to this, works such as the ones developed in [34]–[36] highlight the importance of approaches focused on intelligent offloading. Furthermore, the authors of [34]–[36] emphasize that due to random and uncertain contexts that vary over time, decision-making scenarios have elevated complexity to be optimized using traditional approaches, such as game theory and fuzzy logic. Consequently,

that study recommends using artificial intelligence based on machine learning for multi-access edge computing problems, with the justification that the edge network can self-optimize and self-adapt in the constitution of an intelligent decision-making system.

The study [37] uses machine learning as the basis of a workload orchestrator for vehicular edge computing work. That study uses supervised learning and two-stage architecture for the orchestrator’s decision-making. In the first step, a classification indicates whether the target device has a chance of success or failure. Thus, only for cases where there is a prediction of success, the orchestrator moves to the second ML step, which estimates the service time in that target device. In this way, the orchestrator chooses the remote server with the lowest expected service time. Unfortunately, despite using machine learning, the research work [37] becomes very extensive and expensive since it is necessary to train six different models with different input features for each of those models.

2.2.2 Distributed approach

Real-time intelligent systems are required to process and interpret numerous data simultaneously, demanding high-performance computing and rigorous real-time responses to requested tasks. In the case of autonomous vehicles, we deal with limiting factors such as battery life and constricted computing power that restricts the processing load supported onboard. In this scenario, we must take advantage of external devices that can support the processing performed by the intelligent system

in real-time. Furthermore, with over-the-air communication, vehicles can share their information with each other, in addition to using each other’s processing capacity to perform possible tasks. The execution of tasks outside the origin node can also be done using edge, fog or cloud resources. For example, Figure 2.2 presents a decentralized task processing architecture in a car platoon scenario.

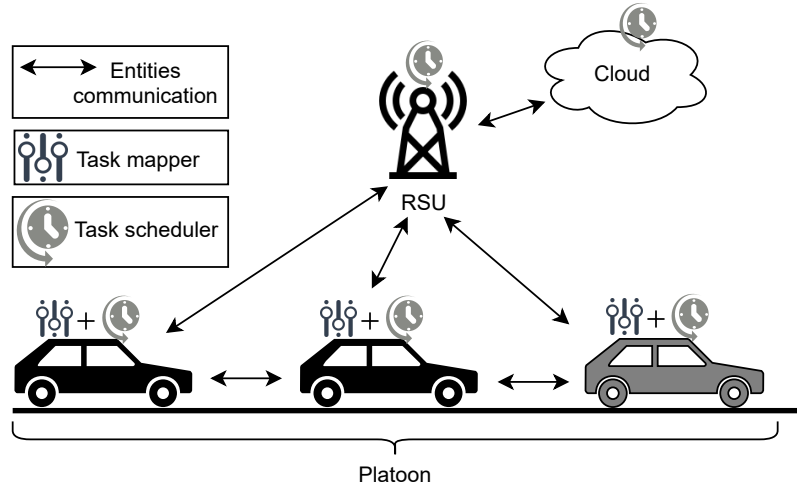


Figure 2.2: Over-the-air task processing in a platooning scenario

2.2.3 Platooning

As an example of distributed approaches, we have the platoons that are essential to improve vehicle traffic and commute time. On the platoon, the cars follow a lead vehicle at a close distance, with each car finding its optimal position to better account for traffic conditions [38]. As shown in Figure 2.2, we have a platoon scenario where the lead vehicle is the gray car in front of everyone.

In addition, other approaches are widely used, such as Round-Robin processing. The papers [39] and [40] make use of this technique for task scheduling in vehicle

platooning scenarios. In that approach, a precisely equal slice of time is given to each task in turns, so a task that requires more processing time will have to go through the scheduler several times until its processing is fully completed. In this context, tasks are placed in a circular queue, where all are treated without taking into account their priority, having to wait in turns for the necessary CPU time slot.

2.3 Learning and continuous monitoring for autonomous real-time vehicle systems

This section discusses the definition of uncertainty in events triggered by autonomous agents. We also explore how a sensitivity analysis mechanism can support identifying which inputs of the agent’s perception most influence the variation of this uncertainty. Furthermore, we provide an explanation of what [XAI](#) is and its purpose. These concepts are related to the core of Learning and performance monitoring of autonomous systems discussed in [Chapter 6](#).

2.3.1 Uncertainty in events

An event refers to all occurrences that take place in a particular location over a specific duration. These events can be categorized into two groups: primitive and derived or complex. This classification is outlined in [\[41\]](#). For example, the data collected from sensors, also known as sensor readings, are considered primitive events. On the other hand, complex or derived events are created by processing primitive events based on [Complex Event Processing \(CEP\)](#) rules, also known as Event Patterns.

When we want to trigger an event, whether it is simple or complex, we need to acknowledge that there is always some uncertainty about whether the event will occur. This uncertainty is present even in reliable measurements like rulers, clocks, and thermometers [42]. To address this uncertainty, we store event predictions in a list for every one-second time window. Then, we calculate the list’s mean and standard deviation every second. The mean represents the value that characterizes the triggered event, while the standard deviation represents the uncertainty [42], [43]. The standard deviation measures how much the forecasts are dispersed around the mean, so a larger standard deviation implies more significant uncertainty about the event. We can calculate the uncertainty using Equation 2.3:

$$uncertainty = \frac{s}{\sqrt{n}} \tag{2.3}$$

where s is the standard deviation, and n is the number of event predictions stored in the list.

The authors of “Complex Event Processing Over Uncertain Data” [44] present a method that considers the production of new events in the face of uncertainty. They have extended the complex event processing literature to manage data with uncertainty. However, to our understanding, the authors of the work [44] do not present the probability (which would indicate the degree of uncertainty) of the new event produced occurring in the final result.

The work [45] states as a contribution the proposal of an input representation

with values of predicted visual abstractions and also confidence values related to that prediction. Furthermore, the authors promise to capture uncertainty using images representing the vehicle’s acting environment, displaying probabilistic patterns to better deal with inconsistencies. That work performs with uncertainty as a measure produced and provided by perception systems. The authors modelled observation (o) as a pair of values $o = (\hat{s}, c)$, where \hat{s} is the estimated state and c is the confidence levels related to these estimated states.

The research [45] uses imitation learning to train a vehicle model in the Carla simulator and verify the success rate in tests performed with a described benchmark. The proposal was to train the model with noisy data, representing the levels of uncertainty in the sensing. The model was tested with inputs that contained the representation of uncertainty through the insertion of confidence levels in the data that illustrated the observation performed. Also, the authors claim that their proposal can be used to handle false positives better. However, at no point in the article did the authors expose how the data acquired from the Carla simulator were treated to fit and adapt to the input format they proposed. Likewise, they do not detail how the uncertainty was extracted from the simulator to be inserted into each observation made.

2.3.2 Sensitivity analysis

Sensitivity analysis is critical because it indicates how much each uncertain parameter contributed to generating the output uncertainty. Then, we chose a variance-based sensitivity analysis to determine how much the input variation influences the output

[46], [47]. Thus, we use the first-order Sobol indices. These indices identify the input parameters with the most significant effect on output variability [48]–[50].

Sobol’s first-order sensitivity index is given by:

$$S_i = \frac{Var(E(H | \mathbf{P}))}{Var(H)} \quad (2.4)$$

Where S_i is the sensitivity index, H is the output resulting from the model with uncertain input parameters list \mathbf{P} . Thus, if we have a low sensitivity index, which can vary with the range $[0, 1]$, we will have that the variance in this parameter will have little effect on the variance of the final result. Therefore, if the parameter’s sensitivity index is high, any variation in this parameter will significantly affect the model’s output.

The total Sobol index for a parameter belonging to \mathbf{P} is defined according to the following equation:

$$S_{\tau i} = 1 - \frac{Var(E(H | P_{-n}))}{Var(H)} \quad (2.5)$$

We have that $Var(E(H | P_{-n}))$ represents the variance of the expected value of the output considering the simultaneous variation of all uncertain parameters of the set \mathbf{P} except for P_n . If we have $S_{\tau i} = 0$, we say that the variability of P_n has no influence on the results and can be ignored in future analyses.

In the work “Complex Event Processing over Uncertain Data Streams” [41], the authors work with one processing engine extension to support the complex events processing in the face of uncertain data. The research also defined a model of how the input and output data should be. In addition, the researchers calculated and presented the confidence probability in the complex event’s output. In our opinion, the definition of what the input data should look like tightens the input data space, limiting the data that can be used as input to produce complex events. Another difference in our approach to that work is no analysis in [41] of how much input data contribute to the output uncertainty.

2.3.3 Explainable Artificial Intelligence (XAI)

As the complexity of decision algorithms increases, AI agents become more powerful, but their decisions also become more challenging to understand. Therefore, it has increased interest in developing explainable, transparent, and interpretable AI models [51]–[53]. Thus, XAI is a branch of AI that seeks to make the decisions and processes taken by machine learning algorithms more understandable and transparent to humans. This is especially important in critical or high-risk situations, like autonomous vehicles, where understanding how the system works is crucial to ensure user safety and trust.

The paper [54] highlights the importance of artificial intelligence models having high interpretability, in addition to the ability to estimate uncertainty through uncertainty wrapper frameworks. In this way, the research stresses the use of decision

tree structures to support interpretability. In addition, however, it emphasizes that small changes in a single model factor can result in significant changes in uncertainty estimation. With the argument defined above, the authors selected some approaches to smooth the transitions between different levels of uncertainty with minimal reduction of the model’s interpretability. Furthermore, the authors intend to control the increase in runtime complexity at low levels while ensuring that the uncertainty estimation performance will not be adversely affected. The approaches chosen were Random Forests, Fuzzy Decision Trees, Fuzzy Random Forests, Soft Decision Trees, and Bagged Soft Decision Trees. The paper seeks to answer the following question: ”How does the uncertainty estimation performance of the uncertainty wrapper frameworks differ when softening approaches are used instead of a decision-tree-based approach?” [54].

As a use case, that article [54] considered pedestrian detection by cameras of moving vehicles. The research also used the Carla simulator to define a vehicle travelling at different times of the day in an urban traffic environment. The objective was to locate pedestrians within a maximum distance of 25 meters from the vehicle under different weather conditions. As an evaluation metric, the authors mainly considered the Brier score. In conclusion, the authors could not highlight a single approach that satisfied all the requirements listed by them. Thus, it is necessary to consider the context of the use of the application to define the best approach to employ.

The work proposed in [55] brings an approach to explain the behaviour (action) of an agent in a given situation (state). For this, the authors propose explaining an

agent’s decisions through similar situations in the agent’s training trajectory. That is, the authors aim to identify which past experiences led the [RL](#) agent to behave in a certain way. However, the authors restrict this approach to offline [RL](#), which has disadvantages compared to online [RL](#) in terms of adaptation to dynamic environments, besides the cost of data collection. In the real world, the agent’s environment can change over time, and the online [RL](#) technique allows the agent to adapt to these changes as it interacts, receives feedback and learns from that environment. The offline [RL](#) agent, however, has its adaptation capacity affected due to the lack of interaction with the environment since, in this case, learning takes place from previously collected data and without direct interaction with the environment. Furthermore, training an offline [RL](#) agent can also be quite expensive because of this need for data collection.

Considering the applicability of [RL](#) algorithms to several real-world problems, the authors of [\[56\]](#) highlight the challenges in explaining the behaviour of [RL](#) agents, who autonomously learn to operate in their environment. From this, the paper [\[56\]](#) proposes using formal model transforms to generate explanations that justify the actions of the [RL](#) agent. The proposed framework involves three entities with specific roles. First, an actor, the [RL](#) agent itself, seeks to maximize its accumulated reward in interactions with the environment. Then there is an observer, the one to whom the explanations will be directed, so this observer expects the actor to behave in a certain way and follow a specific policy. Finally, there is an explainer, an agent responsible for finding explanations for the actor’s actions. The explainer then seeks a

sequence of transformations to be applied to the environment so that the actor’s policy in the transformed environment aligns with the policy anticipated by the observer. However, those produced explanations need to be presented symbolically or intuitively for human observers with non-specialized or with a minor specialization in developing autonomous agents. Therefore, the explanation process in that work is more related to the fact that an algorithm (observer) can understand the behaviour and actions of another algorithm (actor) in an environment through the mediation of a third algorithm (explainer).

Another widely used resource for explaining the actions of **RL** agents is the salience maps applied to the characteristics of specific scenarios. For example, the work by [57] proposes an approach called **Specific and Relevant Feature Attribution (SARFA)** to generate saliency maps to explain the actions of **RL** agents, ensuring the highlighting of only specific and relevant features to the action to be explained. However, the authors point out that, even with the use of salience maps to explain the decisions taken by the model in a specific scenario, their approach needs more information to explain the general behaviour of the model. Thus, some limitations of **SARFA** are related to the possibility of invalid states outside the allowed state space, as the method inserts disturbances in the agent’s state to produce the salience maps. Furthermore, the proposal must consider the dependency and correlation between the analyzed features, which, if not done, can result in inaccurate saliency maps.

Considering the complexity of environments in the interaction of autonomous cars and the need for safety and reliability in the vehicle, interpretability is an essential

factor in the composition of transparency in the decisions taken by the system. Based on this, one of the paper’s contributions [58] is training an intelligent driving agent with an interpretable environment model. Thus, the authors propose the composition of a bird-view semantic mask to graphically explain how the agent interprets and understands the environment in which it operates. However, even with the visual presentation of interpretable explanations about how the agent understands the environment, the proposal also needs to provide an explanation of how decisions are made, knowing that the machine learning algorithm of that work is model-free, which implies the absence of a policy. Therefore, among the information provided as output, there is no way to intuit the reasons that led the agent to take a specific action over another.

2.4 Overview of related works

Table 2.1 outlines the main related works. In addition, the last column of Table 2.1 highlights the limitations found when comparing these related works to our research.

Table 2.1: Summary of related work (N.M=Not Mentioned)

Paper	Main Method	Test Environment	Adaptability to env. changes	Limitations
[16]	Reinforcement Learning	Open Racing Car Simulator (TORCS)	N.M	Interactions with other agents not examined, agent’s performance in new scenarios not showcased

Paper	Main Method	Test Environment	Adaptability to env. changes	Limitations
[19]	Actor-Critical RL approach combined with Trajectory Optimization	DeepMind cheetah environment and the task of inserting a cylinder into the tube	No	Challenges in projecting images into a latent embedding space, loss of relevant information
[25]	Reinforcement Learning with OU Process for Random Exploration	N.M	N.M	Insufficient empirical tests, do not compare Gaussian and OU noises
[26]	Deep Reinforcement Learning	Game Environments	N.M	Evaluation only in-game environments
[20]	Deep Reinforcement Learning with fast and slow rewards	Game Environments	Randomized object locations	Does not include real-world tasks
[21]	Reinforcement Learning with imagined goal	MuJoCo [59] and Real World Visual Door environments	No	Does not combine entropy maximization with other mechanisms
[29]	Fuzzy logic-based	EdgeCloudSim Simulator	Dynamic environment	Fuzzy Logic’s drawbacks in rule establishment, need for additional simulations and tests, lack of precise mathematical definitions
[37]	Supervised-learning	EdgeCloudSim Simulator	Dynamic environment	Requirement to train six different models with different input features for each model
[39]	Round-Robin Processing	Simulation of Urban MObility (SUMO)	Two types of traffic: regular traffic and irregular traffic	Equal treatment of tasks without considering their priority

Paper	Main Method	Test Environment	Adaptability to env. changes	Limitations
[40]	Round-Robin Processing	N.M	N.M	There are no detailed specifications on where the simulations were performed
[44]	Bayesian network and the sampling algorithms	Simulation environment	N.M	No probability indication of the new event occurrence
[45]	Deep convolutional policy network	Carla Simulator	Dynamic obstacles	Lack of detail about data treatment and extraction of uncertainty from the simulator
[41]	Decision trees	N.M	N.M	Restrictive input data definition, lack of analysis of input data contribution to output uncertainty
[54]	Decision trees	Carla Simulator	Other traffic participants (vehicles, trucks, motorcycles, bicycles, and pedestrians) and weather parameters (sun position, cloudiness, precipitation, fog/wind intensity, and road wetness)	No single approach satisfying all presented requirements, context-dependent to choose an approach
[55]	Offline Reinforcement Learning	Grid-world, Seaquest from Atari, HalfCheetah from MuJoCo	N.M	Limited to offline RL, cost of data collection

Paper	Main Method	Test Environment	Adaptability to env. changes	Limitations
[56]	Reinforcement Learning Policy Explanation	Frozen Lake, Taxi, Apple-Picking, Sokoban, Blocks World, Towers of Hanoi, Snake, Rearrangement, Triangle Tireworld, and Exploding Blocks.	N.M	Explanation process more related to algorithm understanding another algorithm's actions
[57]	Deep reinforcement learning	Board games (Chess and Go) and Atari games (Breakout, Pong and Space Invaders)	No	Requirement of more information for explaining general model behaviour, issues with invalid states and correlation between analyzed features
[58]	Latent Deep Reinforcement Learning	Carla Simulator	Different urban scenarios	Absence of explanations of how decisions are made

Chapter 3. Proposed Context-Aware Collaborative Autonomous Real-Time Vehicle Systems Framework

In Chapter 2, we provide a brief overview of the three main proposed components of this dissertation: Context awareness for autonomous real-time vehicle systems, Collaborative task management for autonomous real-time vehicle systems, and Learning and continuous monitoring for autonomous real-time vehicle systems. These three components are combined to create the **Context-Aware Collaborative Autonomous Real-Time Vehicle Systems Framework (CARVS Framework)** figure 5.9 presents the simu, enabling autonomous systems to act and make real-time decisions based on their context and task management. The following Fig. 3.1 shows in detail the composition and operation of **CARVS Framework**.

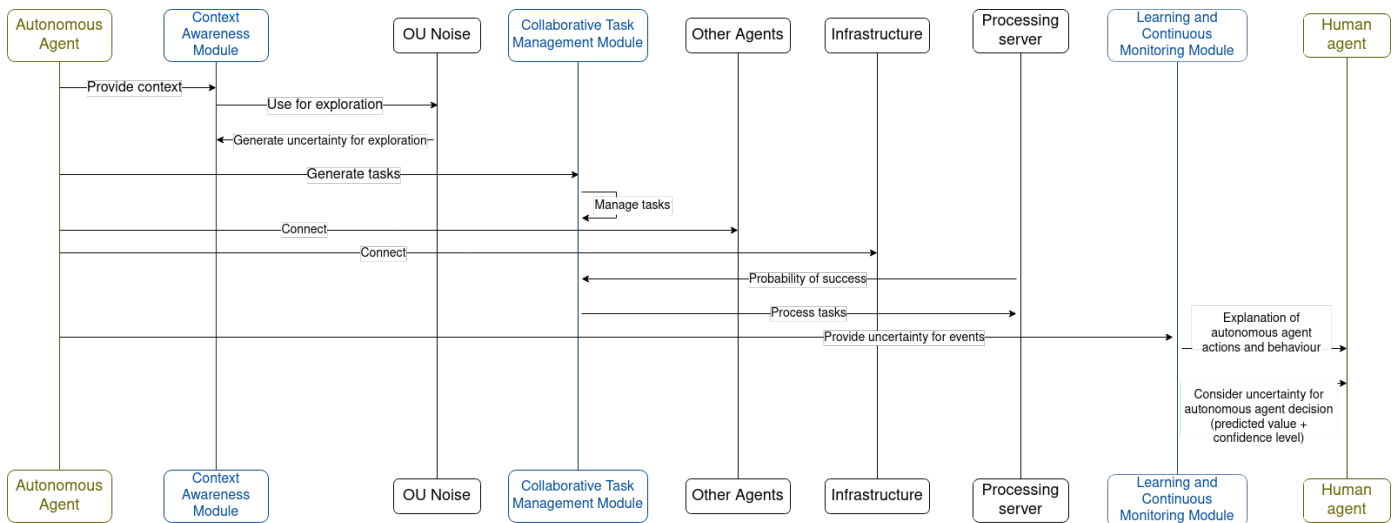


Figure 3.1: Context-Aware Collaborative Autonomous Real-Time Vehicle Systems (CARVS) Framework

Analyzing the Fig. 3.1, the connected autonomous agent provides context to the context awareness module in real-time. This module then uses **OU** noise to create

uncertainty, which helps the agent explore more. The autonomous agent generates tasks that the collaborative task management module handles. Also, the autonomous agent connects to other agents and the infrastructure. Then, the processing server gives the collaborative task management module the likelihood of success so it can choose which server should process each task. The Autonomous Agent shares the uncertainty it senses with the learning and continuous monitoring module, which explains the autonomous agent’s actions and behaviours to the human agent. Finally, the learning and continuous monitoring module also considers the uncertainty for the autonomous agent’s decision-making (predicted value + confidence level) and shares this information with the human agent.

This dissertation is based on three modules, as shown in Figure 3.1. The context awareness module receives raw data from the autonomous vehicle’s sensors and provides information about the vehicle’s context. The collaborative task management module handles the order and processing location of tasks generated in the autonomous vehicle. The learning and continuous monitoring module receives information about the agent’s interpretation of the environment, its following planned actions, and the level of uncertainty associated with each action. Afterwards, it presents this information visually on a dashboard that monitors how the autonomous agent perceives and behaves in the environment.

In addition to the modules, the framework also includes several agents. The autonomous agent is the primary agent of the structure, the autonomous vehicle that encompasses the developed modules and performs movement actions in the environ-

ment. Other agents represent different vehicles that can serve as an edge structure for the primary autonomous agent. Also, the human agent monitors the intentions and behaviour of the primary autonomous agent through the dashboard.

Real-time systems must operate under strict time constraints, which can be challenging in rapidly changing environments. The ability to adapt while still meeting time constraints is crucial for safety and success. Autonomous real-time vehicle systems require a strict commitment to time constraints for safety, efficiency, and functionality. They represent the intersection of advanced robotics, artificial intelligence, real-time systems, and automotive engineering. By integrating contextual perception and collaboration between vehicles, infrastructure, and other systems, we can achieve context-aware collaborative autonomous real-time vehicle systems to improve mobility, safety, and transport efficiency.

The [CARVS Framework](#) comprises multiple components and modules, making it susceptible to challenges during implementation and operation in the real world. The framework aims to make real-time decisions based on context, which requires quick processing, low-latency communication, and fast data analysis. Any delay or inefficiency can lead to outdated decisions. Implementing continuous learning can be resource-intensive, requiring constant updating of knowledge and adaptation to new data, thereby straining computing resources and bandwidth. Additionally, as the number of vehicles or users of the system increases, it is vital to ensure that the framework can scale effectively without any performance degradation. Furthermore, given the critical nature of autonomous vehicle systems, rigorous testing and

validation processes are necessary to ensure safety and reliability. Therefore, the framework needs regular and continuous updates to adapt to evolving technologies and new challenges without interrupting ongoing operations.

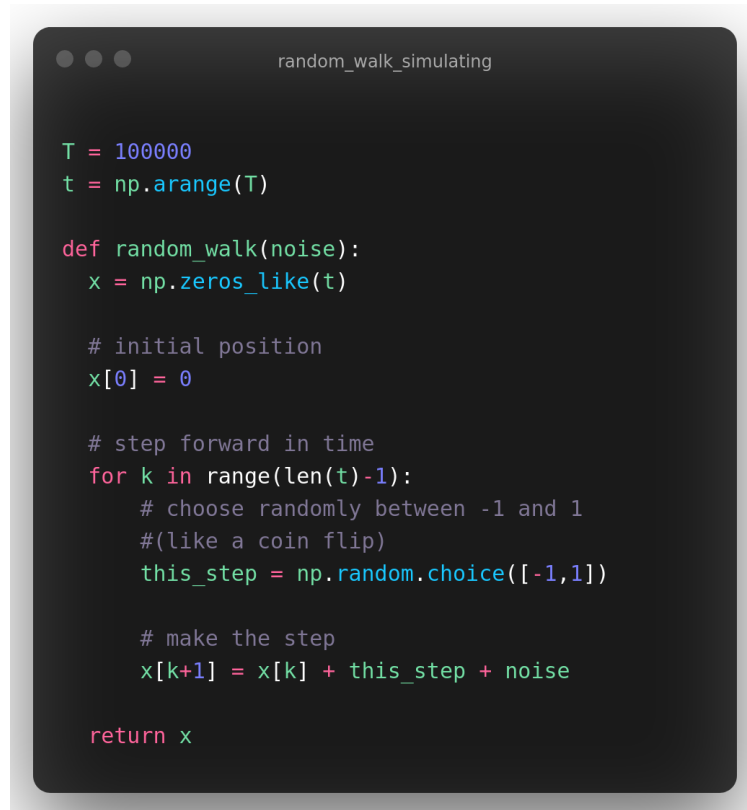
The first step in implementing our framework in a real environment is to select an autonomous agent, such as a prototype car, wheelchair, or domestic robot. Once this is decided, we must ensure the hardware requirements are met. This includes having adequate processing power for support reinforcement learning and real-time processing, sensors and actuators for environmental perception, and over-the-air communication infrastructure.

After the hardware requirements are met, we must also ensure that the software requirements are met. These requirements guarantee task deadlines, autonomous decision-making, and adaptation to new contexts. It is essential to consider the limitations that can affect the framework’s success in the real world, such as network latency, security, and data privacy.

3.1 Context Awareness Module

We assume that the most effective approach for an [RL](#) agent during training is to explore its possibilities by moving randomly. This allows the agent to gain a better understanding of its context and make more informed decisions. The more an agent moves, the better it becomes at exploring its surroundings and becoming context-aware.

To assess the effectiveness of using noises in the random exploration of environments, we conducted a comparison between results from a random walk Fig. 3.2 simulation without noise, with Gaussian noise, and with OU noise.



```
random_walk_simulating

T = 100000
t = np.arange(T)

def random_walk(noise):
    x = np.zeros_like(t)

    # initial position
    x[0] = 0

    # step forward in time
    for k in range(len(t)-1):
        # choose randomly between -1 and 1
        #(like a coin flip)
        this_step = np.random.choice([-1,1])

        # make the step
        x[k+1] = x[k] + this_step + noise

    return x
```

Figure 3.2: Random walk simulation using python

As in Figure 3.2, we start with a one-dimensional random walk. Our agent starts at $x = 0$. Then, by the time, it is like our agent flips a coin, then heads left $\Delta x = -1$ or right $\Delta x = +1$ with some probability based on the noise used. If at time step 1 the result of the coin flip is to head right, then its position at that time step becomes $x_1 = x_0 + \Delta x = 1$. In this way, the agent position at time step $k + 1$ is given by:

$$x_{k+1} = x_k + \Delta x$$

From Figures 3.3 and 3.4, we can infer that the use of OU noise is more suitable to encourage greater exploration of the environment through random steps. In Figure 3.3, for example, the agent with OU noise is the one that most distanced itself from the origin over time. We confirm in Figure 3.4 that the agent with OU noise has a larger interquartile range, indicating greater variability in its position. So the agent with OU noise explored more since it was at more diverse positions during the realization of the random walk. That behaviour is a strong indication of the OU systems model cognitive functions, which include decision-making.

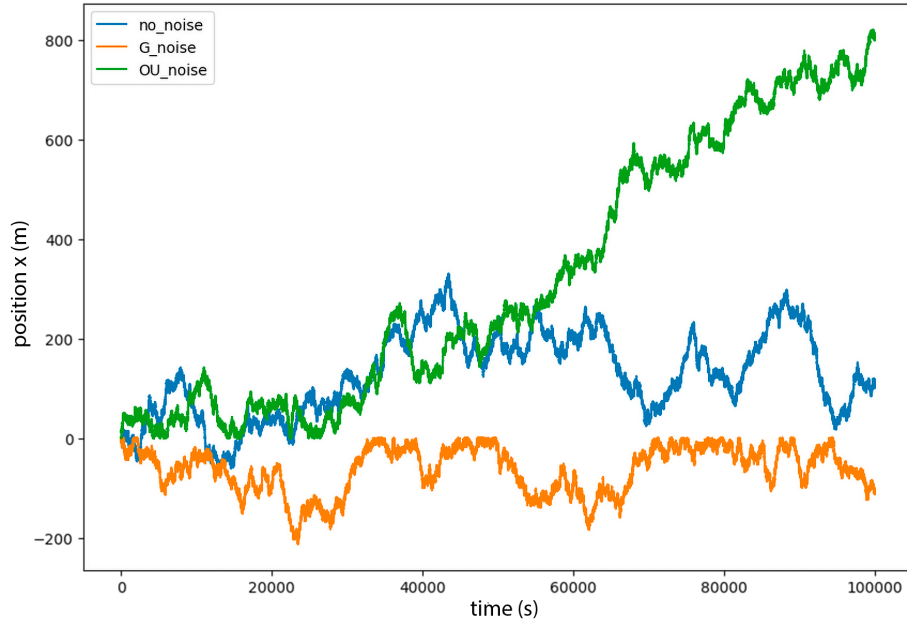


Figure 3.3: Random walk result. Variability of the position x of the agent over time.

Based on experimental testing, the proposed context awareness module shows that OU noise is the most effective method to encourage the random exploration of autonomous agents in their training environments. This approach directly affects the reinforcement learning algorithms, which interact with the environment as they perform actions.

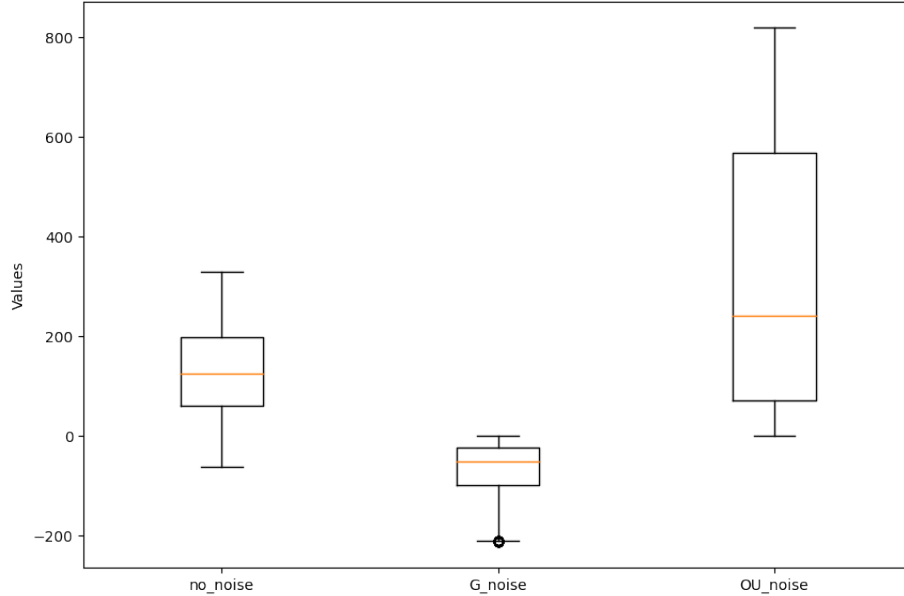


Figure 3.4: Distribution of the agents' displacement data during the random walk.

3.2 Collaborative Task Management Module

In the autonomous real-time vehicle systems scenario, the collaborative task management module plays a crucial role in coordinating safe and accurate decision-making. During the event identification phase, the vehicles generate tasks that need to be processed. The task management module handles and coordinates these tasks, ensuring that the workload is optimized and adequately distributed across the vehicles and infrastructure in the network. This ensures that the vehicles can operate seamlessly and effectively, minimizing the risk of accidents and errors.

One of the biggest challenges for driverless vehicles is ensuring safe operation, considering the time constraints and having to deal with a large amount of daily data. Considering some of the system restrictions, like mobility, bandwidth, and latency,

one of the solutions was the use of cloud infrastructure, as it allows on-demand services and resource scalability [60]. Another alternative is the fog computing paradigm proposed by Cisco Systems researchers [61]. The fog computing proposal emerged amid the mobility, locality, and low latency requirements that extended cloud computing services closer to the network's border on a distributed scale [62].

Vehicle networks have to deal with highly dynamic environments where vehicle dwell time under the covers of a fixed-edge server component, such as a [Roadside Unit \(RSU\)](#), is short. So there is a high demand for data processing and resource sharing with low latency and increased mobility, which gave rise to the [Multi-access Edge Computing \(MEC\)](#) concept. Therefore, the main objective of [MEC](#) is to deploy computing resources closer to end users, with processing and storage performed closer to the origin place, further reducing the latency of requests [63].

Edge computing resources, such as [RSUs](#), are strategically located at specific points along the road. Due to the high mobility of vehicle traffic, we consider a handover approach similar to the one proposed in [64], where the connection and resource are shared among [RSUs](#) to meet vehicle requests that move from one [RSU](#) coverage to another during the processing time of a task.

The collaborative task management module may face several limitations, mainly when vehicles operate in low-density areas or with limited infrastructure. Some concerns are related to network connectivity, latency, coverage area and resource availability. Therefore, to mitigate these limitations, we have fallback strategies such as the autonomous decisions of the module's algorithm, which allows the effective opera-

tion of the vehicle even when isolated. Additionally, Ad Hoc communication between vehicles enables direct resource sharing without infrastructure.

3.3 Learning and Continuous Monitoring Module

In the learning and continuous monitoring module, the vehicle learns to choose actions based on the slightest predicted uncertainty. As autonomous vehicles navigate different environments and situations, they encounter varied scenarios, some of which may not have been anticipated during the initial training phase. The continuous learning module ensures that the vehicle can learn from these new scenarios. Another important aspect of this module is monitoring the quality of decisions made by the autonomous system, identifying anomalies or failures in the system and taking corrective measures or, if necessary, moving the vehicle to a safe operating mode.

Context awareness is crucial for autonomous agents, like autonomous vehicles, which operate in dynamic and uncertain environments. Therefore, it is essential to quantify and consider this uncertainty during the agent's training phase. However, it is not sufficient to simply estimate uncertainty during one phase. Instead, it is essential to continuously monitor and comprehend how the agent handles quantified uncertainty during its decision-making in its environment.

Regarding uncertainty, there are two types: aleatoric and epistemic [65]. The first is related to the input data imperfections, while the last is associated with the lack of data or knowledge and errors in the prediction (model uncertainty). Aleatoric uncer-

tainty may be more relevant when we have tasks with large volumes of data because, in that case, the epistemic uncertainty would be practically invalid due to the amount of information available. On the other hand, in smaller datasets, the quality of these data must be very high, which almost cancels out the aleatoric uncertainty. Therefore, safety-critical application data must be highly qualified for timely information processing. For this reason, we only consider epistemic uncertainty, as this impacts the analysis of the information we want.

Chapter 4. Context awareness for autonomous real-time vehicle systems

To generate an environmental disturbance, we verified that published works [3], [8], [9] use *Ornstein-Uhlenbeck* Process to develop temporally correlated noise for efficient exploration. It means that the *Ornstein-Uhlenbeck* Process produces a noise associated with the previous one to prevent the noise from cancelling or freezing the general training dynamics. Inspired by the studies mentioned above, we use the **OU** process to achieve temporally correlated noise in our approach defined below.

A **POMDP** is a **MDP** generalization, and, in this case, we do not precisely know the current system state. The decision-maker remembers the previous decisions and the observations noticed over time, which helps it to make the next decision. Usually, we do not have a “current system state”, but a probability distribution regarding the states for each action choice [17]. The **POMDP** has immense applicability and is used to model machine maintenance, computer vision, and even behaviour modelling.

Considering that we are working with **POMDP** and our evaluated algorithms are actor-critical based (**Soft Actor Critic (SAC)**, **Proximal Policy Optimization (PPO)** and **A3C**), we have to use stochastic policy for sampling through a probability distribution. So, we define our action sampling like Equation 4.1:

$$\text{Sample action } a' \sim \pi_{\theta}(a'|s') \quad (4.1)$$

Based on a new observation s' , we have to choose an action a' stochastically

from the distribution π . This distribution can indicate the agent’s certainty about its choice. However, this certainty sometimes increases based on local policy and not global, which prevents the agent from better exploring its environment. So, if we generate a disturbance in the distribution π , we will make our agent explore the environment more and, consequently, have higher scores during the evaluation phase. Based on this, we combine the distribution with the [OU](#) process to decrease the agent’s certainty.

$$(\pi_{\theta}(a'|s')) * dX_t \tag{4.2}$$

By multiplying the time-correlated noise, as in [Equation 2.3](#), by the distribution π , as in [Equation 4.1](#), we came up to [Equation 4.2](#). We create a kind of exploration gradient where the agent becomes less confident and explores the environment the more negative the noise is. Besides, depending on the agent’s current position and the observations it receives, the noise may cause more or less disturbance.

We use the three most recent [State-Of-The-Art \(SOTA\)](#) RL algorithms ([SAC](#), [PPO](#) and [A3C](#)) to compose our experiment. We use the [Reinforcement Learning Zoo \(RLzoo\)](#) [\[66\]](#) library as the basis for our implementations and adjust as described in the [SOTA](#) papers [\[67\]](#)–[\[69\]](#). Besides, we also use the [Gym](#) [\[70\]](#) library as a training and evaluation environment for our study. Thus, we chose different settings with diverse tasks and complexity levels, with three environments with continuous action space and discrete action space.

For continuous action space, all algorithms were tested and evaluated. Still, for discrete action ones, only [A3C](#) and [PPO](#) were used, considering that the [SAC](#) version presented in the paper [\[69\]](#) worked only for continuous action space. The first environment used was classic control with the Pendulum task, which has simple continuous action. The goal is to swing it so that it is upright from a random starting position. The second task, more advanced than the first one, was LunarLander from the Box2D environment, whose objective is to land the landing module using four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine. The third is Humanoid from the MuJoCo environment, which has more complex continuous control tasks and whose goal is to control robots as fast as possible.

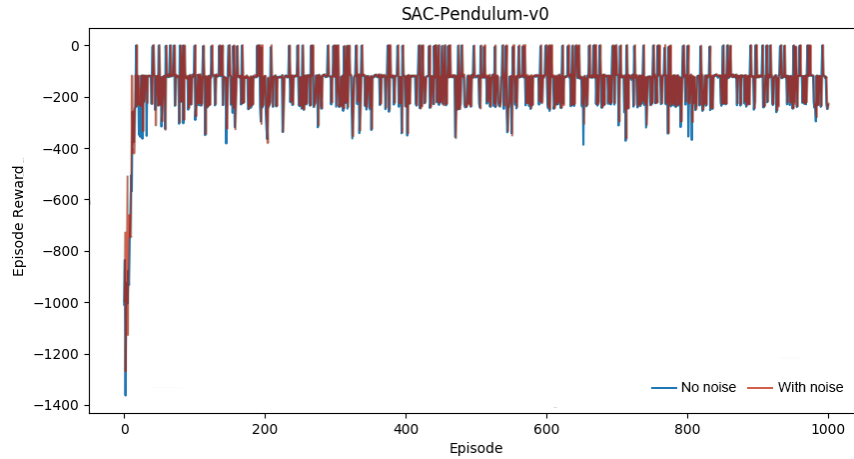
The tasks were run without noise and with the noise of varying parameters per 1000 training episodes and ten evaluation episodes for each experiment. We used a laptop computer with the following specification: AMD Ryzen 7 3800X 3.9 GHz 8-core processor, 32GB DDR4 3600, SSD NVMe 500GB, NVIDIA® GeForce RTX™ 2060 8GB, an Ubuntu-based Linux 20.04 operating system.

We train all algorithms according to the parameters specified in their respective published papers. The first training task was Pendulum-v0, which is a classic problem in the control literature. There is no single, specific solution in this Pendulum task, which means it does not have a specified reward threshold at which it's considered solved. In [Figures 4.1a](#), [4.1b](#), and [4.1c](#), we have the training performance of the studied algorithms with noise and without noise. It is possible to verify that the

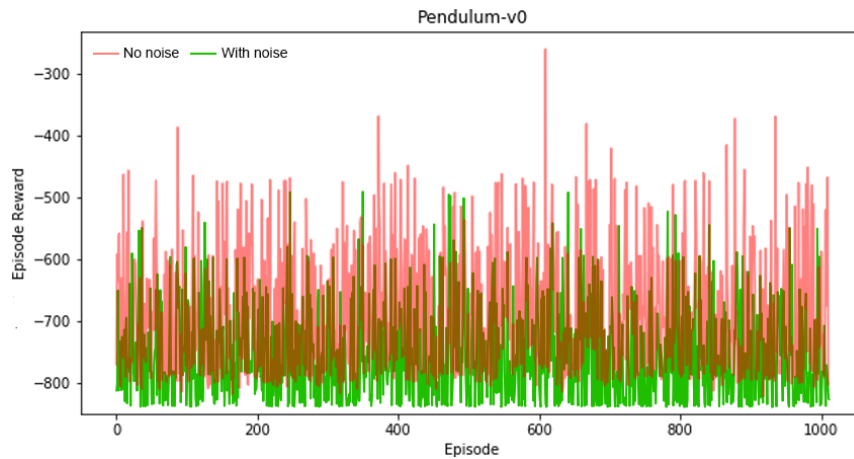
difference between training with and without noise was not very significant in Figures 4.1a and 4.1b. For Figure 4.1c, the performance of training with noise seemed inferior to training without the use of noise. We believe this happens due to task simplicity, as there is not much to be explored due to the environment’s simplicity. Then, the agent is able to explore the entire environment in a few steps without needing the incentive to explore more for this specific task.

The second task, more complex than the first one, was Lunar Lander, which has discrete action space and was executed with PPO and A3C versions, as shown in Figures 4.2b and 4.2a respectively. This task rewards the agent for moving from the top of the screen to the target landing point. The problem is solved through four discrete actions available, and the goal is to get 200 points. Comparing training with and without noise, we do not have a striking difference in those methods’ performance. In 1000 episodes, only the PPO managed to get very close to solving the problem.

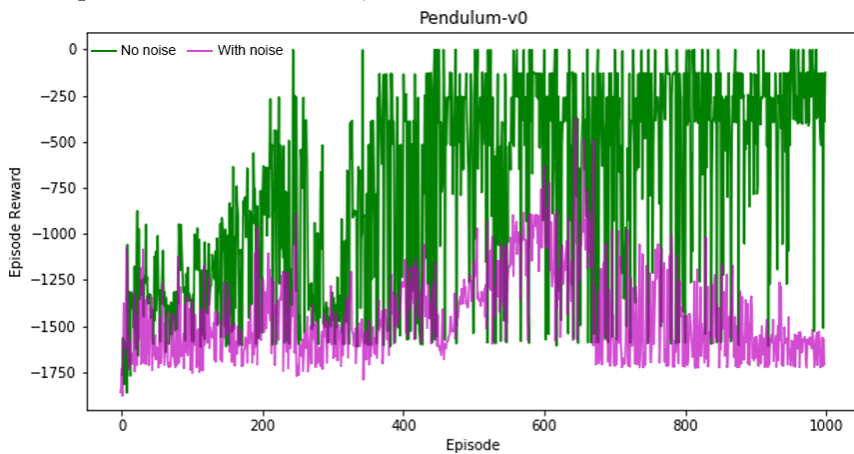
The third task is more complex, with continuous control, and is part of the MuJoCo simulation environment. It is essential to make a three-dimensional bipedal robot walk as fast as possible without falling into that Humanoid task. In training carried out with Humanoid (Figures 4.3a, 4.3c and 4.3b), we were able to notice an expressive performance improvement that the method with noise had compared to the technique without noise in the three tested algorithms. The difference between that task and the previous ones is that it is very relevant, in the Humanoid case, if the environment is explored or not because there are countless alternatives to be done



(a) SAC training performance without noise and with noise. OU noise parameters $\rightarrow \alpha = 0.5; \sigma = 0.5$

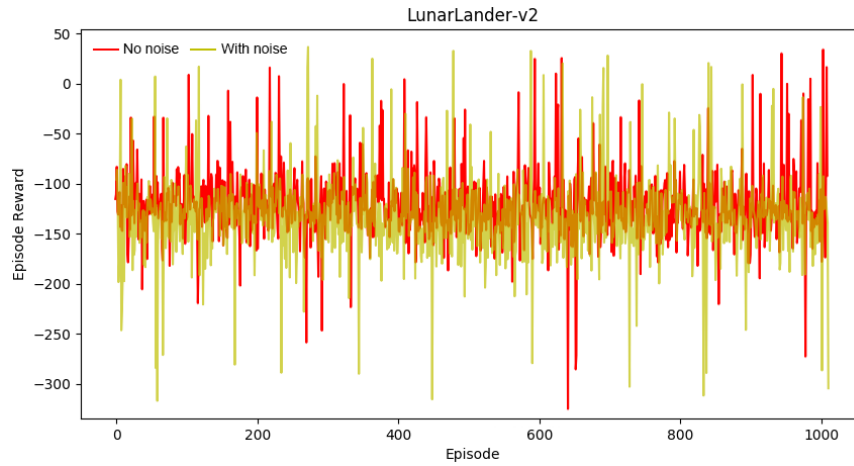


(b) A3C training performance without noise and with noise. OU noise parameters $\rightarrow \alpha = 0.3; \sigma = 0.9$

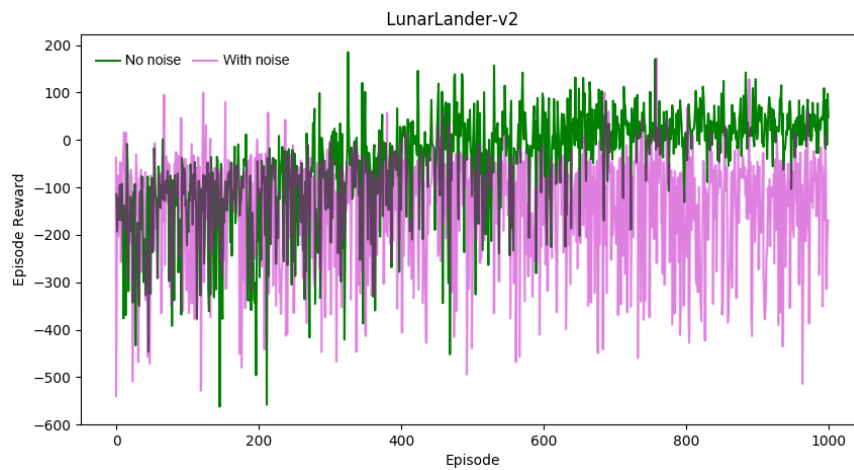


(c) PPO training performance without noise and with noise. OU noise parameters $\rightarrow \alpha = 0.3; \sigma = 0.9$

Figure 4.1: Pendulum-v0 task - training

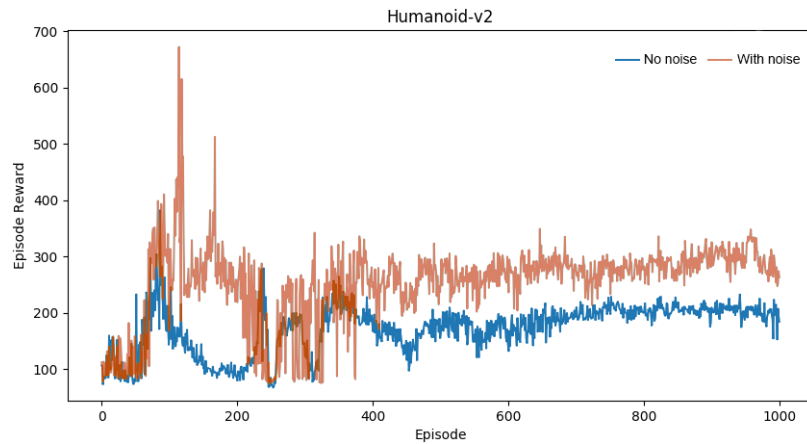


(a) A3C training performance without noise and with noise. OU noise parameters $\rightarrow \alpha = 0.3; \sigma = 0.1$

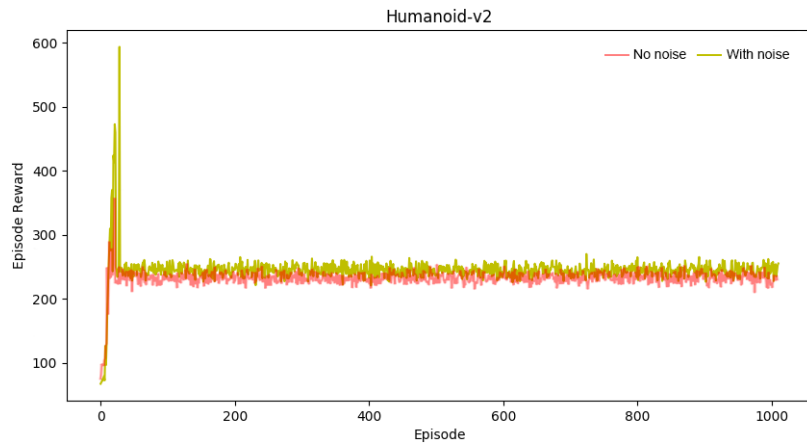


(b) PPO training performance without noise and with noise. OU noise parameters $\rightarrow \alpha = 0.3; \sigma = 0.1$

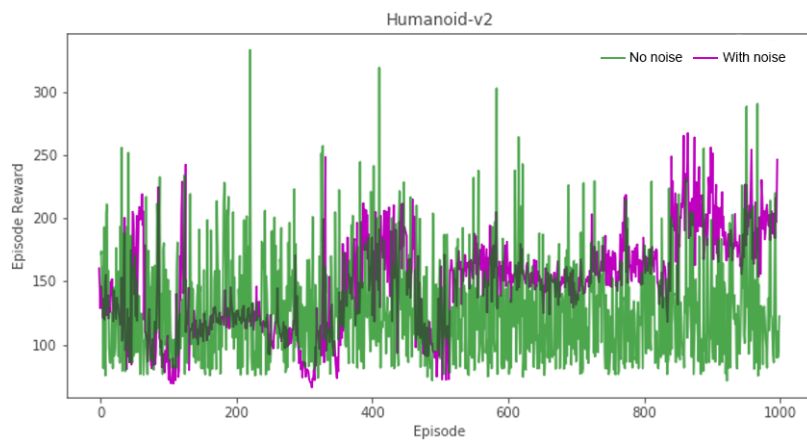
Figure 4.2: LunarLander-v2 task - training



(a) SAC training performance without noise and with noise. OU noise parameters $\rightarrow \alpha = 0.3; \sigma = 0.5$



(b) A3C training performance without noise and with noise. OU noise parameters $\rightarrow \alpha = 0.3; \sigma = 0.5$



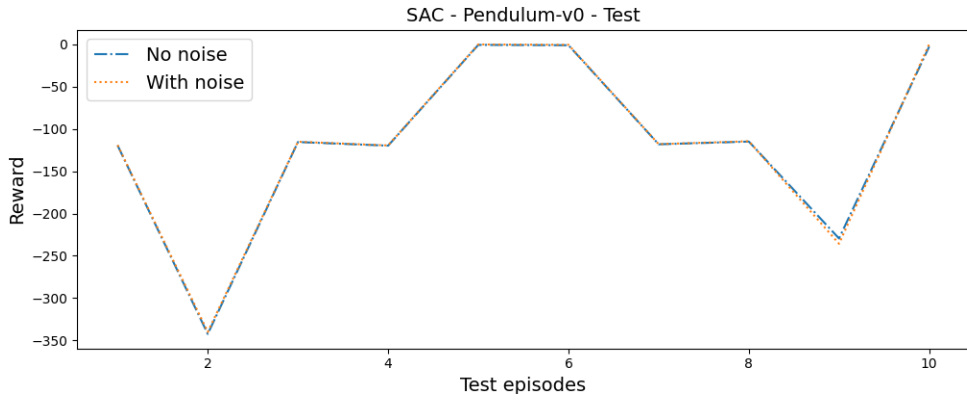
(c) PPO training performance without noise and with noise. OU noise parameters $\rightarrow \alpha = 0.3; \sigma = 0.5$

Figure 4.3: Humanoid-v2 task - training

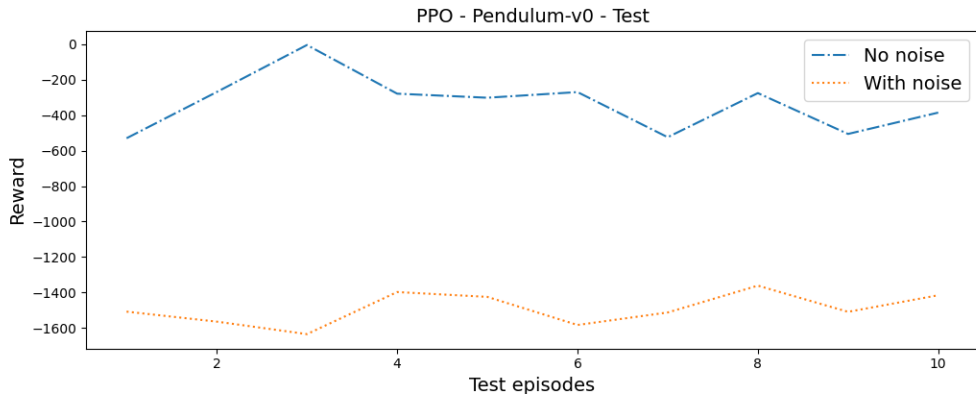
with the robot, where to take it, and there is also no single defined solution.

Figure 4.4 below shows the results of the evaluations performed with each trained method. In this case, the difference between training with noise and without becomes more evident. We can see how much the noise contributed to improving the algorithms' performance for the evaluation phase. As the Pendulum environment was the simplest, there was practically no difference in evaluating the SAC algorithm (4.4a). For the PPO algorithm (4.4b), the noise caused in the Pendulum environment was harmful. And for the A3C method (4.4c) had a slightly better evaluation result for the agent trained with noise.

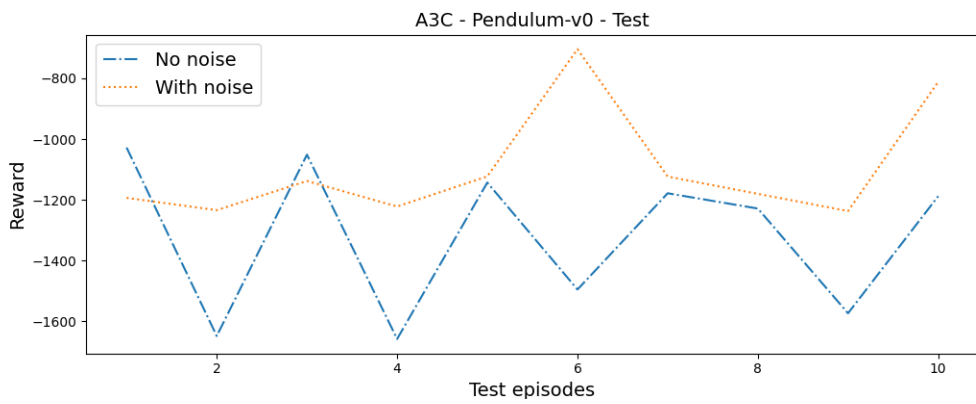
As tasks and the environment become more complex, we realize the positive difference generated in the evaluation of methods trained with noise. We have a significant performance gain with the PPO (4.5a) and A3C (4.5b) algorithms, trained with noise, during the assessments of the LunarLander environment. This gain is even greater in the evaluation of Humanoid (Figures 4.6a, 4.6b and 4.6c) task. We can affirm, based on these results, that the approach proposed by us and performed with the SAC, PPO and A3C methods generates an improvement in the agent's performance in the more complex the task and the environment.



(a) Evaluation tests for SAC algorithm training with and without noise

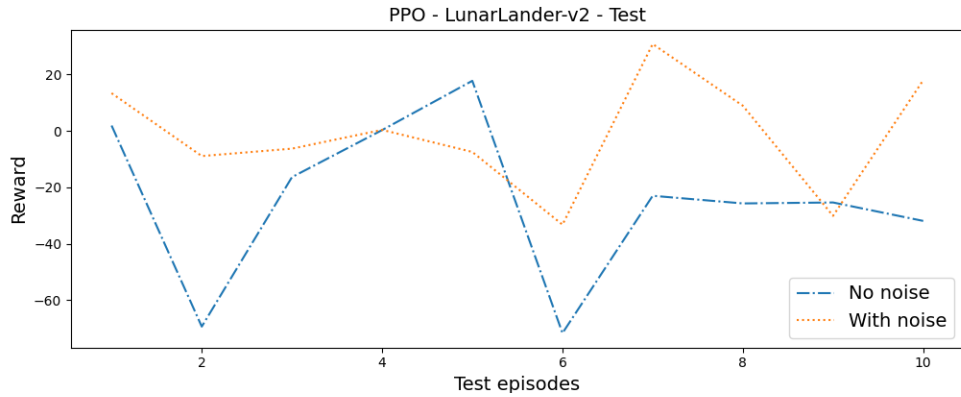


(b) Evaluation tests for PPO algorithm training with and without noise

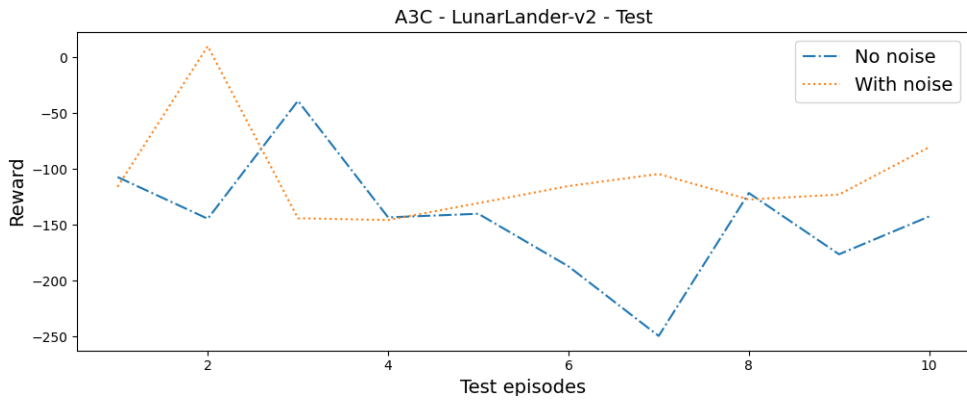


(c) Evaluation tests for A3C algorithm training with and without noise

Figure 4.4: Pendulum-v0 task - evaluation

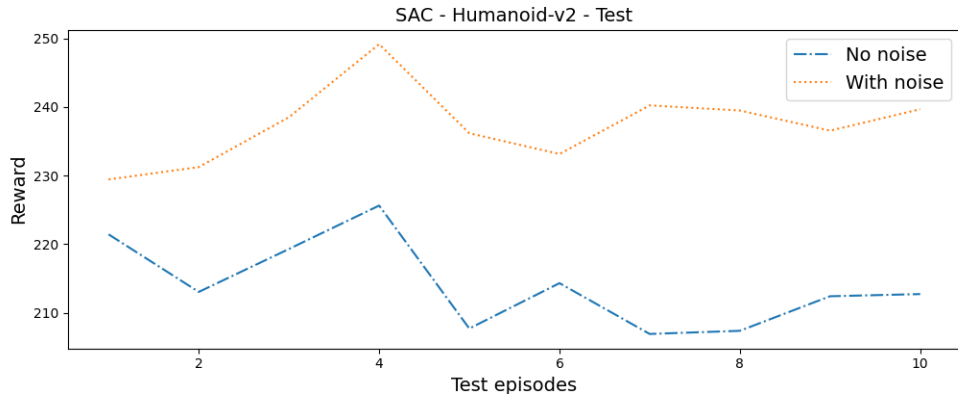


(a) Evaluation tests for PPO algorithm training with and without noise

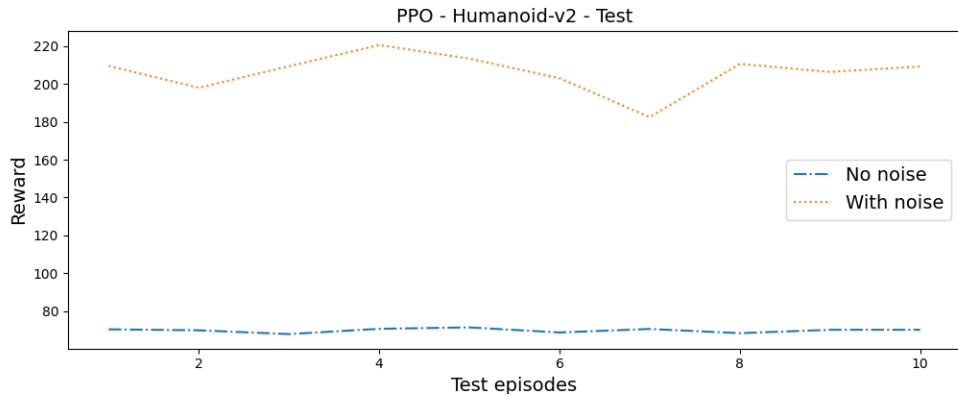


(b) Evaluation tests for A3C algorithm training with and without noise

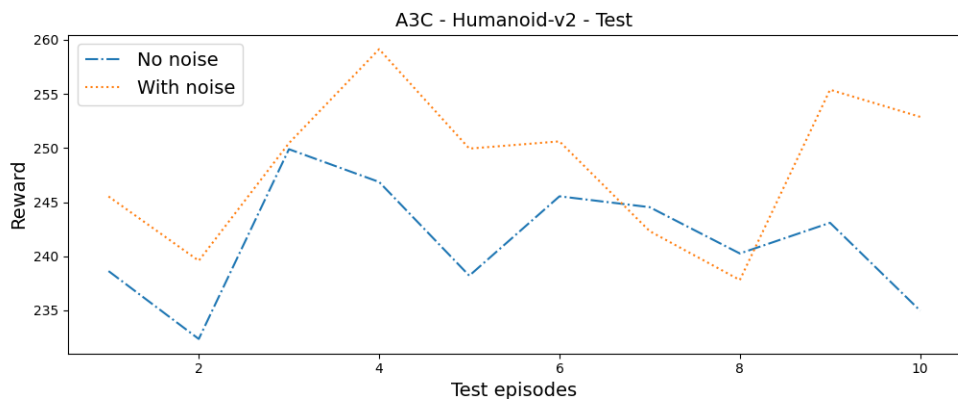
Figure 4.5: LunarLander-v2 task - evaluation



(a) Evaluation tests for SAC algorithm training with and without noise



(b) Evaluation tests for PPO algorithm training with and without noise



(c) Evaluation tests for A3C algorithm training with and without noise

Figure 4.6: Humanoid-v2 task - evaluation

Chapter 5. Collaborative task management for autonomous real-time vehicle systems

We researched how to efficiently handle vehicle data and tasks, considering the multiple devices involved, the constant vehicle movement, and the need for real-time processing. Therefore, we examined two methods, centralized and distributed, for mapping and processing these tasks.

5.1 Centralized Approach

We used the EdgeCloudSim simulator [71] to analyze our centralized approach that uses machine learning-based load orchestration. We have chosen that simulator because it simulates computational and network resources inherent to edge computing and supports cloud computing as well. Therefore, the most vital point of EdgeCloudSim is the simulation of mobile devices, making it possible to simulate [Wireless Local-Area Network \(WLAN\)](#) and [Wide-Area Network \(WAN\)](#) networks. Those main simulator features are essential for us, considering that we are working in the autonomous vehicle domain, which has high dynamicity.

Figure 5.1 presents the workflow of our edge orchestrator. We start with connected autonomous vehicles sending the tasks to the orchestrator. Then the shortest service time to process the task is responsible for defining the node where the task will be processed. After being processed, the task result is sent back to the orchestrator, which sends it back to the autonomous requesting vehicle. Besides workflow, Algorithm 1 brings the pseudocode to detail the edge orchestrator implementation. In this

way, the Algorithm 1 starts with a task as input and initializes available options from remote servers. Then, for each of these options, it estimates the task processing time using a machine learning algorithm. Finally, as the server with the shortest service time is chosen, the algorithm checks which **Virtual Machine (VM)** is closest to the user and returns the selected server as output so that the task can be offloaded to it.

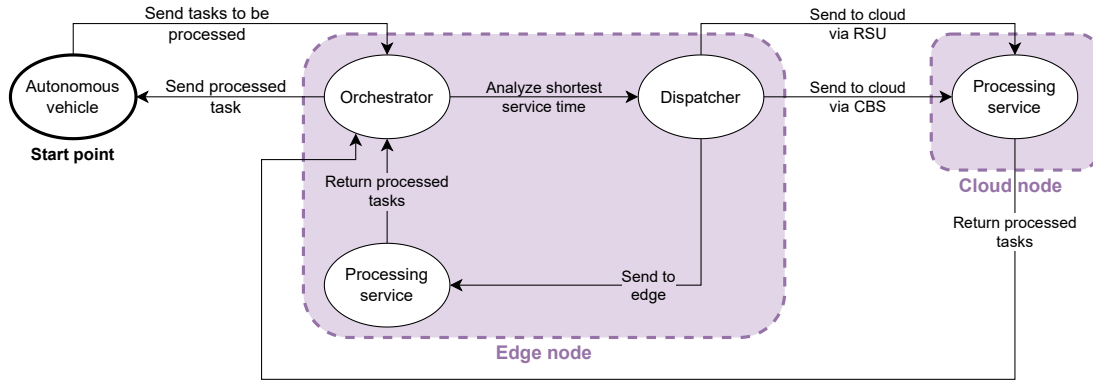


Figure 5.1: Edge orchestrator workflow (RSU = Roadside Units/ CBS = Cellular Base Station)

Since vehicles have several sensors and send different data to a remote server, we need to select which of these data are essential for what is being processed at the moment. Thus, if the vehicle sends a task related to navigation to be processed over-the-air, it does not matter at that moment the data received and related to the infotainment system, for example. Thus, according to our model presented in Figure 5.2, one of the first steps at the beginning of our approach is selecting the most relevant input features using sensitivity analysis.

Initially, we had 12 different inputs to estimate the service time for the chosen server. Thus, as shown in Figure 5.3, we had the following features: Decision, VehicleLocation, SelectedHostID, TaskLength, WANUploadDelay, WANDownloadDelay,

Algorithm 1: How our orchestrator handles task offloading

RSU = Roadside Units

CBS = Cellular Base Station

Output: selectedVM

Input: task

```
1 var device : integer;
2 var selectedVM : VM;
3 var predictedServiceTime : double;

4 Initialize servers options list  $\leftarrow$  [
    EDGE_DATACENTER,
    CLOUD_DATACENTER_VIA_RSU,
    CLOUD_DATACENTER_VIA_CBS];

5 for each server option in list do
6 | predictedServiceTime  $\leftarrow$  ML_model(selected_input_features);
7 end

8 if Edge = shortest predictedServiceTime then
9 | return device  $\leftarrow$  EDGE_DATACENTER;
10 else
11 | if Cloud_via_RSU = shortest predictedServiceTime then
12 | | return device  $\leftarrow$  CLOUD_DATACENTER_VIA_RSU;
13 | else
14 | | return device  $\leftarrow$  CLOUD_DATACENTER_VIA_CBS;
15 | end
16 end

17 if device = CLOUD_DATACENTER_VIA_RSU or
    device = CLOUD_DATACENTER_VIA_CBS then
18 | var CloudHosts  $\leftarrow$  get the number of cloud hosts available;
19 | var hostIndex  $\leftarrow$  get an index based on CloudHosts and user location;
20 | var vmIndex  $\leftarrow$  get an index based on hostIndex;
21 | return selectedVM with hostIndex and vmIndex;
22 else
23 | var EdgeHosts  $\leftarrow$  get the number of edge hosts available;
24 | var hostIndex  $\leftarrow$  get an index based on EdgeHosts and user location;
25 | var vmIndex  $\leftarrow$  get an index based on hostIndex;
26 | return selectedVM with hostIndex and vmIndex;
27 end
```

GSMUploadDelay, GSMDownloadDelay, WLANUploadDelay, WLANDownloadDelay, AvgEdgeUtilization and NumOffloadedTask.

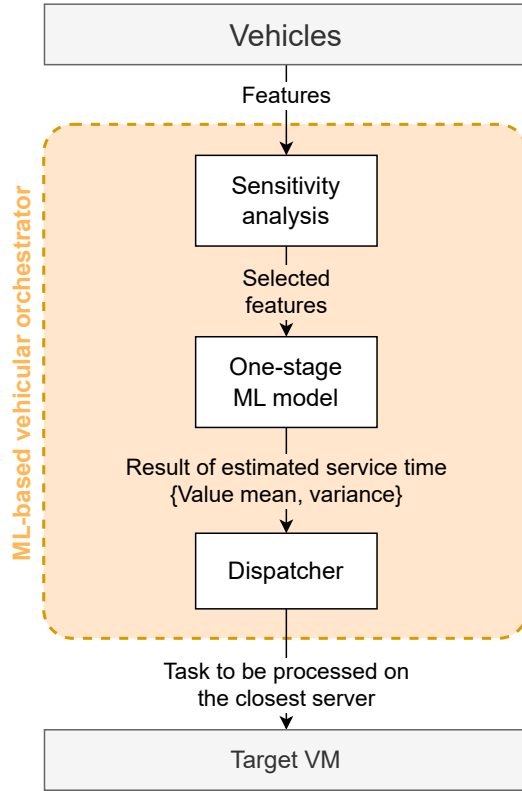


Figure 5.2: Our one-stage proposed architecture

Furthermore, we performed a sensitivity analysis to determine the importance of each of these 12 features. Thus, the most priority feature to estimate the service time of a server is WLANUploadDelay, followed by Decision, TaskLength, NumOf-floodedTask, then SelectedHostID and, finally, GSMUploadDelay. The other features that were not chosen did not present relevance to impact the machine learning model decision, as we can notice in Figure 5.3.

We used the sensitivity analysis explained in Section 2.3.2 and extracted our experiment’s main input features. Table 5.1 brings the specification of these features

with a description for each one.

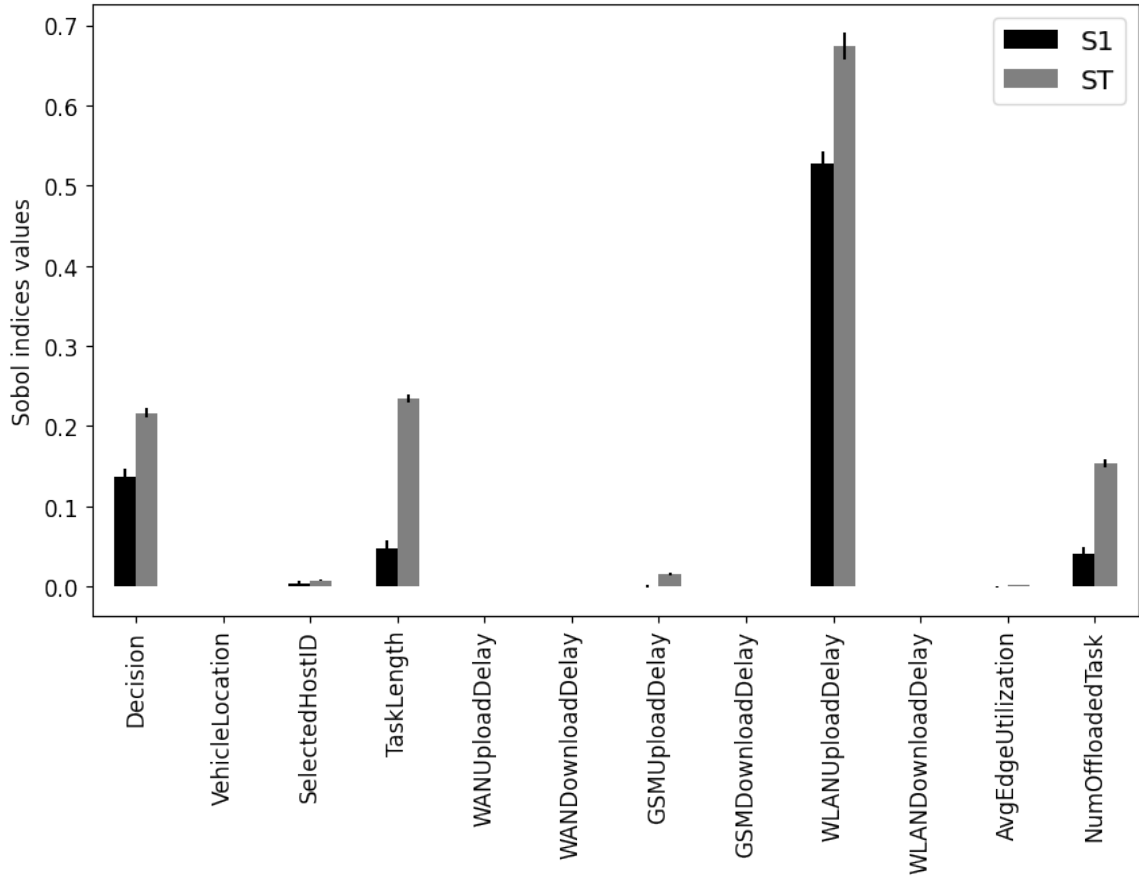


Figure 5.3: (S1 = first-order Sobol index, ST = total Sobol index) - Variance-based sensitivity analysis to define ‘Selected features’.

Table 5.1: Details of features selected as input

Selected feature	Feature Description
Decision	If is to edge (1), cloud via RSU (2) or cloud via CBS (3)
SelectedHostID	Selected server hosting id
TaskLength	Offloaded task size
GSMUploadDelay	Upload delay via Cellular Base Station (CBS)
WLANUploadDelay	Upload delay using wireless local-area network (WLAN)
NumOffloadedTask	Total number of tasks transferred to the selected server in a recent past

Working with sensitivity analysis and feature selection has been demonstrated to be essential for the system’s success. Our goal is to allow our system to work only

with the inputs essential for its decision. In the literature, many authors work with the prior selection of inputs in their systems. For example, the paper [72] points out that including excessive input variables that are not relevant to the target variable can not only complicate the estimation and selection of the model but also impair its performance. Similarly, the work [73] says that incorporating an excessive number of variables in a model can produce significant outcomes but may not hold true in the current population context.

To carry out our experiment, we used as a basis the work developed in [37]. The authors of [37] also propose a centralized, multi-tier architecture for vehicular networks, with workload orchestration based on machine learning. However, the two-stage structure based on ML presented (the red highlight in Figure 5.4) may be simplified and improved to achieve better results. The suggested two-stage architecture, shown in Figure 5.4, is divided to respond to three tiers: i) edge; ii) cloud via RSU; iii) cloud via Cellular Network (CBS); and each of these layers needs two ML models. The first model is to classify whether the unloaded task will succeed or fail, and the other model calculates the service time of the tier predicted to be successful in the offloading process. In total, the two-stage architecture needs to train six different machine learning models to make its predictions. In addition, each of the six models uses various input features, and the reasons for such a choice in the two-stage proposal are unclear [37].

As exemplified in Figure 5.2, we have our one-stage proposed architecture. First, we transmit the vehicle output to the orchestrator, where we initially perform the

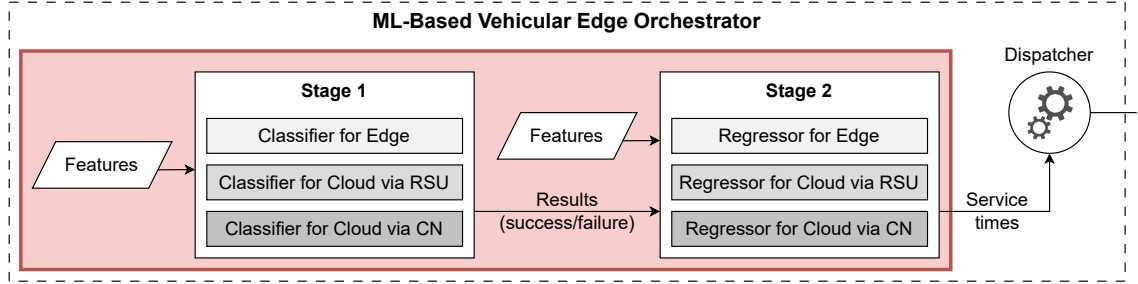


Figure 5.4: Two-stage architecture proposed in [37]

sensitivity analysis. Then the selected features go to our one-stage ML model responsible for estimating the service time in interval ranges for each tier: edge, cloud via RSU or cloud via CBS. We have an output with the interval mean and its variance based on the predicted ranges for each tier. Based on the variance, we can estimate the degree of uncertainty of the model regarding its prediction. Thus, the greater the variance, the more uncertain the model prediction. From this, the choice for the offloading process is made considering the lowest value of the mean and the lowest variance value. Therefore, in addition to being concise and consistent, our proposal considers the forecast’s uncertainty, which is not made in absolute numbers, but in intervals that estimate the minimum and maximum of the service time may take. Consequently, we always consider the service time estimation with the most minor predicted interval for offloading.

In our experiment, the EdgeCloudSim simulator was configured to generate data during a vehicular mobility simulation, in which 100 to 1800 vehicles were randomly distributed at the beginning of the simulation with dynamic speeds, varying in each segment along the road to represent different traffic densities. The simulated road was modelled in a circular path so that the number of automobiles in the simulation

remained constant. For all experiments, we used a notebook with the following specification: AMD Ryzen 7 3800X 3.9 GHZ 8-core processor, 32GB DDR4 3600, SSD NVMe 500GB, NVIDIA® GeForce RTX™ 2060 8GB and Ubuntu-based Linux 20.04 operating system.

The data obtained from the simulation are related to three applications: a navigation app, a danger assessment app, and an infotainment app. Furthermore, the produced data are as follows: Decision (edge, cloud via [RSU](#) or cloud via [CBS](#)), Result (success or failure), ServiceTime, ProcessingTime, VehicleLocation, Selected-HostID, TaskLength, TaskInput, TaskOutput, WANUploadDelay, WANDownloadDelay, GSMUploadDelay, GSMDownloadDelay, WLANUploadDelay, WLANDownloadDelay, AvgEdgeUtilization and NumOffloadedTask. Therefore, considering these data, we used only Result-related entries with success values, which gave us 11,533,902 entries. Furthermore, [Table 5.2](#) presents a summary of the main characteristics defined for each of the applications used.

Table 5.2: Defined characteristics for each of the applications used

	Navigation app	Risk Evaluation app	Infotainment app
Usage percentage ratio	30%	35%	35%
Task interarrival time (sec)	3	5	15
Max delay requirement (sec)	0.5	1	1.5
Delay sensitivity	0.5	0.8	0.25
Upload/Download data (KB)	20/20	40/20	20/80
Task length (GIPS)	3000	10000	20000
RSU /Cloud VM utilization	6%/1.2%	20%/4%	40%/8%

The first part of our proposal is related to the selected features to use as input entries for our [ML](#) model stated in [Figure 5.2](#). For this, we use variance-based sen-

sitivity analysis, with the first- and total-order Sobol indices [46], to determine the inputs that most influence the ML model’s final decision. From that, we select the most relevant input features. As Figure 5.3 shows, the main features chosen for our experiment are Decision, SelectedHostID, TaskLength, GSMUploadDelay, WLANUploadDelay, and NumOffloadedTask.

In a comparative study, we trained six models precisely as specified in paper [37] for the two-stage architecture. First, we prepared three MultiLayer Perceptron (MLP) models for each tier (edge, cloud via RSU or cloud via CBS) to classify success and failure cases. The features used by the authors in [37] for each model were divided as follows: the edge classifier used NumOffloadedTask, WLANUploadDelay, WLANDownloadDelay, TaskLength and AvgEdgeUtilization as input; the cloud classifier via RSU used NumOffloadedTask, WANUploadDelay, and WANDownloadDelay as input; the cloud classifier via CBS used as input NumOffloadedTask, GSMUploadDelay, and GSMDownloadDelay. Then, to predict the service time for task offloading, three linear regression models also used different entries: the edge regressor used TaskLength and AvgEdgeUtilization as input; the cloud regressor via RSU used TaskLength, WANUploadDelay and WANDownloadDelay as input; the cloud regressor via CBS used TaskLength, GSMUploadDelay and GSMDownloadDelay as input. The regression models only estimated the service time of the classifier whose prediction was equal to success, allowing the orchestrator to choose the shortest service time among the estimated ones.

In the one-stage architecture, we studied machine learning algorithms suitable

for solving our regression problem. Therefore, as shown in Table 5.3, we analyzed the following algorithms: linear regression, MLP, M5Rules, Support Vector Machine (SVM) and random forest. Having the cross-validation results of Table 5.3 as a basis, we chose the Random Forest algorithm that presented the best performance and, consequently, is the one that best fitted the problem after our investigations.

Table 5.3: Performance evaluation with 10-fold cross-validation

	Linear Regression	MLP	M5Rules	SVM	Random Forest
Total Number of Instances	1048575	1048575	1048575	1048575	1048575
Correlation coefficient	0.7678	0.9986	0.9985	0.7281	0.9991
Mean absolute error	0.1874	0.0133	0.0139	0.1138	0.0099
Root mean squared error	0.2541	0.0209	0.0218	0.2791	0.0168
Relative absolute error	77.2125%	5.4683%	5.7302%	57.2074%	4.0636%
Root relative squared error	64.0659%	5.2775%	5.4848%	80.1651%	4.2359%

Thus, we isolated all dataset entries whose “Result” column was equal to success and relabeled the “Decision” column as edge = 1, cloud via RSU = 2 and cloud via CBS = 3. In this way, we trained our model to predict the service time on each tier using the following input features: Decision, SelectedHostID, TaskLength, GSMUploadDelay, WLANUploadDelay and NumOffloadedTask in only one-stage architecture. Consequently, we use the same input features and the same model to estimate the service time on edge, cloud via RSU or cloud via CBS.

We present in Figure 5.5 the result of the performance evaluation for the two approaches to predict the task offloading service time. We use the R2 score, MSE and

Mean Absolute Percentage Error (MAPE) as the proposal’s quality evaluation metrics. As shown in Figure 5.5, the result of our approach, the one-stage architecture, significantly outperforms the results of the two-stage architecture. The one-stage architecture’s R2 is incredibly better than the two-stage architecture’s R2. Furthermore, the two-stage architecture’s MAPE has a significantly poor result, and its MSE is also worse than the one-stage architecture’s MSE in all three tiers. The results in Figure 5.5 demonstrate that in addition to our proposal being concise, the results of the one-stage architecture are still much more promising.

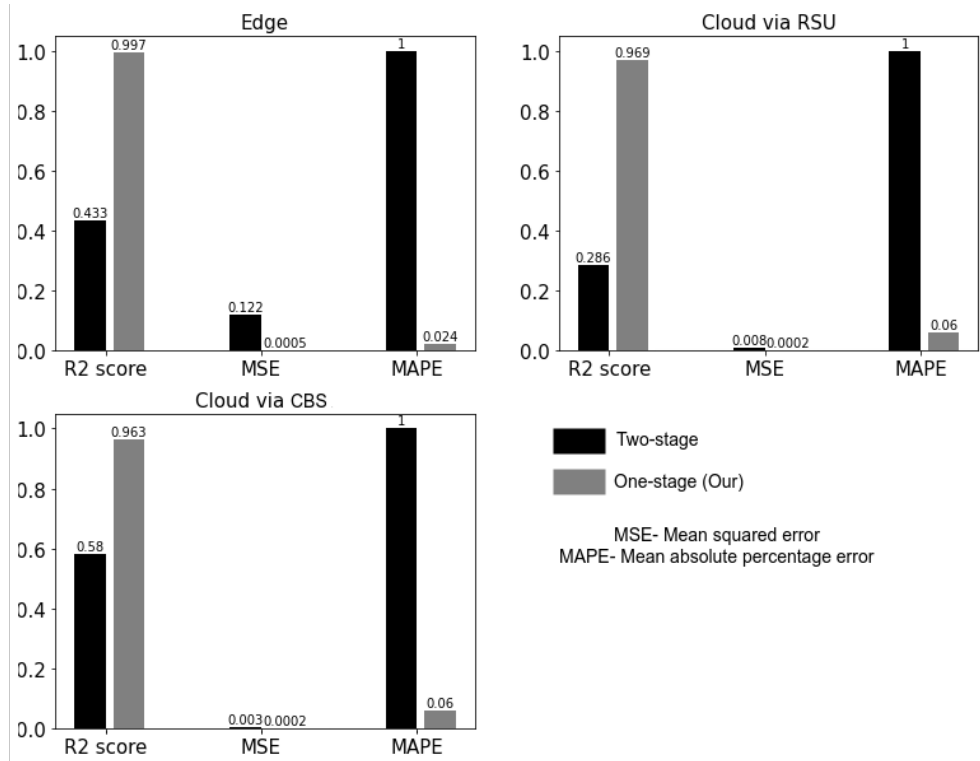


Figure 5.5: Comparison of our proposal with the proposal presented in [37]

After these preliminary results, we used the EdgeCloudSim simulator to perform task-offload simulations on a remote server managed by the ML-based orchestrator. Therefore, we compared the performance of our one-stage orchestrator, a two-stage

orchestrator by [37], and a random orchestrator. The experiments performed on the EdgeCloudSim simulator were based on modelling computational and network resources and the representation of mobile vehicles. We use the random orchestrator to select the offload server randomly, so the probability of selecting a specific target server is equal to the possibility of choosing any of the other available servers. A random technique is used to represent the worst-case scenario.

In the simulated vehicular network, vehicles had some tasks that did not need to be processed locally, so sending them to remote servers on the edge or cloud is possible. Cloud servers can be accessed in two different ways, through [RSU](#) or [CBS](#). Therefore, the orchestrator is responsible for choosing the remote server based on the computational load of the task and the execution time estimated through machine learning. The base configuration of the simulator used in our experiment was the same as described in the paper [37]. Thus, the vehicle applications used to generate the offloaded tasks had different characteristics regarding the arrival time, duration, and size of the upload and download data.

The results of the experiments performed with the EdgeCloudSim simulator are presented below. In these results, we can visually compare the performance of the approaches following different criteria. For example, [Figure 5.6](#) demonstrates the average failure rate of tasks according to the number of vehicles in the simulation. In this way, our one-stage proposal outperforms the other two approaches. Initially, the two [ML](#)-based model starts with the same margin of task failures. Then, as the number of vehicles in the given scenario grows and surpasses the threshold of 1,200, the

one-stage model exhibits a significant improvement in reducing the average number of tasks that fail to meet the specified requirements. This improvement consolidates the one-stage model's position as the most efficient and effective model for the provided scenario.

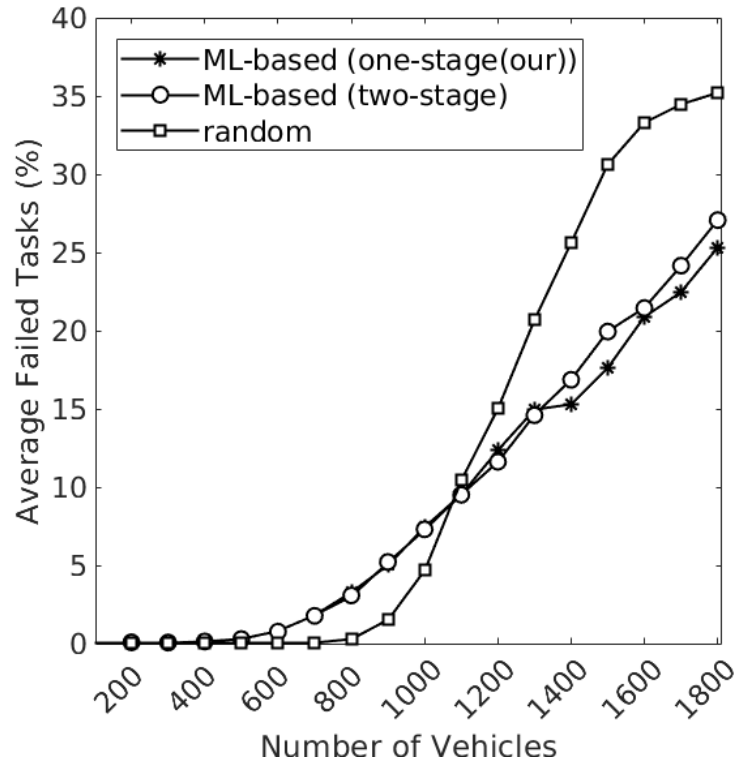


Figure 5.6: Number of tasks that failed during simulation in EdgeCloudSim

Figure 5.7 shows the overhead suffered by the orchestration algorithms as the number of vehicles in the simulation increases. The random model has little overhead because it pushes random choices, making decisions fast, and not generating overhead, but there is no service quality guarantee. The other two models, based on ML, need time to make their decisions, which results in overhead. However, our one-stage proposal presents the best results, considering that it also has the lowest average of failed tasks.

Figure 5.8 illustrates the average network delay at the time of operation of each approach. The random model has many inconsistencies, mainly between 1200 and 1800 vehicles, where the delay increases abruptly and then drops drastically. The unpredictable nature of the random strategy can explain this behaviour of the random model. The delay can vary widely depending on how resources are randomly allocated, especially if the number of resources is limited or the demand is highly variable. However, the two ML-based models, one-stage and two-stage, have the same average network delay during their performance, demonstrating that both have similar quality in this criterion. The inconsistency of the random model reinforces the importance of more sophisticated methods, such as ML-based algorithms, which aim to optimize network delay more predictably and reliably.

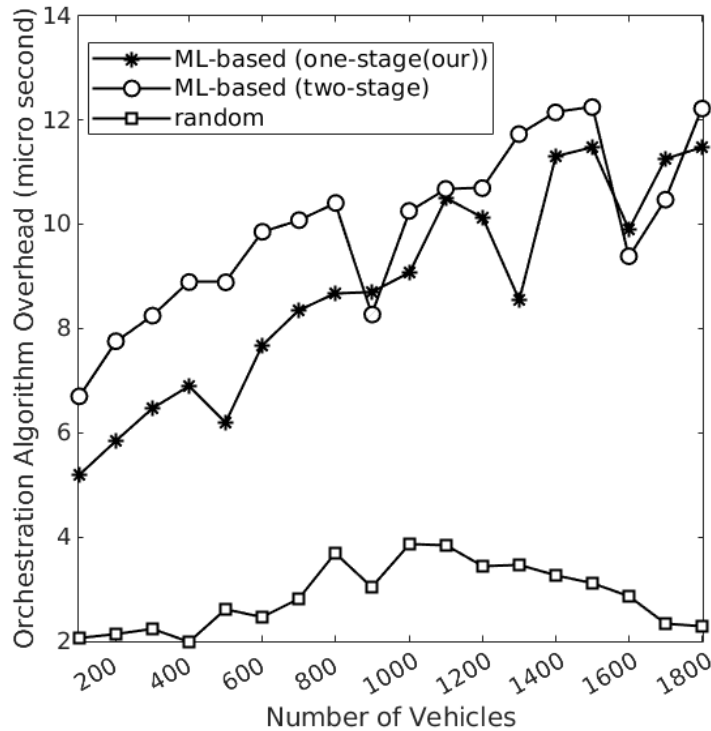


Figure 5.7: Orchestration algorithm overhead during simulation in EdgeCloudSim

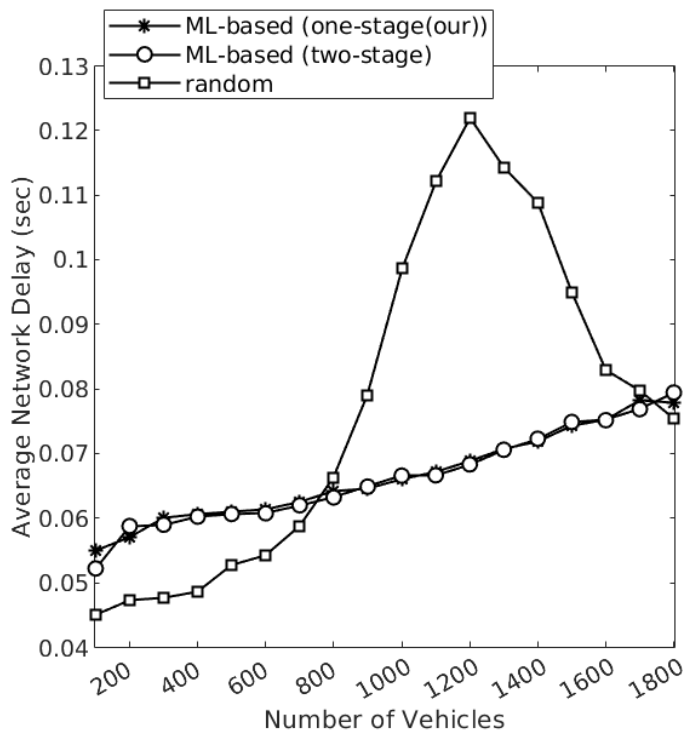


Figure 5.8: Network delay of approaches during simulation in EdgeCloudSim

Figure 5.9 presents the simulation time in minutes for each orchestrator. Based on this, the random algorithm has the shortest simulation time because of the arbitrary decisions made with no elaborated rules. The other two algorithms spend more simulation time. However, the one-stage model still has a significantly better simulation time than the two-stage model.

Overall, the results indicate that our approach best fits what is proposed, having outperformed the experiments. In addition, we can save computational resources when conducting sensitivity analysis, as there is no need to process all the data received. Furthermore, by using the one-stage ML model that outputs the average of the prediction interval and the variance of this interval, we can measure how confident our model is in its prediction. Finally, the Equation formula 5.1 defines the Quality

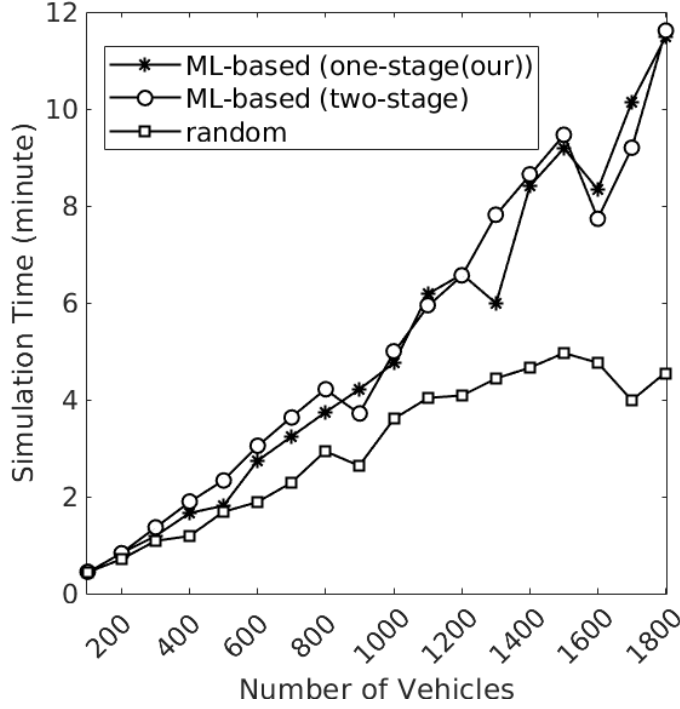


Figure 5.9: Algorithm simulation time duration in EdgeCloudSim

of Experience [Quality of Experience \(QoE\)](#) that evaluates the service time provided by the orchestrator and the number of lost tasks.

$$QoE_i = \begin{cases} 0, & \text{if } T_i \geq 2R_i \\ (1 - \frac{T_i - R_i}{R_i}) \cdot (1 - S_i), & \text{if } R_i < T_i < 2R_i \\ 1, & \text{if } T_i \leq R_i \end{cases} \quad (5.1)$$

Where T_i , as defined in [37], is the current service time, R_i is the delay requirement, and S_i is the delay sensitiveness of a task τ_i . The delay requirements define the maximum available service time for the corresponding task. The average [QoE](#) value decreases as task τ_i is completed after R_i . Delay sensitivity guides delay tolerance.

This value ranges from 0 to 1 and higher for applications with delay intolerance. Figure 5.10 shows the average QoE associated with the number of automobiles. We determine the delay requirements (by task sizes) and task delay sensitivities as in Table 5.2.

As shown in Figure 5.10, the average QoE of models that use machine learning presents the best results, as its objective is to minimize the service time for task processing. On the other hand, the random approach does not seek to reduce the service time and has the worst result.

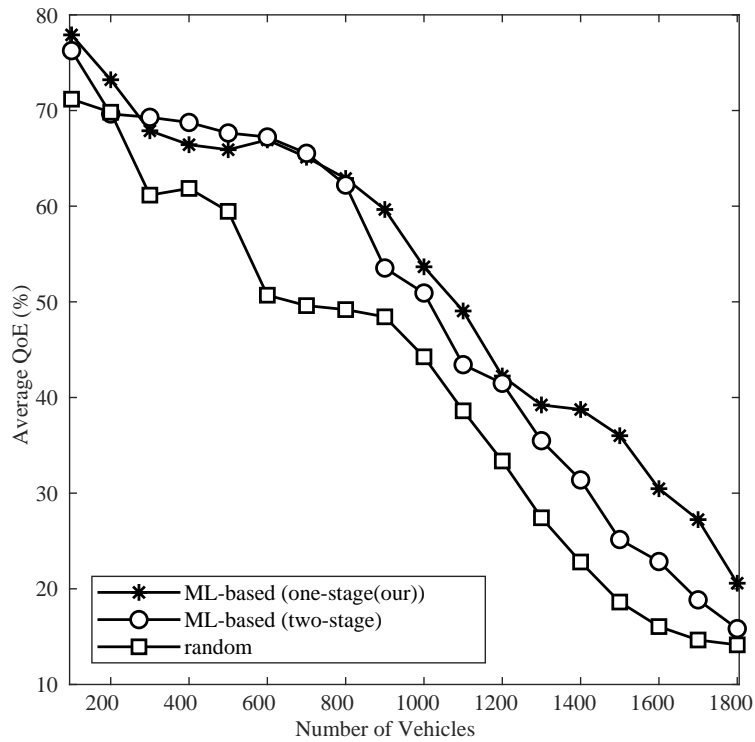


Figure 5.10: Average QoE for the number of vehicles

5.2 Distributed Approach

Real-time intelligent systems are required to process and interpret numerous data simultaneously, demanding high-performance computing and rigorous real-time responses to requested tasks. In the case of autonomous vehicles, we deal with limiting factors such as battery life and constricted computing power that restricts the processing load supported onboard. In this scenario, we must take advantage of external devices that can support the processing performed by the intelligent system in real-time. Furthermore, with over-the-air communication, vehicles can share their information with each other, in addition to using each other's processing capacity to perform possible tasks. The execution of tasks outside the origin node can also be done using fog or cloud resources.

We implemented a machine learning-based task mapper to select the best server for a specific task. Thus, we tested some algorithms (Linear Regression, MLP, k-Nearest Neighbors (KNN) Regressor, SVM and Random Forest), as presented in Table 5.4, and we chose the Random Forest Regressor for presenting the best performance.

Furthermore, the Random Forest Regressor algorithm also proved to be the best option in other work [74] for similar situations. To train and evaluate each of the algorithms in Table 5.4, we collected 204316 data from a simulation with 1000 steps performed in the Simpy-AD simulator [75]. The data used for this training include those presented in Table 5.5 plus a column called 'status' with information about the probability of success for processing the task. This probability of success is the target

Table 5.4: ML algorithms performance evaluation

	Linear Re- gression	MLP	KNN Re- gressor	SVM	Random Forest
R^2 score	0.592	-819835332536004.5	0.595	0.185	0.761
Mean absolute error	0.242	9245813.765243515	0.175	0.203	0.120
Mean squared error	0.101	204958291958594.8	0.101	0.203	0.059
Median absolute error	0.170	2420636.0602586647	0.0	0.0	0.004

column of each evaluated algorithm and varies from 0 (no chance of success) to 1 (100% chance of success). Consequently, the task mapper chooses the server most likely to process the requested task successfully based on the features listed in Table 5.5 and the closest available servers list.

Table 5.5: Features used as inputs by task mapper

criticality	The importance degree of the task can vary between high, medium or low.
deadline	Deadline for task completing.
task_size	The task’s memory footprint.
task_flop	Computation capacity needed by the task.
pu_actual_memory	Currently available server memory.
pu_memory	Total server memory.
pu_task_execution_time	Task execution time on the server.
pu_task_energy_consumption	Task energy consumption on the server.
pu_queue_size	Number of tasks in the server queue.
pu_availability	Amount of server processing available.

In order to validate our approach, we use the Simpy-AD simulator [75], which is a tool for offloading tasks in an autonomous driving context. Furthermore, this simulator allows the use and implementation of edge, fog and cloud communication methods. Thus, it is possible to establish the sharing and collaboration of knowledge

and resources between edge, fog and cloud nodes. The Simpy-AD simulator is a tool developed on top of the SimPy [76] framework. It is a process-based discrete-event simulation representing real-world systems and processes, including urban car platoons and their task processing systems.

In our work, we developed the task mapper based on machine learning as presented in Algorithm 2. In addition, we implemented a task scheduler that considers each task’s deadline and criticality and creates a single circular queue in which tasks that need more time to be processed can go back to the scheduler and be processed again, taking into account their deadline. The pseudocode of our task scheduler is represented in Algorithm 3. Finally, the tasks used in the simulation and their flop, size and criticality are detailed in Table 5.6.

Algorithm 2: ML-based task mapper policy

```

1 Class MLTaskMappingPolicy()
2   Function _init_(env: simpy.Environment):
3     super()._init_(env)
4     ML_model ← loaded_ML_model()

5   Function assignToPu(task, AvailableServers):
6     inputs = [ ]
7     for each server in AvailableServers do
8       features = convertFeatureToVector(task, server)
9       Append features to inputs
10      bestServerProb = ML_model.predict(inputs)
11      index = max(bestServerProb)
12      bestServer = AvailableServers[index]
13      Call bestServer.submitTask(task)

```

Managing tasks in an RSU can be made more efficient and straightforward using a single queue. This approach reduces the complexity of organizing and prioritizing

tasks and allows for more effective monitoring with just one data structure. However, having multiple queues increases the system complexity, requiring the task scheduler to manage and synchronize several queues, which can lead to fairness and starvation problems. Also, resource fragmentation may become an issue, as dividing the resources among queues may not efficiently utilize the [RSU](#)'s total capacity. Additionally, synchronizing queues to prioritize critical tasks introduces delays and complexity. Therefore, selecting a single queue maximizes efficiency and simplifies task management while considering deadline and criticality constraints.

The [Algorithm 2](#) has an `assignToPu` method that first calls the `convertFeatureToVector` method for each server in the server list (`AvailableServers`) to convert the task and server to a feature vector. It then passes all these feature vectors to a pre-trained machine learning model stored in `ML_model` to get predicted success probabilities for each server. It then selects the server with the highest predicted success probability. Finally, it submits the task to the selected server.

The `convertFeatureToVector` method converts a task to a list of features for each server analyzed, including the task's criticality, deadline, size, and FLOPs and the server's available memory, execution time, energy consumption, queue size, and availability as stated in [Table 5.5](#).

The [Algorithm 3](#) has an `addTaskInQueue` method that adds a task to the scheduling queue and increments the scheduler round of the task.

The `getNextTask` method returns the next task from the scheduling queue based

Algorithm 3: Earliest Deadline First and High Criticality (EDFHC) scheduler policy

```
1 Class EDFHCSchedulingPolicy()  
2   Function addTaskInQueue(task_list, task):  
3     Append task to task_list  
4   Function getNextTask(task_list):  
5     while task_list is not empty do  
6       Sort task list by deadline and criticality  
7       selected_task ← first task in task_list  
8       Remove selected_task from task_list  
9       return selected_task  
10  Function getQueueSize(task_list):  
11    return length of task_list  
12  Function getQuantum(quantum):  
13    return quantum
```

on the earliest deadline and highest criticality. It first checks if the queue is not empty, then sorts the task list based on the earliest deadline and highest criticality using the sort method with a key function. Finally, it pops and returns the highest priority task, which is the first task in the sorted list.

The `getQueueSize` method returns the number of tasks in the scheduling queue. Furthermore, the `getQuantum` method returns the scheduling quantum (the time slice (or time quantum) for each task) of the policy, which will help tasks return to the scheduler if necessary.

To simulate a city scenario and establish random departure and arrival points for vehicles and platoons, we use a routing service API, `OpenRouteService` [77]. With this, we defined a city in North America with the respective latitude and longitude in the simulation configuration file. In this way, when starting the simulation, each

Table 5.6: Tasks defined for the simulation

Task name	Task FLOP	Task size	Task criticality
ObjectDetectionTask	230000000	2100	High
ObjectTrackingTask	230000000	2100	Medium
MappingTask	230000000	2100	Medium
GameTask	230000000	2100	Low
MotionPredictionTask	230000000	2100	Medium
TrajectoryPlanningTask	2300000000	2100	High
CollisionPreventionTask	5000000000	2200	High
InformationServiceTask	2000000000	2100	Medium
MusicTasks	1000000000	200	Low
TrafficLightDetectionTask	40000000	100	High

vehicle and platoon has a randomly defined starting and arrival point.

We used the Simpy-AD simulator with 1000 simulation steps to validate our approach. In addition, we defined a total of 100 vehicles per simulation, 5 [RSUs](#), and 5 data centers. Each platoon could have a maximum size of up to 5 vehicles. Each car has an embedded NVIDIA Jetson AGX Xavier as an onboard processing unit. The cars compose the edge nodes with sharing resources and information, which allows them to request the available processing capacity from another vehicle member of the platoon to offload their tasks. Each [RSU](#) makes up the fog level and has the Tesla V100 [Graphics Processing Unit \(GPU\)](#) as the processing unit. Above that, the cloud nodes with their data centers are composed of the DGX A100 server. Edge, fog, and cloud nodes collaborated and shared resources and information.

For our collaborative task management system, we propose using a task mapper based on machine learning as presented by Algorithm 2. Furthermore, we propose a task scheduler, Algorithm 3, which prioritizes tasks with a shorter deadline and

high criticality. Our task scheduler also creates a circular queue for tasks requiring extra processing time, considering their deadline limits. Thus, we call our approach EDFHC_ML, where the name's first part [Earliest Deadline First and High Criticality \(EDFHC\)](#) refers to the scheduler policy, and the second [ML](#) refers to the mapper policy.

To confront the performance of our approach, we compared it with other models. Thus, approach 2 is EDFHC_random, which consists of our proposed scheduler policy ([EDFHC](#)) with a random mapper policy. In the random task mapper, the choice of a server to offload the task is made randomly without considering any choice constraint. Approach 3 is RoundRobin_random, consisting of the Round Robin task scheduler policy, commonly used in computer operating systems and other real-time systems. The Round Robin scheduler assigns a time slice (also known as a time quantum) to each task in its queue without considering the specificities of each task. Finally, approach 4 is RoundRobin_ML, which uses the Round Robin scheduling policy and the mapper policy proposed in this dissertation ([Algorithm 2](#)).

All approaches were performed under the same conditions. [Table 5.7](#) presents an overall result of the performance of the approaches considering the execution of tasks on time. Based on this table, our EDFHC_ML approach had the best performance in executing tasks within the deadline, with a success rate of 64.00%, against 59.56% for EDFHC_random, 53.15% for RoundRobin_random and 51.86% from RoundRobin_ML. It is important to note that none of the approaches left tasks incomplete. However, only the RoundRobin_ML approach did not leave any tasks unexecuted, possi-

bly due to the combination of the Round Robin scheduling policy with our [ML](#)-based mapping policy. With that, our [ML](#)-based mapping policy considers the current conditions of each server, such as queue size and available processing power, for receiving specific tasks. Meanwhile, the Round Robin scheduling policy allocates a time slice to each task without considering its specificities, ensuring all tasks are started on time, even if they are completed after their deadline.

Table 5.7: General comparison of the analyzed approaches

	EDFHC_ML (our)	EDFHC_- random	RoundRobin_- random	RoundRobin_- ML
All tasks	192610	268810	279010	168600
Success tasks/ %	123265/ 64.00%	160103/ 59.56%	148290/ 53.15%	87436/ 51.86%
After dead- line	69335	108697	130710	81164
To execute	10	10	10	0
Incomplete	0	0	0	0

With the simulation results obtained, we randomly selected the vehicle with ID 0 from each platoon to take the average and plot the results. Thus, [Figure 5.11](#) presents the percentage of tasks distributed on each server according to the analyzed approach. In this [Figure 5.11](#), we can better view how the task mapper policy works. Considering latency and possible response delays, it would be more coherent for most tasks to be sent to the edge (AGX servers), a smaller number to the fog (Tesla servers), and an even smaller number to the cloud (DGX servers). The mapping policy that showed the most similar decision was the [ML](#)-based mapper in the EDFHC_ML and RoundRobin_ML approaches, having sent a more significant number of tasks to the AGX edge servers, then a smaller number to the Tesla fog servers, and lastly, an even

smaller number of tasks were sent to the DGX cloud server.

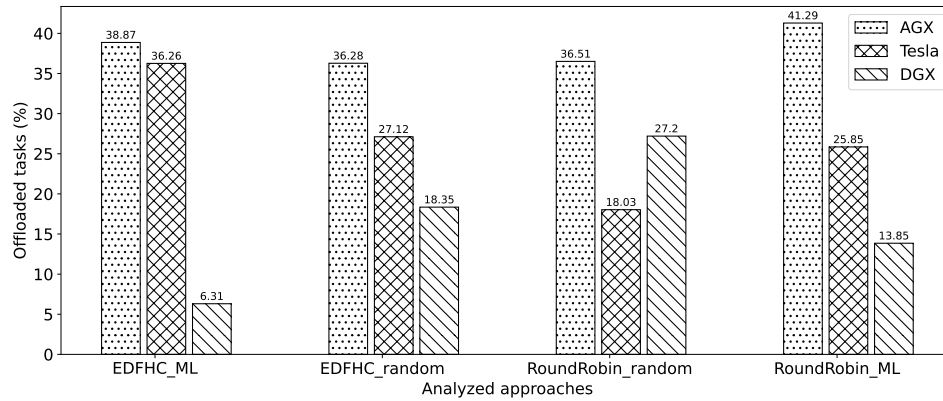


Figure 5.11: Tasks distributed percentage on each server by analyzed approach

Considering only the vehicles with ID 0 from each platoon, Figure 5.12 displays the percentage of tasks successfully completed in Total and on each server. Our approach EDFHC_ML had the best performance in executing tasks successfully in the overall comparison and also on the AGX edge and DGX cloud servers. However, in the Tesla fog server, the EDFHC_random approach had a slight advantage over our EDFHC_ML approach. While our approach executed three tasks on the Tesla fog server after the deadline, the EDFHC_random approach executed only one task after the deadline. However, comparing the type of task that missed the deadline in both approaches, we have that in EDFHC_ML, the three tasks were GameTasks, with low criticality. In contrast, EDFHC_random had one MappingTask with a medium criticality that missed the deadline, as indicated in Table 5.6. Thus, we can say that our approach has a better performance when compared to the successfully executed tasks on the edge, fog, and cloud servers.

Regarding task criticality, Figures 5.13, 5.14, and 5.15 show the percentage of

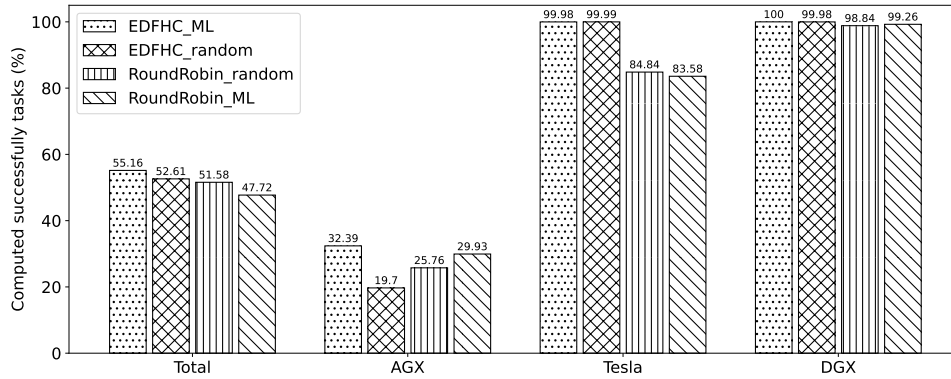


Figure 5.12: Percentage of successfully completed tasks in total and by server

successfully completed tasks according to their criticality in AGX edge servers, Tesla fog servers, and DGX cloud servers, respectively. In Figure 5.13, we can see that the EDFHC_ML and EDFHC_random approach focuses on executing high-criticality tasks first and then proceeding to medium- and low-criticality tasks. On the other hand, the RoundRobin_random approach executes medium-criticality tasks first, then low-criticality tasks, and finally, high-criticality tasks. Finally, the RoundRobin_ML approach executes medium-criticality tasks first, followed by high-criticality tasks and then low-criticality tasks.

Figure 5.14 reveals the percentage of successfully completed tasks according to their criticality on the Tesla fog server. Therefore, we can see that the RoundRobin_random and RoundRobin_ML approaches prioritize the processing of medium-criticality tasks first, followed by low-criticality tasks, and then high-priority tasks. The EDFHC_random approach prioritizes executing high- and low-criticality tasks first and then processes medium-criticality tasks. The EDFHC_ML approach gave priority to the execution of high- and medium-criticality tasks and then processed low-priority tasks.

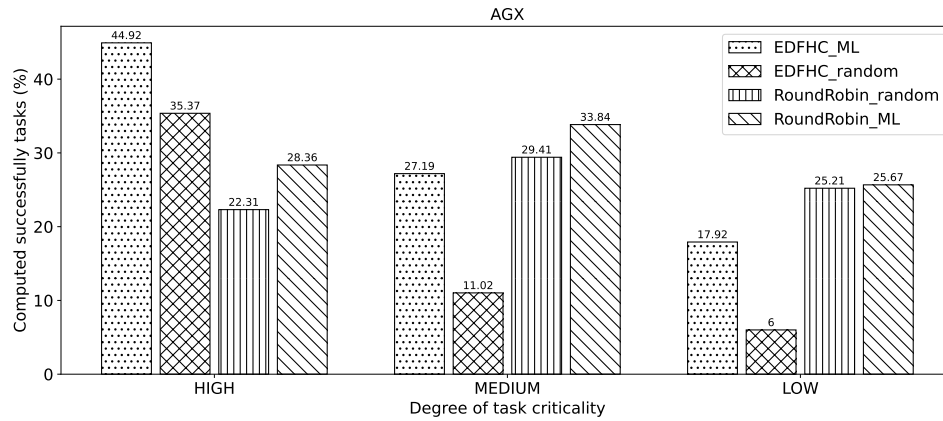


Figure 5.13: Percentage of successfully completed tasks according to their criticality on the AGX Edge Server

In Figure 5.15, we see that the RoundRobin_random and RoundRobin_ML techniques first prioritized medium- and low-criticality tasks before executing high-priority tasks on the DGX cloud server. Meanwhile, the EDFHC_random approach considered prioritizing low-criticality tasks first and then executing high- and medium-criticality tasks. Our approach, EDFHC_ML, executed all high, medium, and low criticality tasks equally on the DGX cloud server.

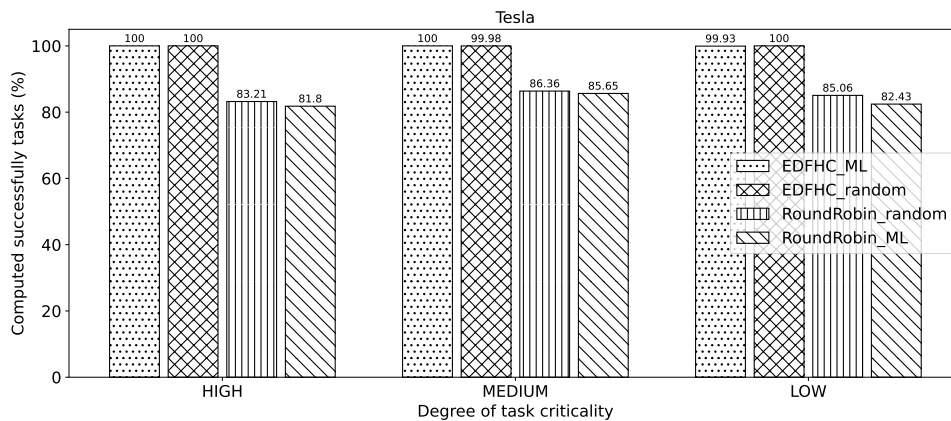


Figure 5.14: Percentage of successfully completed tasks according to their criticality in Tesla Fog Server

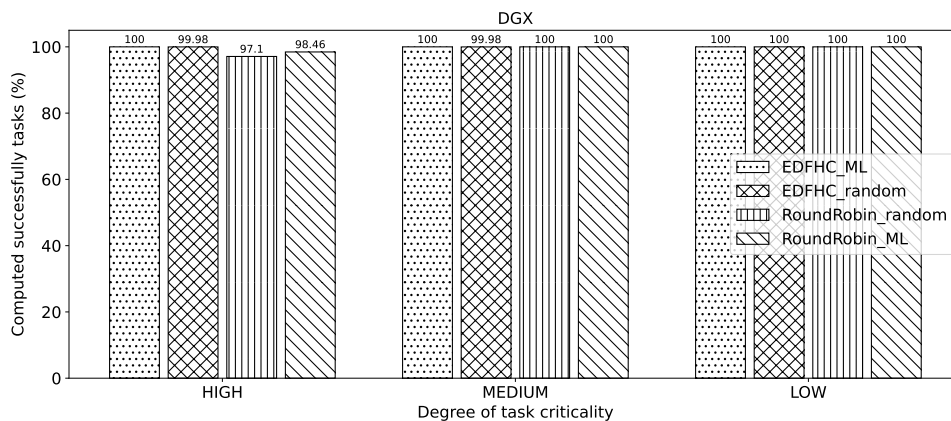


Figure 5.15: Percentage of successfully completed tasks according to their criticality on the DGX Cloud Server

In a real-time system with tasks specified according to their criticality, high-criticality tasks are always expected to be prioritized. If any task has to miss its deadline, it is expected to be a low-criticality task, with medium-criticality tasks being sacrificed only in favour of high-criticality tasks. With this in mind, we have noticed that our EDFHC_ML approach outperformed the other approaches, as it always considers executing high-criticality tasks first, followed by medium-criticality tasks, and only then executing low-criticality tasks.

Analyzing the task definition Table (5.6), we can see that the CollisionPrevention-Task is highly critical and requires more significant processing resources. Herefore, it is common for this task to miss the deadline the most in all approaches. However, the approach proposed by us in this dissertation (EDFHC_ML) works in a circular queue-like manner, where after the allotted processing time for the task has expired and it has not yet been completely processed, it is reallocated in the server execution queue considering its deadline and criticality concerning the remaining tasks in the

queue. As a result, we noticed in Table 5.8 that the CollisionPreventionTask was successfully executed more times by our approach.

Table 5.8: Analysis of the processing success percentage of each approach to CollisionPreventionTask

Approach	Success for the CollisionPreventionTask
EDFHC_ML	6.02%
EDFHC_random	1.05%
RoundRobin_random	2.28%
RoundRobin_ML	0.83%

The demand for high-performance computing and real-time response in processing and interpreting massive data has increased with the rise of intelligent systems, particularly in the case of autonomous vehicles. However, battery life and constricted computing power can limit the processing load supported onboard. Thus, offloading tasks to external devices like fog or cloud resources and using over-the-air communication can be an effective solution. We used the Simpy-AD simulator to validate our approach, and the results showed promising performance. Furthermore, with the help of routing service APIs, random departure and arrival points for vehicles and platoons can be established in a city scenario simulation. Overall, our approach can be beneficial for developing intelligent systems that require high-performance computing and real-time response.

Chapter 6. Learning and continuous monitoring for autonomous real-time vehicle systems

According to the [78] and [47] papers, there are different methods for quantifying uncertainty. Using statistical metrics [79], the typical way is to calculate the mean or expected value and also the variance of the output. The following expression gives the mean of the output:

$$E(H) = \int_a^b hf(h) dh \quad (6.1)$$

Where $E(H)$ is the expected value of the output H , defined by the integral with range $[a, b]$, which is the output space of H . Furthermore, $hf(h) dh$ represents the probability that H is in a range width dh around h .

The variance is defined as the expected value of the squared deviation from the mean of H :

$$Var(H) = E((H - E(H))^2) \quad (6.2)$$

To generate a prediction interval of our prediction (I_H), we are going to use the percentile method (P) to specify the thresholds:

$$I_H = [P_{(a/2)}, P_{(1-a/2)}] \quad (6.3)$$

Therefore, we have a prediction range of 90% of the occurrence of 90% of all H

results. So 5% of those results are below this range, and 5% are above it. $(a/2)$ is the lower critical value and $(1 - a/2)$ is the upper critical value for the standard normal distribution, and $100a\%$ is the confidence level.

Conformal predictors [80] is a widely used approach when we are working with uncertainties [81], [82], being able to make predictions around confidence thresholds. Given a significance level α , a model is capable of making predictions within a $1 - \alpha$ confidence limit. In other words, the probability of the correct label being within the predicted confidence set is almost exactly $1 - \alpha$.

Predictions will use ranges around the actual value in regression problems rather than predicting a single absolute value. For classification problems, the prediction will consider the possibility that each class in the data set is the proper label. Figure 6.1 shows the general operation of conformal predictor work.

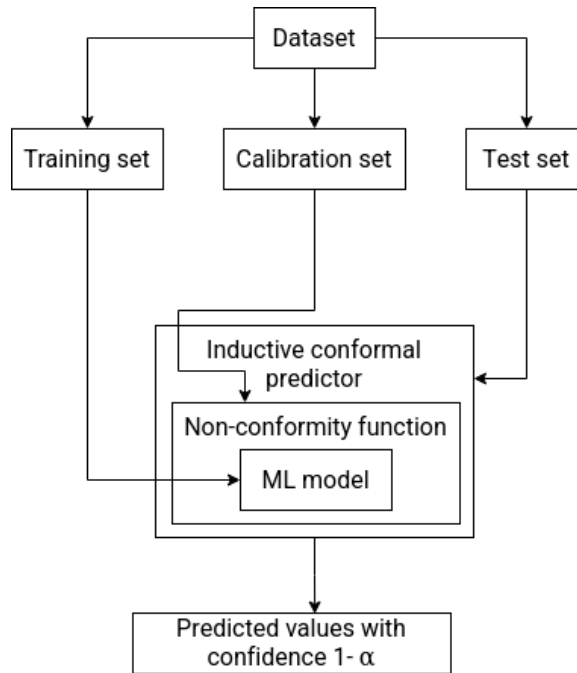


Figure 6.1: Conformal predictor workflow

We use the conformal predictor shown in Figure 6.1 as part of our approach proposed. Therefore, according to Figure 6.1, we first divided our dataset into training, calibration and test sets. So we fit an ML model with the proper training set. Then, we define a nonconformity function that makes a sample distribution based on scores determined by the similarity of the elements in its set with those of the training set. The nonconformity function then receives the calibration set and generates the calibration scores. These scores feed the inductive conformal predictor, which is the top layer of the model. Finally, that inductive conformal predictor layer is responsible for generating predictions distributed according to the calibration scores, using as a cut-off point the significance level that varies from 0 to 1. Therefore, for example, we can say that we have an ordered set of 500 conformal scores. Then, if $\alpha = 0.1$, the cut-off point for statistical significance will be the conformal score at the 90th percentile, in this case, the 450th conformal score.

We will use in our work an [Inductive Conformal Predictor \(ICP\)](#) [83] instead of the [Transductive Conformal Predictor \(TCP\)](#) [84]. We decided to focus on ICP because it is the most widely used approach to conformal predictions. Furthermore, it requires that the ML model be trained one single time for all predictions until a new amount of data is collected, and it is worth retraining to update the model [85].

We use two scenarios: a congestion identification scenario and a fire identification scenario. We will use simulated data for the first scenario and real sensor data for the second one.

6.1 Scenario 1

Congestion strongly influences mobility, accessibility and the emission of pollutants but varies widely over time and across space [86]. Therefore, we consider that the congestion detected event occurs when the traffic flow reaches its maximum value, and the traffic density continues to increase.

For this scenario, we used a traffic dataset generated through the [Simulation of Urban MObility \(SUMO\)](#) framework [87]. Thus, we used the net file with the Downtown Toronto abstraction made in [88] to identify congestion points on city roads, as revealed in Figure 6.2. The simulation map has 52 intersections and roads with 1 or 2 lanes. Thus, we placed [SUMO](#) lane area detectors (E2) scattered along the roads to measure the traffic generated by 1000 vehicles inserted in randomized traffic for a defined time from 0 to 100000.

We split our dataset between the training and test sets, with the test set having 25% of the dataset’s data and the training set having 75% for experiments that did not consider uncertainty. Then we kept the same 25% for the test dataset and split the rest evenly between the training and calibration sets in the experiments that considered uncertainty. The total number of entries in the dataset was 532,014, with the test sample size always 177,338.

To quantify the uncertainty of our model, we use the percentile to determine what range we would like our answer to fall in. Based on equation 6.3, we set this range

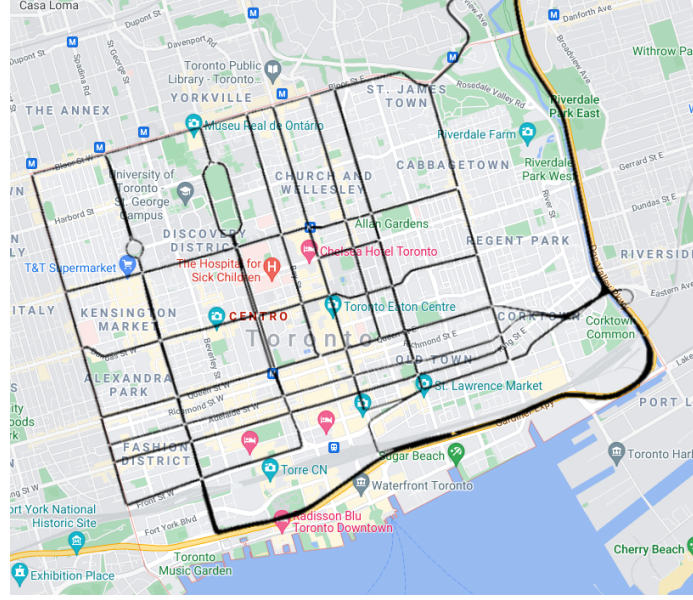


Figure 6.2: Comparison of the traffic network used in the simulation with the real map of Toronto

to 97%. For this, we set the conformal prediction significance to $\alpha = 0.03$. Thus, every time our model predicted labels 0 or 1, there was a 97% probability that this prediction was correct.

We performed three different tests to validate our approach in scenario 1. The first experiment was a binary classification problem, the second was a multi-level classification problem, and the third was a regression problem. For the three problems, we had as input the data related to the average speed (m/s), the flow (counts), the average occupancy (%) and the mean halting duration (s). Below we detail all the experiments.

Consider the four primitive events consumed for complex event composition:

- **AverageSpeed**(timestamp=1647037870, m/s=13.5)
- **Flow** (timestamp=1647037870, counts=1.38)

- **AverageOccupancy**(timestamp=16470, percent=3.34)
- **HaltingDuration**(timestamp=1647037870, sec=1)

Our model presents the same structure for dealing with classification or regression problems, using a combination of Random Forest Regressor [89], k-Nearest Neighbor (kNN) [90] and conformal prediction as illustrated by Algorithm 4. First, as shown in Figure 6.3, the model receives the inputs and verifies the essential ones through sensitivity analysis. Afterward, the selected features enter the model, composed of conformal prediction and an ML model as stated in Figure 5.4. So the prediction for the classification or regression problems can be performed. Then, our model predicts intervals from which we take the mean that indicates the prediction result.

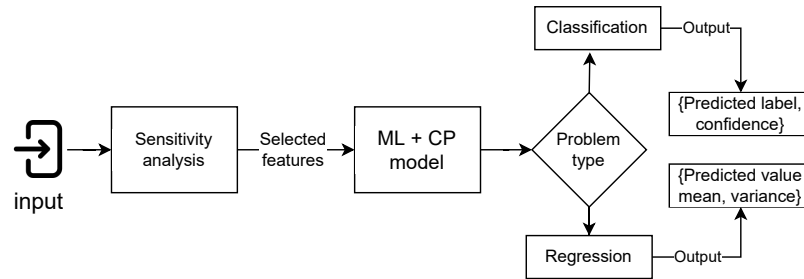


Figure 6.3: Proposed approach to estimate uncertainty in events

For comparison, we implemented two other replicable approaches per the paper’s information and definitions [91]. Thus, Algorithms 5 and 6 represent the **Normal Probabilistic Model (NPM)** and **Improved Probabilistic Model (IPM)** approaches for classification (Algorithm 5) and regression (Algorithm 6) problems. The difference between **NPM** and **IPM** declared by the [91] authors is the probability distribution function of the error, which in **NPM** is $N(0, 1)$ and in **IPM** is the normal distribution of the standard deviation representing the inaccuracy of sensors specified by the manufacturer.

Algorithm 4: Proposed Machine Learning with Conformal Predictor (ML-CP) model

```
1 Class
  ML_CP_model(x_train, y_train, x_calibrate, y_calibrate, x_test, problem_type)
2   underlying_model  $\leftarrow$  RegressorAdapter(RandomForestRegressor())
3   normalizing_model  $\leftarrow$  RegressorAdapter(KNeighborsRegressor(n_neighbors =
4     5))
5   normalizer  $\leftarrow$ 
6     RegressorNormalizer(underlying_model, normalizing_model, AbsErrorErrFunc())
7   nc  $\leftarrow$  RegressorNc(underlying_model, AbsErrorErrFunc(), normalizer)
8   icp  $\leftarrow$  IcpRegressor(nc)
9   icp.fit(x_train, y_train)
10  icp.calibrate(x_calibrate, y_calibrate)
11  prediction  $\leftarrow$  icp.predict(x_test, significance = 0.03)
12  prediction_mean  $\leftarrow$  np.mean([prediction[:, 0], prediction[:, 1]], axis = 0)
13  results  $\leftarrow$  []
14  foreach value in prediction_mean do
15    frac  $\leftarrow$  value mod 1
16    confidence  $\leftarrow$  100 if frac < 0.1 else (1 - frac) * 100 if frac  $\leq$  0.5 else
17      frac * 100
18    if problem_type = "binary classification" then
19      | results.append((1 if value > 0.5 else 0, confidence))
20    else
21      if problem_type = "multi-classification" then
22        | class_prediction  $\leftarrow$  len(thresholds)
23        | foreach i, threshold in enumerate(thresholds) do
24          | if value  $\leq$  threshold then
25            | | class_prediction  $\leftarrow$  i
26          | else
27            | | class_prediction  $\leftarrow$  i + 1
28          | end
29        | end
30        | results.append((class_prediction, confidence))
31      else
32        | var  $\leftarrow$  |prediction[:, 1] - prediction[:, 0]|
33        | results.append((value, var))
34      end
35    end
36  end
37  return results
```

In scenario 1, we are working with simulated data, and therefore, we need information regarding the factory inaccuracy of the sensors. Considering this, we assume that sensor uncertainty is proportional to the standard deviation of the simulated data ($stds*0.1$). In this way, we have the probability distribution function of the error equal to $[\mathcal{N}(0, 1), \mathcal{N}(0, 1), \mathcal{N}(0, 1), \mathcal{N}(0, 1)]$ for **NPM** and $[\mathcal{N}(0, 0.7), \mathcal{N}(0, 0.4), \mathcal{N}(0, 1.0), \mathcal{N}(0, 3.7)]$ for **IPM** relative to the four input primitive events.

Algorithm 5: Probabilistic Model Classification

```

1 Class ProbabilisticModelClassification
2   Function __init__(self, detector_stds){
3     |   self.means ← None
4     |   self.stds ← detector_stds
5   }
6   Function Fit(self, X, y){
7     |   n_features ← length(X[1])
8     |   n_classes ← length(unique(y))
9     |   self.means ← zero_matrix(n_classes, n_features)
10    |   foreach label in range(n_classes) do
11    |     |   X_label ← X[y == label]
12    |     |   self.means[label] ← mean(X_label)
13    |   end
14  }
15  Function Predict(self, X){
16    |   n_classes ← length(self.means)
17    |   log_likelihoods ← zero_matrix(length(X), n_classes)
18    |   foreach label in range(n_classes) do
19    |     |   log_likelihoods[:, label] ←
20    |     |     sum(norm_logpdf(X, self.means[label], self.stds))
21    |   end
22    |   return argmax(log_likelihoods)
23  }

```

Analysis of the proposed method for a binary classification problem:

The first experiment is a simple binary classification problem. The objective is to determine whether there is congestion (no-congestion=0, congestion=1).

Algorithm 6: Probabilistic Model Regression

```
1 Class ProbabilisticModelRegression
2   Function __init__(self, feature_stds){
3     self.means ← None
4     self.feature_stds ← feature_stds
5   }
6   Function Fit(self, X, y){
7     n_features ← length(X[1])
8     self.means ← zero_array(n_features)
9     foreach i in range(n_features) do
10    | self.means[i] ← mean(y * X[:, i])
11    end
12  }
13  Function Predict(self, X){
14    y_pred ← zero_array(length(X))
15    foreach i in range(length(X)) do
16    | weights ← 1/(self.feature_stds2)
17    | y_pred[i] ← sum(X[i, :] * self.means * weights)/sum(X[i, :] * weights)
18    end
19    return y_pred
20  }
```

With our proposed approach, the sensitivity analysis indicates “speed”, “occupancy” and “mean_Halting_Duration (s)” as the essential inputs (Figure 6.4). S1 means first-order Sobol indices, and ST indicates total Sobol indices. First-order Sobol indices consider each parameter’s direct contribution, excluding interaction terms, to a specific output. The total Sobol indices provide a view of all interactions of a given input, not providing details of which parameter this input interacts with nor in which order.

Following the Algorithm 4, our approach initially predicts a range between 0 and 1, for which we calculate a mean. Then, the estimated mean value is used as a confidence index to indicate the choice of a label. For example, if we have a mean between 0 and 0.50, the output label will be 0, and the confidence index will range

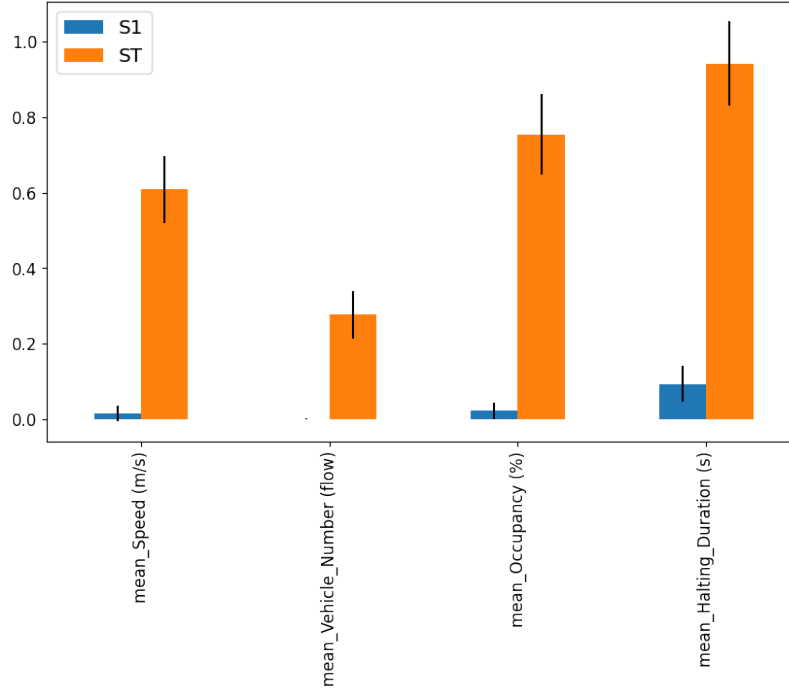


Figure 6.4: Sobol analysis to indicate priority input for binary classification problem from 50% to 100%, depending on the mean. On the other hand, if the mean is from 0.51 to 1, the chosen label will be 1, with a confidence index ranging from 51% to 100%. Consequently, if we have $mean = 0.30$, then $label = 0$ and $confidence = 70\%$.

We calculated the sensitivity and specificity of each model as defined by Equations 6.4 and 6.5, respectively.

$$Sensitivity = \frac{TP}{TP + FN} \quad (6.4)$$

$$Specificity = \frac{TN}{TN + FP} \quad (6.5)$$

TP - the number of true positives.

FN - the number of false negatives.

TN - the number of true negatives.

FP - the number of false positives.

After the previous clarifications, we present the obtained results in Table 6.1.

Table 6.1: Models performance for the binary classification problem

	ML_CP	NPM	IPM
Accuracy	0.98	0.64	0.63
Sensitivity	0.97	0.31	0.29
Specificity	0.98	0.99	0.99

We analyzed the accuracy, sensitivity and specificity of each model. The sensitivity indicates the proportion of label 1 classes that were actually classified as label 1 by the model (true positive). On the other hand, specificity indicates the proportion of label 0 that was classified as label 0 (true negative). Thus, considering sensitivity and specificity, in addition to accuracy, the **ML_CP** model presents the best results. Although **NPM** and **IPM** have a marginally higher specificity rate than **ML_CP**, it does not necessarily mean that the **NPM** and **IPM** models were effective. The **NPM** and **IPM** models tended to classify most of their predictions as 0, resulting in high specificity but low sensitivity and accuracy.

Analysis of the proposed method for a multi-level classification problem:

For the experiment involving the multi-classification problem, we replaced the last column of our dataset, which initially only indicated whether there was (label 1) or not (label 0) congestion. In place of this last column, we now have three different situations in case there is congestion: light congestion (label 1), moderate congestion (label 2) and severe congestion (label 3), in addition to label 0 in case there is no congestion. Also, we maintain the same four analysis features (speed, flow, occupancy

and halting_duration). For the [ML-CP](#) model, we only use the features selected by sensitivity analysis presented in [Figure 6.5](#), which means “mean_Vehicle_Number (flow)”, “mean_Occupancy (%)” and “mean_Halting_Duration (s)”.

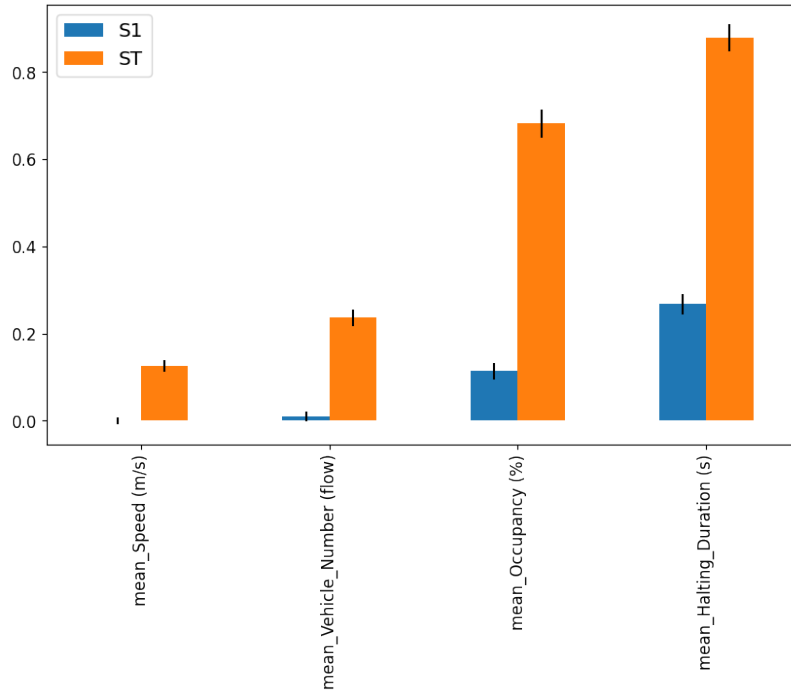


Figure 6.5: Sobol analysis to indicate priority input for multi-level classification problem

The traffic jam level was defined based on the average jam length in meters. Thus, when the average congestion length is less than or equal to 82.5 meters, we have congestion level 1. When the congestion length is more significant than 82.5 meters and less than or equal to 165 meters, we define congestion as level 2. Finally, if the congestion length exceeds 165 meters, we have congestion level 3.

Table [6.2](#) presents the performance of the models for this problem, quantifying the global model sensitivity and specificity, as defined in [Equations 6.4](#) and [6.5](#), respectively. In addition, the table also discriminates the accuracy of each of the models.

Table 6.2: Models performance for the multi-level classification problem

	ML_CP	NPM	IPM
Accuracy	0.98	0.72	0.74
Sensitivity	0.97	0.71	0.74
Specificity	0.97	0.75	0.76

The models can classify their results with labels 0, 1, 2 or 3. As shown in Table 6.2, the results for the **NPM** and **IPM** models are pretty similar. The **ML_CP** model still has more promising results.

Analysis of the proposed method for a regression problem:

We use the same dataset used in the other two experiments for this regression problem, changing only the target column. Thus, we have speed, flow, occupancy and halting duration as default data features. In addition, we defined a mean_max_-jam_length_in_meters column as the target, ranging from zero to 250 meters. For this problem, the model would have to predict the max value of the jam length considering the provided features.

For the **ML_CP** model, we used our proposal discussed above (Algorithm 4). The inputs selected through sensitivity analysis for this problem are “mean_Speed (m/s)” and “mean_Vehicle_Number (flow)”, displayed in Figure 6.6. Thus, we calculated an interval with a 97% chance of containing the value in meters of the congestion size, which can vary from 0 to 250 meters. Then, we calculate the mean and the variance based on the interval values found, which are our model outputs for the specified regression problem.

The performance of regression models was evaluated, and the results are presented

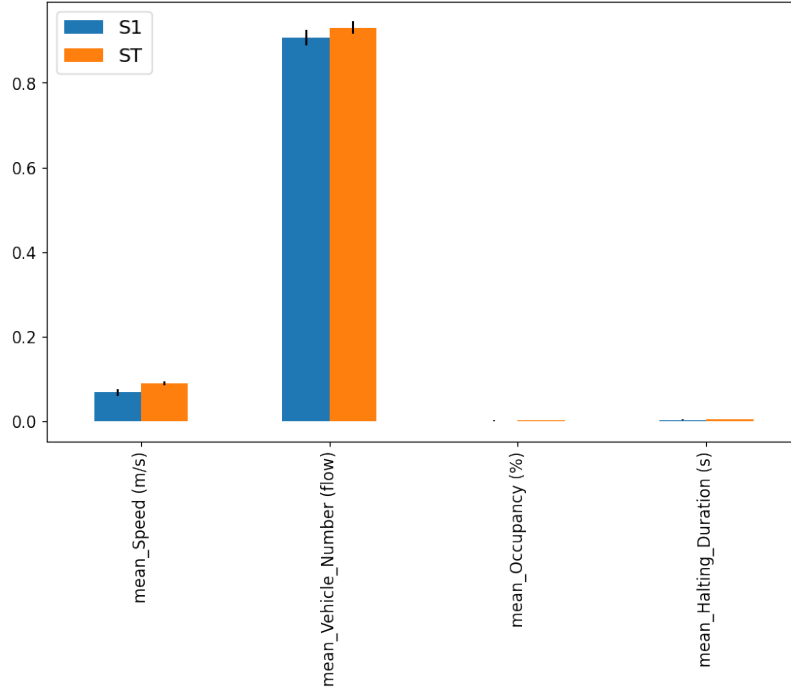


Figure 6.6: Sobol analysis to indicate priority input for regression problem

in Table 6.3. The metrics used for evaluation were R^2 , mean absolute error, mean squared error and median absolute error. The results of Table 6.3 indicate that our [ML_CP](#) proposal outperformed other models in this regression problem, as well as in the classification problems previously analyzed.

Table 6.3: Models performance for the regression problem

	ML_CP	NPM	IPM
R^2	0.99	-132.0	-8.11
Mean absolute error	0.29	234.39	59.4
Mean squared error	2.76	124276.16	8512.38
Median absolute error	0.003	118.93	36.07

6.2 Scenario 2

This scenario involves detecting fires in various environments by analyzing temperature, smoke, and flame characteristics. Scenario 2 was employed to demonstrate that the uncertainty quantification technique proposed for vehicle scenarios can also be extended to other scenarios. It is worth noting that the dataset used in this scenario comprises real-world data collected and provided by the authors of [92]. To build this dataset, the researchers used three sensors: a DHT11 for measuring temperature, an MQ-2 smoke sensor, and an LM393 flame sensor. This dataset was also used to assess the effectiveness of the proposed DST-CEP [91]. Thus, we reproduce the tests to evaluate and compare our approach (ML-CP) with the DST-CEP, Normal Probabilistic Model (NPM) and Improved Probabilistic Model (IPM).

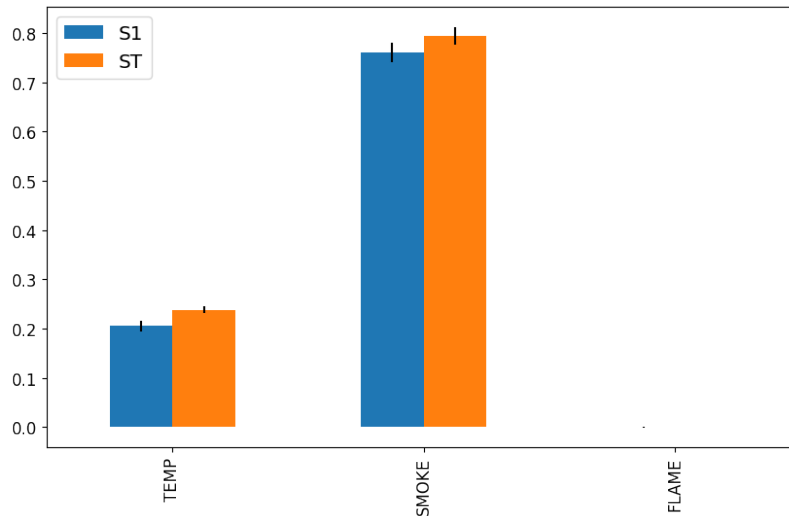


Figure 6.7: Sobol analysis to indicate priority input for fire detection problem

In the sensitivity analysis of our model, shown in Figure 6.7, only the temperature and smoke features were considered to be of high relevance for the detection (label 1)

or non-detection (label 0) of fire. Consequently, as input, we only use the temperature and smoke values.

In Table 6.4, we have presented the results of our performance analysis compared to three other approaches. The table clearly shows that our solution surpasses all others in terms of Accuracy, Precision, Recall, and F-Measure metrics.

Table 6.4: Models performance for fire detection scenario

Approach	Accuracy	Precision	Recall	F-Measure
NPM	81.14%	67.32%	55.68%	60.95%
IPM	81.43%	68.21%	55.68%	61.31%
DST-CEP	95.00%	85.71%	97.30%	91.14%
ML_CP	99.00%	98.00%	99.00%	99.00%

Chapter 7. Experimental Results and Discussion

In this chapter, we will be examining the outcomes of our proposed [CARVS Framework](#). This involves analyzing the results of the final experiments conducted for Context Awareness for Autonomous Systems, Task Management in Autonomous Systems, and Learning and Performance Monitoring of Autonomous Systems. These three cores are depicted in [Fig. 3.1](#) as the Context Awareness Core, Task Management Core, and Learning and Performance Monitoring Core, respectively.

7.1 Context Awareness for Autonomous Systems

We prepare a solution to apply Gaussian or [OU](#) noises in an autonomous vehicle’s training. As the [Algorithm 7](#) shows, in line 9, we introduced our proposal to the [A3C](#) model. Thus, N is the turbulence factor added to the policy, and it can be either Gaussian or *Ornstein-Uhlenbeck*. We use $\sigma = 0.02$ as the noise factor for our tests with both Gaussian and [OU](#). After several tests, we found that the value of $\sigma = 0.02$ is the one that best contributes to the exploration of our agent. We can adjust this value according to the problem objective. In general, the noise level does not need to be high for extensive exploration by the higher-scoring agent. Tests performed by other researchers, such as [\[93\]](#) and by us, point out that large noise induces the agent to a local optimal or sub-optimal policy, and what we are looking for is an optimal global policy.

The autonomous vehicle should act according to the above pseudocode, [Algorithm 7](#). The first step is to initialize the global policy and the value network. Then, it

starts the global step count, and the episode begins. Each agent performs the steps from line 4 to line 20 asynchronously. Thus, each agent resets its gradients to 0, starts the internal time, and collects the first state observation. And then, the agent repeatedly updates the reward amount according to policy-based action, taking into account the global gradient and increments the time. Thus, if the state is the last, the reward receives 0 and, if it is not the terminal one, the reward is updated with the value function with the discount factor. Then, edit the gradients as the global policy and value have been updated as well. Each local policy has entropy.

Algorithm 7: A3C with noise - pseudocode

```

1 //Assume global parameter vectors  $\theta$  and  $\theta_v$ ;
2 //Initialize step counter  $t \leftarrow 1$ ;
3 //Initialize episode counter  $E \leftarrow 1$ ;
4 repeat
5   | Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ ;
6   |  $t_{start} = t$ ;
7   | Get state  $s_t$ ;
8   | repeat
9   |   | Perform  $a_t$  according to policy  $\pi(a_t|s_t; (\theta \leftarrow N))$ ;
10  |   | Receive reward  $r_t$  and new state  $s_{t+1}$ ;
11  |   |  $t \leftarrow t + 1$ ;
12  | until terminal  $s_t$  or  $t - t_{start} == t_{max}$ ;
13  |  $R = \begin{cases} 0 & \text{\#for terminal } s_t \\ V(s_t, \theta_v) & \text{\#for non-terminal } s_t \end{cases}$ 
14  | for  $i \in \{t - 1, \dots, t_{start}\}$  do
15  |   |  $R \leftarrow r_i + \gamma R$ ;
16  |   | Update gradients wrt:
17  |   |  $\theta : d\theta \leftarrow d\theta + \nabla_{\theta} \log \pi(a_i|s_i : \theta)(R - V((s_i : \theta_v)) + 0.01(H(a_i)))$ ;
18  |   | Perform an asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ ;
19  |   |  $E \leftarrow E + 1$ 
20 until  $E > E_{max}$ ;

```

The A3C forward view selects actions using its exploration policy for up to E_{max} .

The agent will then receive the rewards from the environment since its last update.

When updating the policy and the value function using the advantage method, actions that perform better than the value function's expectation will receive more weight and are more likely to be selected in the next steps.

We have the input passing into the neural network with the dimensions 84×84 . As Figure 7.1 shows, we use two convolutional layers with [Rectified Linear Unit \(ReLU\)](#) function each. After that, we have a fully connected dense layer with 256 neurons, and another fully connected layer outputs the Value function and the probabilities for each action. This structure is a standard architecture used by the community when working with the [A3C](#) approach.

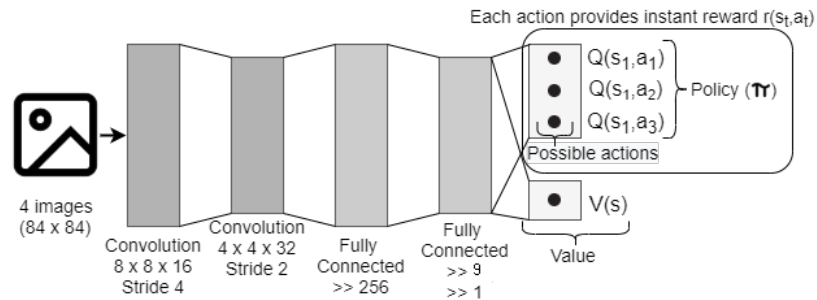


Figure 7.1: Agent [A3C](#) model

Considering our autonomous driving agent, we have a limited number of possible actions to move the agent from one state to another, which characterizes our conditions as a discrete environment. Besides, current actions taken may affect future activities, which describe a sequential environment [94].

[Car Learning to Act \(CARLA\)](#) is a suitable simulator for handling our assumptions and constraints. [CARLA](#) simulator is an open-source autonomous driving research simulator made in partnership with researchers from Intel Labs, Toyota Research

Institute, and Computer Vision Center [95]. Moreover, CARLA has system flexibility and several available resources to run our experiments.

We started one server in the CARLA simulator for performing the tests. Each asynchronous agent resulting from our A3C global agent had its copy of the environment connected to the server as a client, as shown in Figure 7.2. We define the number of asynchronous agents resulting from the global network with the Python code: `multiprocessing.cpu_count()`. According to our system specifications, we have a total of 12 threads resulting from that Python process. It means that we could train 12 asynchronous agents at the same time.

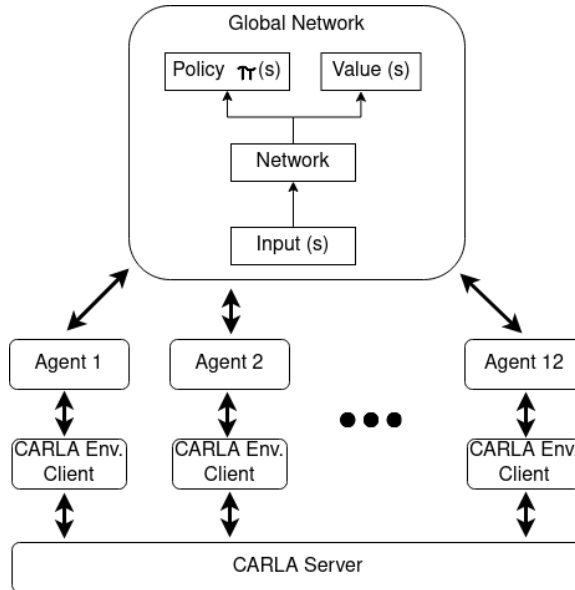


Figure 7.2: Specification of the A3C agent connected to the CARLA simulator

Since we are in a discrete environment, the agent’s actions are also discrete. It can perform nine steps, including turning left, turning right, forward, and brake. Therefore, we use 3-dimensional vectors, with the throttle, steer, and brake values, respectively, to represent each action, as shown below:

$actions = 0 : [1.0, 0.0, 0.0], Forward$
 $= 1 : [0.0, -1.0, 0.0], Turn Left$
 $= 2 : [0.0, 1.0, 0.0], Turn Right$
 $= 3 : [1.0, -1.0, 0.0], Forward Left$
 $= 4 : [1.0, 1.0, 0.0], Forward Right$
 $= 5 : [0.0, 0.0, 1.0], Brake$
 $= 6 : [0.0, -1.0, 1.0], Brake Left$
 $= 7 : [0.0, 1.0, 1.0], Brake Right$
 $= 8 : [0.0, 0.0, 0.0], No Action$

We desire to find out what kind of noise would be most beneficial to the agent's policy, whether a more random or a temporally correlated one. Then, we perform the modification shown in algorithm 8 in order to be able to compare the two types of noise. First, we define the noise we will use, whether Gaussian or *Ornstein-Uhlenbeck* [OU](#), always setting the mean (μ) and the standard deviation (σ). Then, the policy receives that noise and incorporates it into its decision estimates, which will not reach very high certainties and stimulate significant exploration growth.

The twelve asynchronous copies of the [A3C](#) agent are initially placed at random positions. They must move to a second position other than the initial one without colliding with any object, as described in the list of Tasks detailed below.

- Task 1 - The vehicle is placed randomly in an initial position and aims to reach a specific point of 5m in front of it. In this first Task, the vehicle needs to move straight ahead.
- Task 2 - The vehicle is placed randomly in an initial position and aims to reach a specific point of 30m distance. The vehicle is unlikely to make any right or left turns.

Algorithm 8: A3C + Disturbance

```
1 //Decide which noise will be used;
2 Function getNoise(noise); // Specify Gaussian or
   Ornstein-Uhlenbeck (OU) noise
3
4   if noise == Gaussian then
5     | noise ~  $GP(m(x), K(x, x'))$ ;
6   else if noise == OU then
7     | noise ~  $\alpha(\mu - X_t)dt + \sigma dW_t$ 
8 //Given A3C's network policy;
9 Function worker_network()
10  | for policy_network do
11    |  $-\log \pi(a_i|s_i) * A(s, a) + 0.01(-\pi * \log(\pi)) + getNoise()$ ;
12  | for value_network do
13    |  $(r_t + \gamma * V(s') - V(s))^2$ ;
14  | return disturbed_policy_network, value_network
```

- Task 3 - The vehicle is placed randomly in an initial position and aims to reach a specific point of 200m. The vast majority of times, the vehicle will need to turn right or left to reach its destination point.

If $distance_to_the_target_point < initial_distance/2$, the vehicle gains a +1 reward, as well as if it reaches the target point. Also, it receives a -1 reward if it collides with an object or gets away from its destiny ($distance_to_the_target_point \geq initial_distance * 2$). In case of a collision, the episode is finished regardless of the distance travelled. Each training session has a total of 300,000 steps to be performed in up to 1,000 episodes.

We performed our tests with no noise, with Gaussian noise ($\sigma = 0.02$) and with OU noise ($\sigma = 0.02$). We performed Tasks 1, 2, and 3 firstly with the city empty, that is, without dynamic objects. After completing this first phase, we reran all three Tasks with the city populated with 50 vehicles and 100 walkers that were always

moving around in the city.

7.1.1 Tasks with empty city

After running several tests, our experimental results are available below. Therefore, Figure 7.3a presents the average reward obtained in each of the analyzed approaches to fulfill Task 1. Based on this figure, we realize that the OU noise approach generates a better return than the approaches without noise or even with Gaussian noise.

Figure 7.3c shows the average returns of the approaches during the performance of Task 2. Again, the approach with OU noise presents the best results obtained during the Task.

Figure 7.3e represents the models' performance in the most difficult task performed with the empty city, which means without the presence of other dynamic objects, such as vehicles and pedestrians. The training conducted with OU noise had a significant improvement in its reward compared to training without noise or with Gaussian noise.

We also evaluated agent displacement with no noise, Gaussian, and OU noise during each reinforcement learning task. Figure 7.3b shows the normal distribution of the agent's displacement during the first Task in an empty urban environment. Based on Figure 7.3b, the largest displacement is performed by the agent that used Gaussian noise since the variance of its bell curve is the largest. We must consider that the more concentrated on the positive side of the x-axis line, the closer the agent gets to having accomplished 100% of the task objective. Thus, the further the agent

is from the mean going to the negative side of the graph, the further away from the goal specified by the Task the agent is.

From this clarification, we realized that the agent with Gaussian noise was the one that most distanced itself from the target point of Task 1. In this way, the agent with **OU** noise represents the best displacement obtained by the agent during Task 1, even having a minor variance on its curve, but it has the smallest distance from the mean to the negative side of the graph.

The agent displacement needs to be significant, not just wide. For example, the agent can move to a position and then return to the original point, which leads to the low significance of the environment exploration for our tasks. Instead of progressing with the displacement, it keeps going and returning to its origin. Thereby, the agent's displacement with Gaussian noise has this behaviour. Therefore, the agent may even go from position 0 to position 1 at first but returns to position 0 when it should continue rising, for example, to position 2. Figure 3.3 illustrates precisely the behaviour of agents with different types of noise.

The displacement of Task 2 in the empty city, represented in Figure 7.3d, was performed more efficiently by the agent who also used **OU** noise since the variance in the bell curve is the one that moves the least away from the mean to the negative side of the graph.

In the displacement of Task 3, shown in Figure 7.3f, the agent that uses Gaussian noise has the smallest variance among the curves, and it is completely located on the

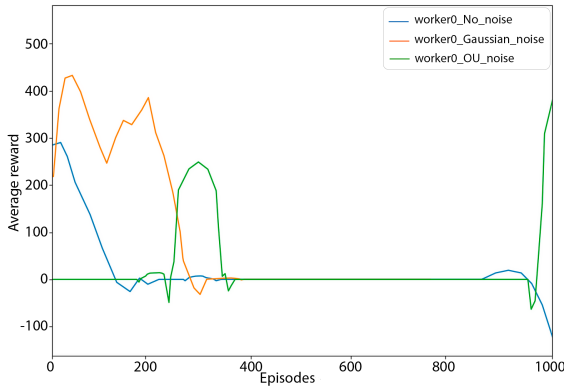
positive side of the figure. The agent with **OU** noise has a great performance, but it moves away to the negative side of the distribution by 2 standard deviations from the mean and has a bigger variance than the agent with Gaussian noise. Thus, the Gaussian noise agent seems to perform a better execution in carrying out the third Task with the city empty.

7.1.2 Tasks with populated city

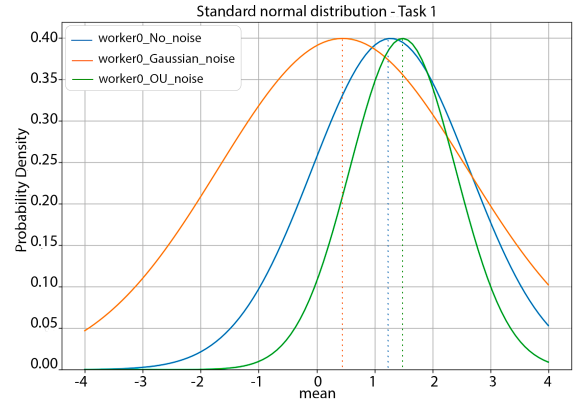
Figures 7.4a, 7.4c, and 7.4e show the average reward obtained by the agent during the training of Tasks 1, 2, and 3, respectively. Figure 7.4a shows that, in general, the agent with **OU** noise got the best return on the first Task with the city populated with dynamic objects.

During the second Task execution with the populated city (Figure 7.4c), the agent with **OU** noise obtained the best performance. Likewise, during Task 3, shown in Figure 7.4e, the agent with **OU** noise also achieved the highest scores.

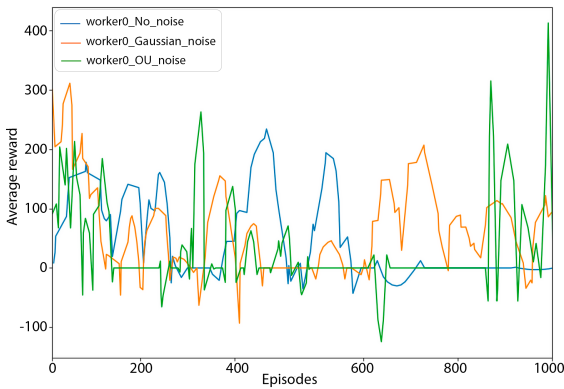
Figures 7.4b, 7.4d, and 7.4f show the normal distribution of displacements travelled by agents in Tasks 1, 2 and 3, respectively. Analyzing Figure 7.4b, we can see that the agent with **OU** noise, even having achieved a better reward as shown in Figure 7.4a, is not the best solution for Task 1 with a populated city, as it concentrates the greatest variance. In this specific case, in which the task requires moving a short distance (5m) with other dynamic objects around it, the agent with Gaussian noise seems to be the best option, as it has less variance and is concentrated much more



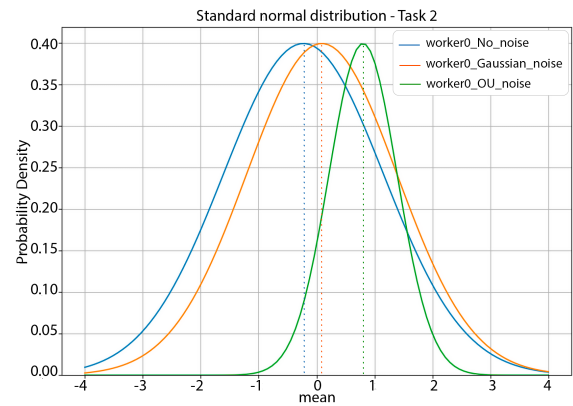
(a) Average reward return during training - Task 1 (5m)



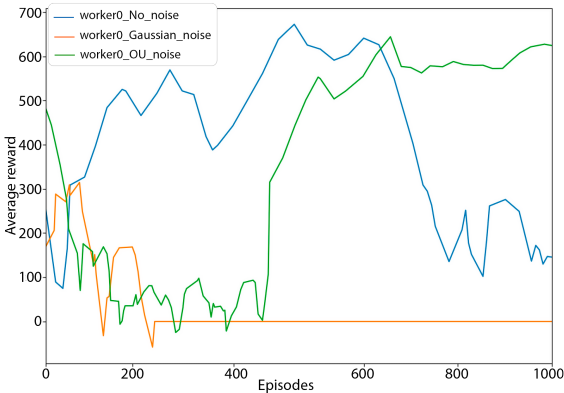
(b) Standard normal distribution for the agent displacement - Task 1



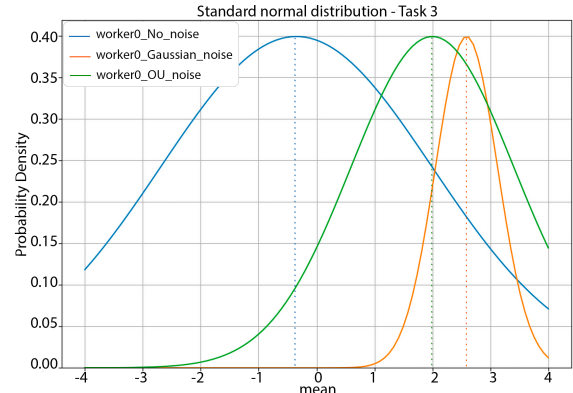
(c) Average reward return during training - Task 2 (30m)



(d) Standard normal distribution for the agent displacement - Task 2

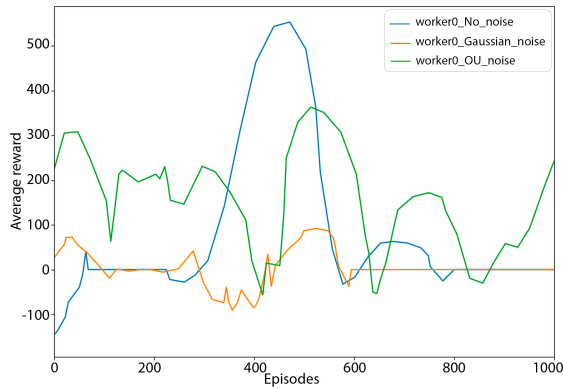


(e) Average reward return during training - Task 3 (200m)

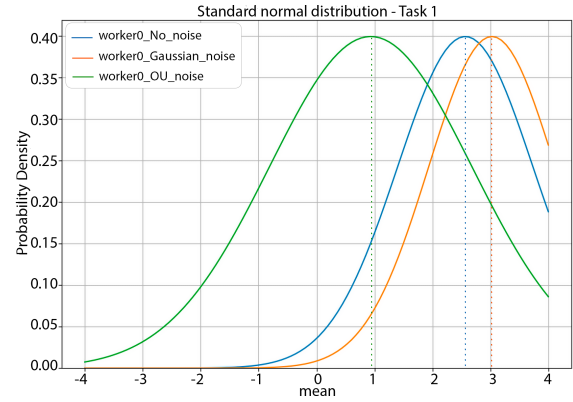


(f) Standard normal distribution for the agent displacement - Task 3

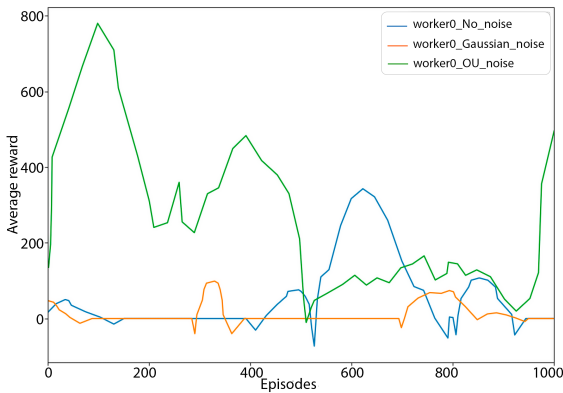
Figure 7.3: Tasks performed with empty city



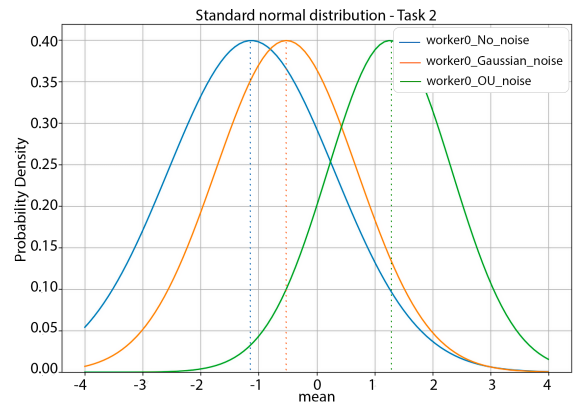
(a) Average reward return during training - Task 1 (5m)



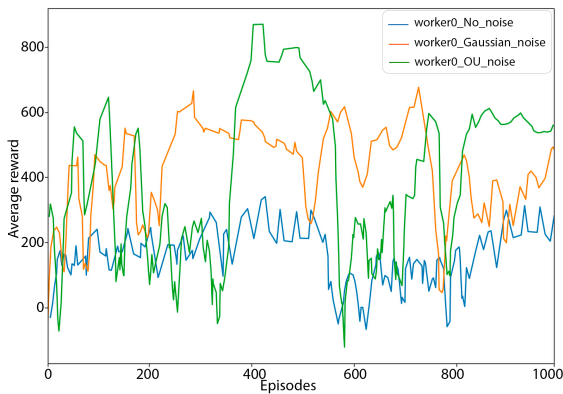
(b) Standard normal distribution for the agent displacement - Task 1



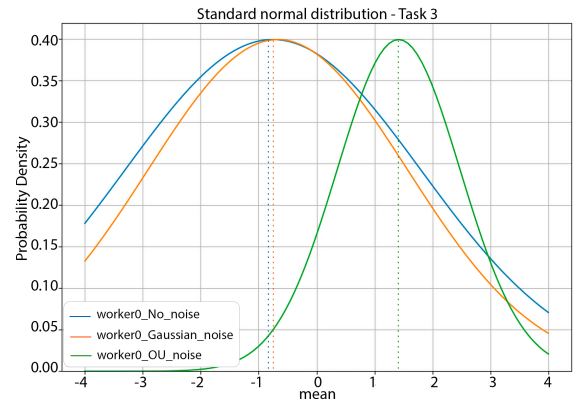
(c) Average reward return during training - Task 2 (30m)



(d) Standard normal distribution for the agent displacement - Task 2



(e) Average reward return during training - Task 3 (200m)



(f) Standard normal distribution for the agent displacement - Task 3

Figure 7.4: Tasks performed with populated city

on the positive side of the graph.

Figure 7.4d, the agents with Gaussian noise and OU noise have very similar variance, but the OU noise curve represents the best displacement for Task 2 with the populated city. This is because that agent's curve is concentrated on the most positive side of the graph and distances from the mean to the negative side by around 2 standard deviations.

In Figure 7.4f, we clearly perceive that the agent with OU noise is the best solution for Task 3, as it is mostly concentrated on the positive side and moves away from the average to the negative side only by 1 standard deviation.

According to the results presented, we can affirm that a good score obtained during training does not always mean the best solution to controlling an autonomous vehicle trained with reinforcement learning. However, in all Tasks, the agent with OU noise proved to be a good solution, although in Task 3 with the empty city and Task 1 with the populated city, the agent with Gaussian noise performed better. Considering the results as a whole, we can conclude that OU noise for the control training of autonomous vehicles seems to be the best solution, presenting the best returns with an efficient exploration of the environment.

Exploring autonomous agents using noise can sometimes lead to discomfort and distrust. However, we have considered several concerns and strategies to mitigate possible risks. One is adjusting the OU noise variance over time as the agent becomes more familiar with its environment. We have also carefully defined the agent's

reward system, ensuring that penalties for dangerous or unwanted actions are significant enough to discourage the agent from taking them. Furthermore, our agents must undergo rigorous testing and continuous performance reviews before and during deployment in real-world environments. Our goal is to strike a balance between the need for exploration to develop an effective policy through reinforcement learning and the security and reliability requirements, ensuring that the resulting agent can adapt and improve and is safe to use in critical applications.

Table 7.1 presents the result of the comparison of our model ([Asynchronous Advantage Actor Critic with Disturbed Policy \(A3C-DP\)](#)) with seven other approaches in three hard exploration games (Private Eye, Gravitar and Pitfall) and the other one score exploit game (Seaquest) from the Atari OpenAI [70]. Our model managed to perform better than the other seven works presented in the Gravitar and Pitfall games. In the Private Eye and Seaquest games, our approach had a higher ranking than Random[96], PPO [68], [Advantage Learning - Deep Q-Network \(AL-DQN\)](#) [97], [Advantage Actor Critic \(A2C\)](#) [98], [A3C](#) [98] and [A3C+](#) [99], trailing only for [Deep Code Search \(DeepCS\)](#) [100] scores in these games.

We proved that using the *Ornstein-Uhlenbeck* (OU) process to perturb the agent’s certainty works much better than the Gaussian process. This improvement is because the OU process is time-correlated, so it always considers the previous noise to increment the current noise. Thus, in our final approach, including the one used to compare with other works already published, we employed only the OU process, which we proved to be also efficient in the comparison tests performed (Table 7.1).

Mean Reward (at convergence)				
	Private Eye	Gravitar	Pitfall	Seaquest
Random[96]	25	173	-229	68
PPO [68]	69.5	737.2	-32.9	1,204.5
DeepCS [100]	1,105	881	-186	3,443
AL-DQN [97]	153	540	-100	–
A2C [98]	91.3	194.0	-55.0	1,714.3
A3C [98]	206	269	-78	2,300
A3C+ [99]	99.32	238.68	- 259.09	2,015.55
A3C-DP	218.31	1,005.52	-0.94	2,380.30

Table 7.1: Comparison with previously published works on hard exploration (Private Eye, Gravitar and Pitfall) and one score exploit (Seaquest) Atari games.

Furthermore, the Atari OpenAI Gym games were chosen for the comparison because it is a reinforcement learning environment widely used and well-known by researchers in the field. In addition, the Private Eye, Gravitar, and Pitfall games are complex environments to explore, being significant challenges for reinforcement learning applied to autonomous agents. Finally, Seaquest is a game of broad exploration, which we have also chosen to demonstrate the efficient performance of our approach to dealing with less complex environments.

7.2 Task Management in Autonomous Systems

In order to compare the centralized and distributed approaches, we utilized the definitions previously explained in the Section 5.1 and Section 5.2. The comparison simulation was conducted using the parameters outlined in the Section 5.2. The key distinction between the approaches is illustrated in the Fig. 7.5, where the distributed approach displays vehicle communication while the centralized approach does not.

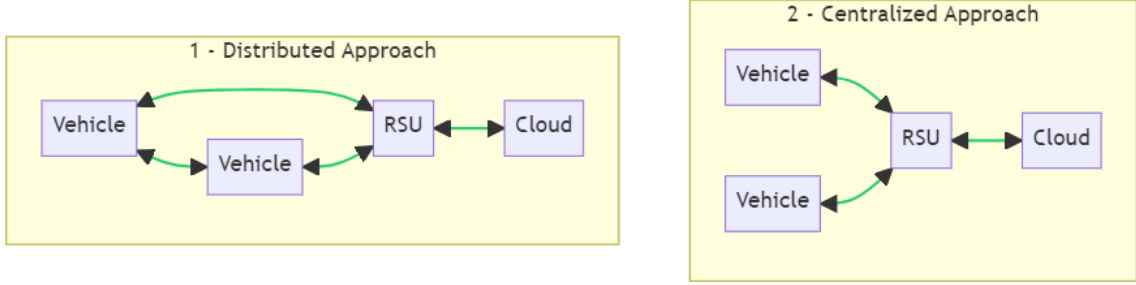
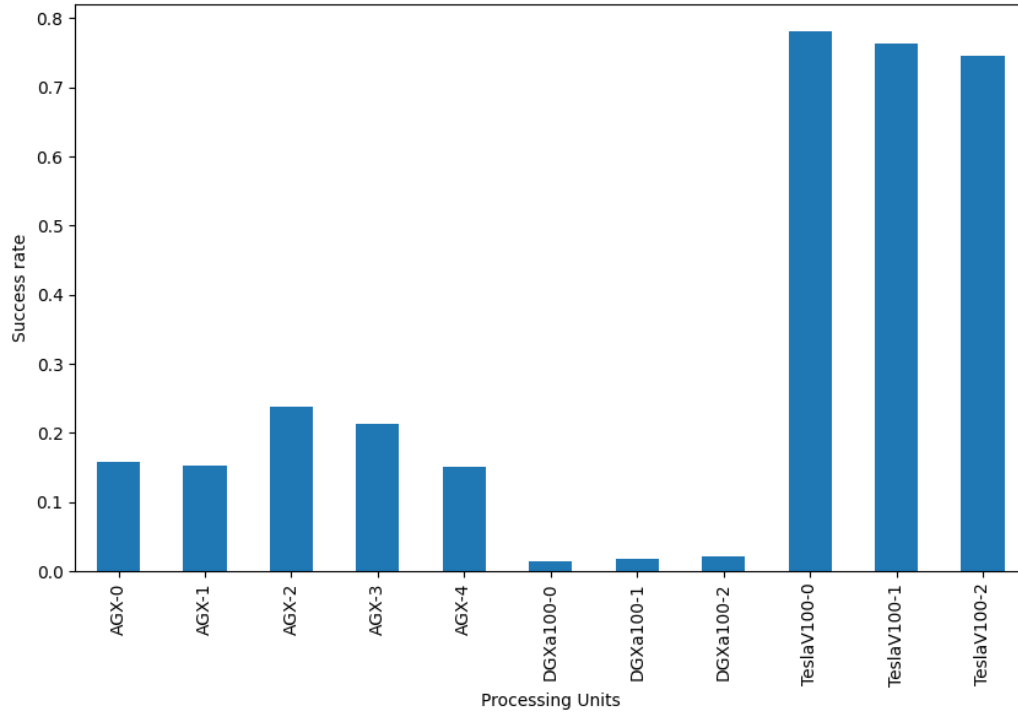


Figure 7.5: Comparison between distributed and centralized approaches

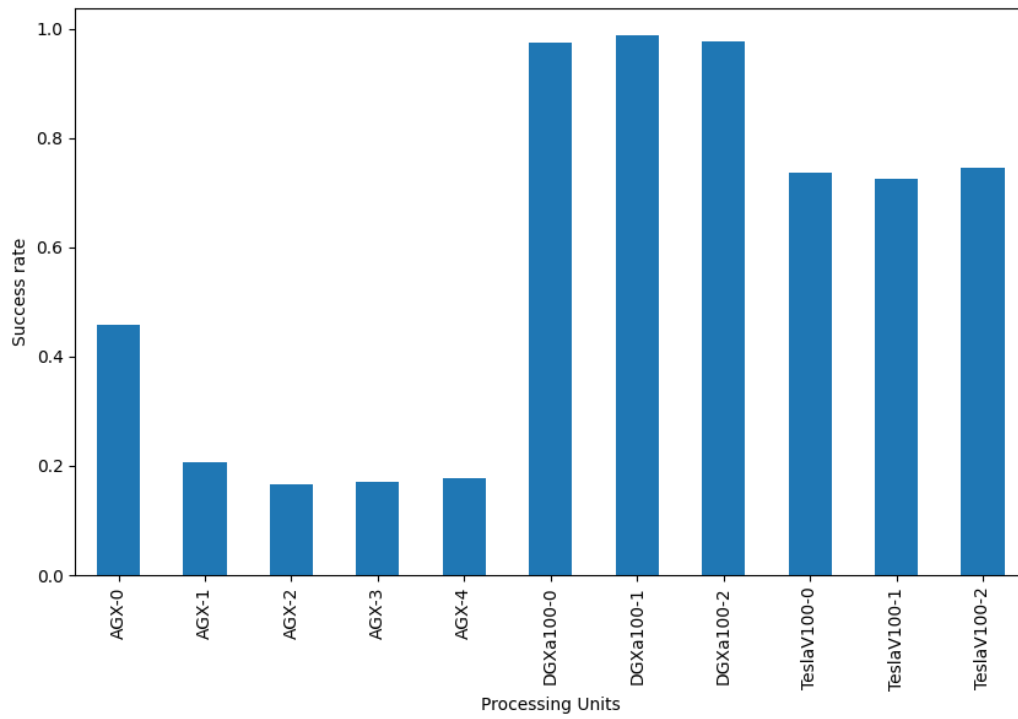
With data processing and resource sharing between **Over-The-Air (OTA)** devices, we must highlight data security concerns from hacker attacks and malicious intrusions, as detailed in [101], [102]. However, the cybersecurity approach in **MEC** networks is outside the scope of this research. Still, we stress the importance of implementing robust security measures across all system components, from edge devices to the task orchestrator. These may include data encryption, user authentication, access control and constant monitoring for suspicious and malicious activity [103], [104].

Fig. 7.6 presents a comparison between centralized (Fig. 7.6a) and distributed (Fig. 7.6b) approaches to the analysis of task success by **Processing Unit (PU)**. In Fig. 7.6, we noticed that **PUs** from the distributed approach are more successful than **PUs** from the centralized approach, which only has a higher success rate in TeslaV100 **PUs**.

In the Fig. 7.7, we can see the number of successful and failed tasks, categorized by their criticality level. The graph includes data for both the centralized (Fig. 7.7a) and the distributed (Fig. 7.7b) approaches. From analyzing Fig. 7.7, we can conclude that the distributed approach has a higher success rate than the centralized approach.



(a) Centralized approach



(b) Distributed approach

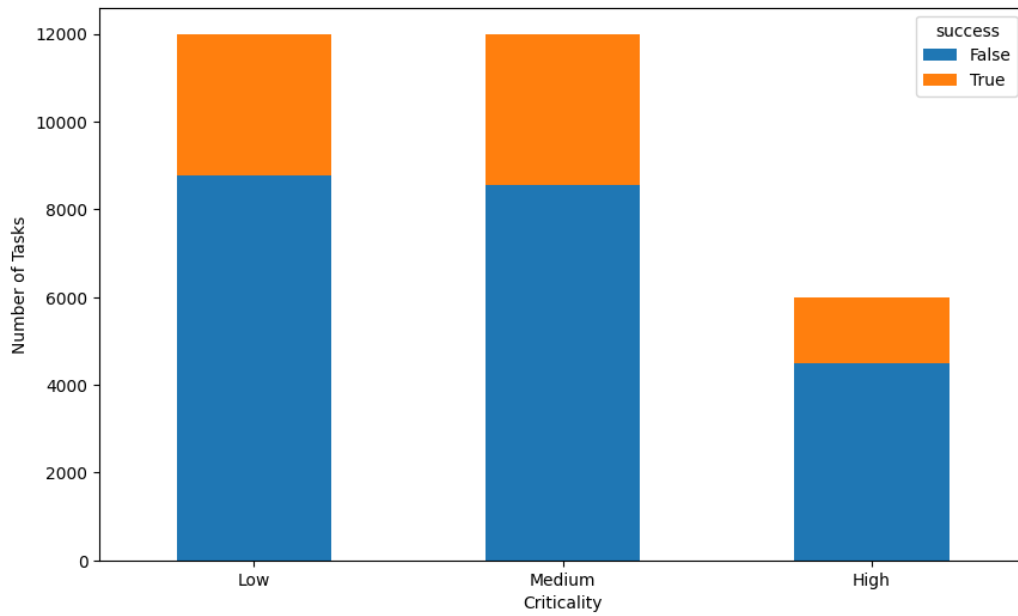
Figure 7.6: Success Rate by Processing Units

Additionally, the most critical tasks in Fig. 7.7b have a higher success rate, which is crucial for safe-critical systems as these tasks should have the highest priority.

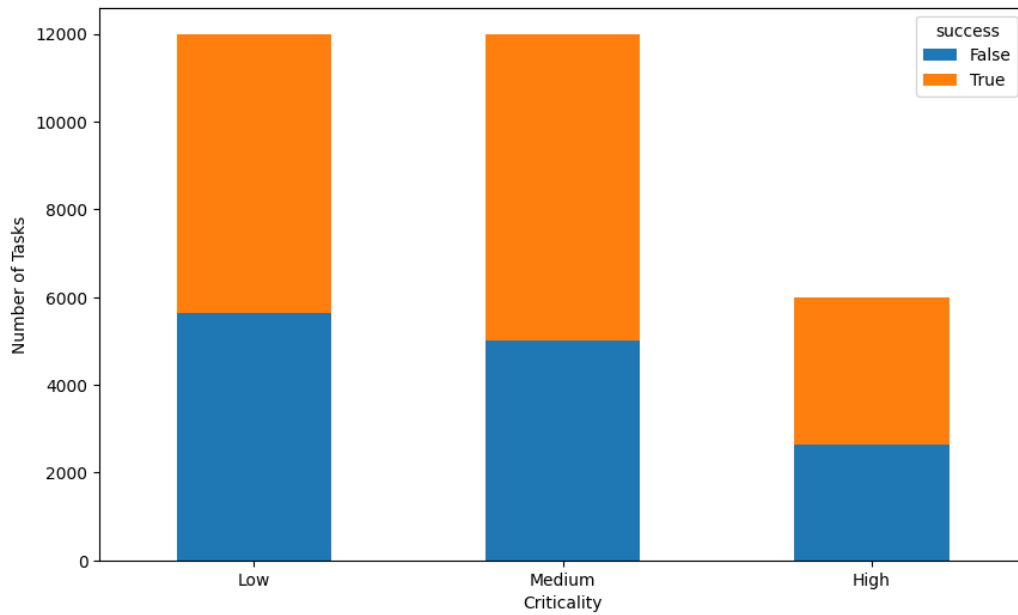
The Fig. 7.8 displays the success rates of the different task types, including ObjectDetectionTask, ObjectTrackingTask, and MappingTask. The distributed approach shown in Fig. 7.8b has the highest success rates for each task type. In conclusion, after analyzing the results present in Fig. 7.6, Fig. 7.7 and Fig. 7.8, we conclude that the distributed approach has the best performance in our data and resource sharing scenario for the autonomous vehicles. This is because when the approach is distributed and shared among vehicles, edge and cloud, the availability of resources for processing tasks is much greater than when each vehicle has access only to its own resources, to the edge and cloud.

7.3 Learning and Performance Monitoring of Autonomous Systems

DDQN [105] is an improvement of the original Deep Q-Network (DQN) algorithm and addresses the problem of Q-values overestimation by using two separate neural networks to update and evaluate the Q-values. This technique helps stabilize learning and improve model convergence. By combining DDQN with Prioritized Experience Replay (PER) [106], the algorithm adds a priority layer to the experiences stored in the replay buffer. This priority is based on the prediction error, which is the difference between the predicted and target Q-values. Experiences with higher errors are more likely to be selected for training, allowing the agent to learn more quickly from these experiences. Accordingly, we will use the DDQN-PER in our work as a baseline. It

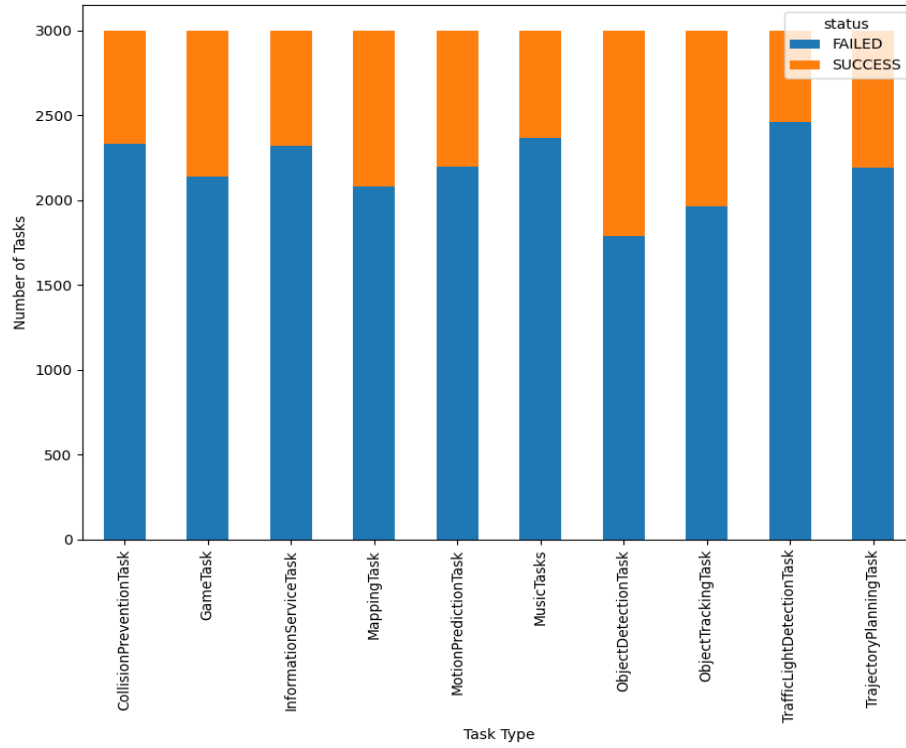


(a) Centralized approach

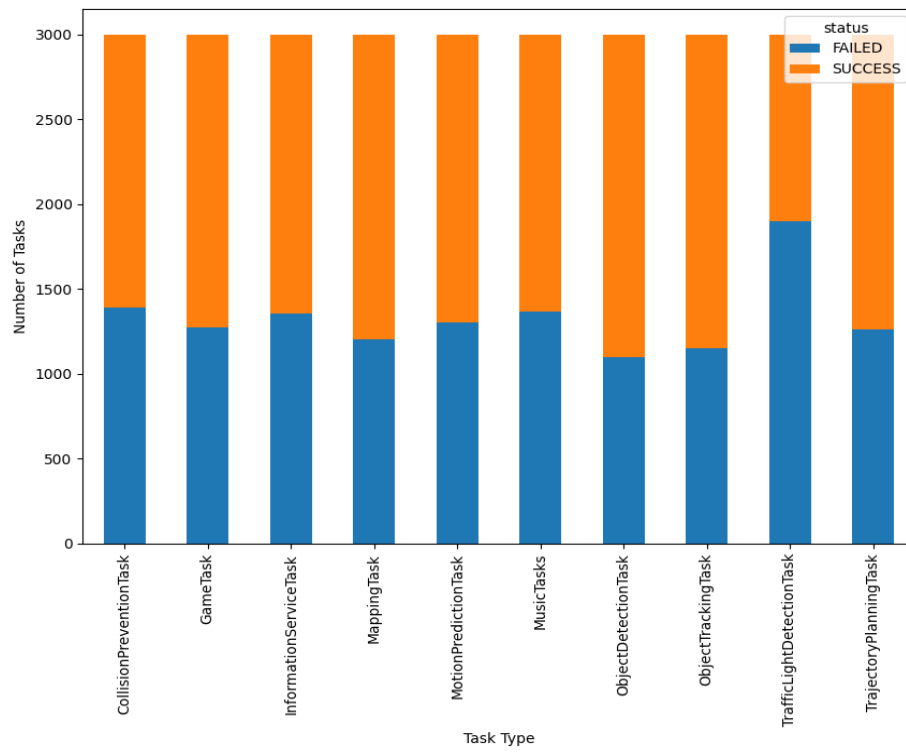


(b) Distributed approach

Figure 7.7: Task Success by Criticality



(a) Centralized approach



(b) Distributed approach

Figure 7.8: Task Success Analysis

learns from more relevant experiences and reduces the overestimation of Q-values, resulting in more efficient and stable reinforcement learning.

We also use the [DDQN](#) plus [PER](#) approach with uncertainty estimation (DDQN-PER with uncertainty) to compare with our baseline (DDQN-PER). For simplicity, when we refer to the [DDQN](#) form in this section, we point to DDQN-PER. So our baseline, ‘DDQN-PER’, will be treated as just [DDQN](#), and our proposed ‘DDQN-PER with uncertainty’ will be [DDQN with uncertainty](#). An uncertainty-aware machine learning approach is crucial to balancing exploration and exploitation. For example, when an agent is uncertain about the quality of its actions, it needs to explore more of the environment to gather additional information and improve its estimations, so its actions improve in the face of uncertainties [107]. Also, when uncertainty is low, the agent can concentrate on exploiting the actions it already knows well. Thus, effective uncertainty management can create a better balance between exploration and exploitation and, consequently, a more optimized policy.

Our objective in this study is to present information visually and graphically to explain the behaviour and the motivations of the agents ‘[DDQN](#)’ and ‘[DDQN with uncertainty](#)’ for their decision-making. Explainable Artificial Intelligence (XAI) is a study field that seeks methods and techniques that can make artificial intelligence (AI) models and systems more understandable and interpretable for humans [108]. [AI](#) agents have become highly capable with the increasing complexity of decision algorithms, but their decisions are becoming more challenging to comprehend. This has led to a growing demand for transparent, explainable, and easily interpretable

AI models. By understanding the behaviour and actions of an autonomous agent, we are able to trust more in their decisions and accept the proposed model [109]. Another point to consider is that from the understanding of model decisions, experts and developers can identify and correct possible errors or biases of this model, thus improving its performance and effectiveness [109].

The most common methods used for explainability models include natural language, saliency map, and gradients heatmap [109], [110]. However, explainability approaches typically use only one or a few of these methods. Therefore, to develop our explainability proposal, we are integrating a screen for rendering the environment as seen by humans; a screen with the representation of the agent's view, considering that it receives an array as data input; the Q-values referring to each possible action; the state value ($V(s)$) over time, where s is the state pointed out; a heatmap relative to the agent's input, so it is possible to see which features had more weight for decision making; and, finally, a visual presentation of the uncertainty associated with each Q-value. By quantifying model uncertainty, we can provide additional information about the predictions and decisions made by the model, improving human understanding and confidence in that model. For example, suppose our agent is making a decision based on a Q-value of the action with high uncertainty. In that case, we can inform that this decision can be unreliable and that more training and exploration are needed to improve the model's reliability. The proposal in Figure 7.9 suggests a generalized way to estimate Q-value and uncertainty by calculating predictions' mean

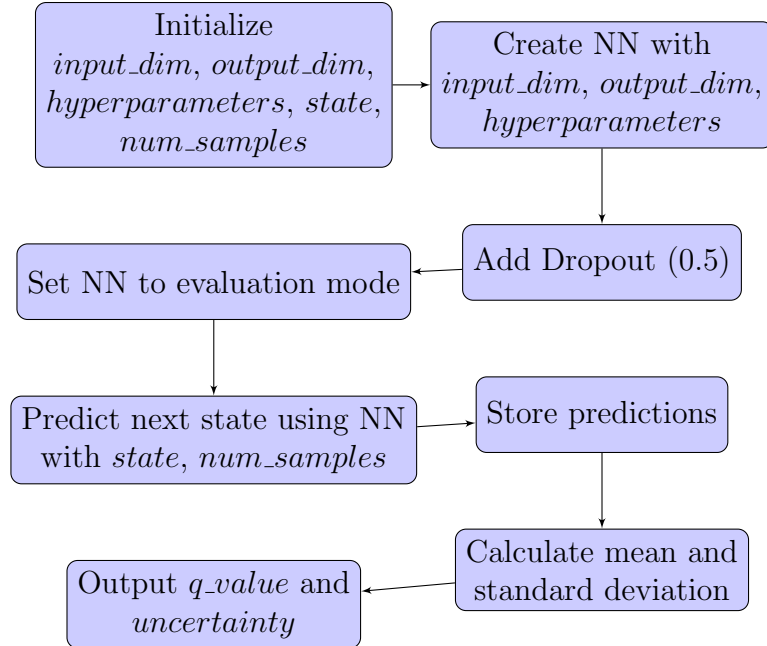


Figure 7.9: Abstraction of our proposal [DDQN](#) with uncertainty

and standard deviation.

Accordingly, to estimate the uncertainty of actions, we are combining deep learning, through a neural network and Bayesian inference, by calculating the mean and standard deviation of predictions, for the composition of Bayesian deep learning as presented by the Algorithm 9. As stated in the Algorithm 9, the expected output is *q_value* and *uncertainty* related to each action that the agent can perform. For this, we create a neural network NN that receives as input *input_dim*, *output_dim* and *hyperparameters*. In addition, one of the passed parameters is a dropout value equal to 0.5, added after each dense layer in the Q network architecture, creating a Bayes dropout network [111] necessary to estimate the uncertainty in the Q-value predictions.

Then we have the function *predict_bayesian*, which uses the neural network NN

Algorithm 9: Bayesian deep learning to get action uncertainty - [DDQN](#) with uncertainty approach

Input: $input_dim$, $output_dim$, $hyperparameters$, $state$, $num_samples$
Output: q_value and $uncertainty$

- 1 **Function** `create_NN`($input_dim$, $output_dim$, $hyperparameters$):
 - 2 $NN \leftarrow$ create new neural network with $input_dim$, $output_dim$, and $hyperparameters$
- 3 **Function** `predict_bayesian`($state$, $num_samples$):
 - 4 Set the neural network to evaluation mode
 - 5 Adjust the dimension of the tensor $state$ to be 2-dimensional
 - 6 Initialize an empty list $predictions$
 - 7 **for** $i = 1$ **to** $num_samples$ **do**
 - 8 $prediction \leftarrow$ predict $state$ using the neural network
 - 9 Append $prediction$ to $predictions$
 - 10 Set the neural network back to training mode
 - 11 Compute the mean and standard deviation of $predictions$
 - 12 $q_value \leftarrow$ mean of $predictions$
 - 13 $uncertainty \leftarrow$ standard deviation of $predictions$

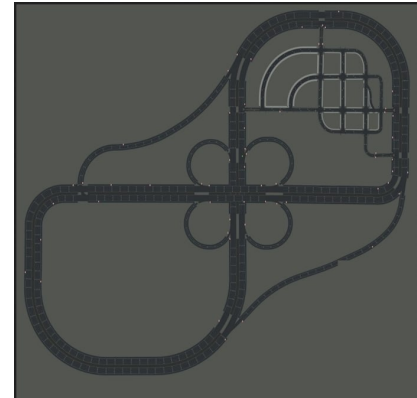
to make predictions based on the given $state$. These predictions are stored in a list for calculating and returning the mean and standard deviation. The standard deviation value, mentioned here, represents the uncertainty [42], [112] that the agent has about a given action (Q-value). The standard deviation provides a dispersion measure of forecasts around the mean, so the more significant the standard deviation, the greater the uncertainty in the action (Q-value) forecast. With that in mind, the uncertainty calculation performed inside Algorithm 9 can be represented by Equation (2.3).

Thus, we use this measure of uncertainty to guide the choice of action to be taken by our agent, ‘[DDQN](#) with uncertainty’, which makes it more cautious in situations where its predictions are uncertain. Using model uncertainty as one more explainability mechanism helps users understand which actions the model has less confidence in and assess the impact of that uncertainty on agent performance. Also,

users can better understand model limitations by incorporating uncertainty into [XAI](#).



(a) Sample view of CARLA simulator scenario



(b) Simulated city map layout (Town 04)

Figure 7.10: Simulation environment

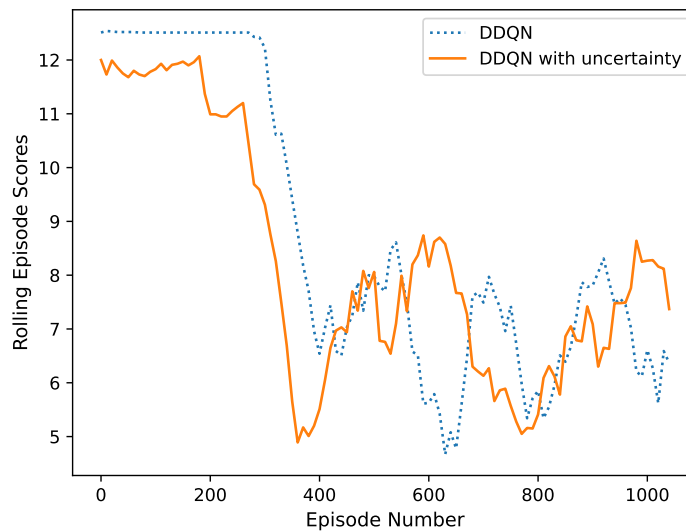


Figure 7.11: Scoring performance comparison between approaches with and without uncertainty

We utilized the CARLA simulator [113], an open-source platform created for autonomous driving research, to train and test two agents - [DDQN](#) and [DDQN](#) with uncertainty - for obstacle avoidance over 1000 episodes. Figure 7.10 shows a visual representation of the simulator. Thus, Figure 7.10a shows an example of an obstacle

avoidance scenario, while Figure 7.10b displays the city map, specifically town 04, that we used in this study.

A comparison of the obtained reward of the two algorithms is shown in Figure 7.11 below. Analyzing this Figure 7.11, we notice that the results of the training scores are pretty similar, with the ‘DDQN’ agent gaining an advantage at the beginning of the training and the ‘DDQN with uncertainty’ agent doing better towards the end. Table 7.2 presents the detailed specifications for both trained agents.

Table 7.2: Training details of evaluated agents

Learning rate	0.1
Batch size	256
Buffer size	20000
Epsilon	1.0
Epsilon decay rate denominator	1.0
Discount rate	0.9
Tau	0.01
Alpha prioritised replay	0.6
Beta prioritised replay	0.1
Incremental temporal-difference error	1e-08

We have created a proposal to explain why the agent makes certain decisions, and we have provided a detailed explanation in Figure 7.12. The red square labelled “1” shows the original environment as seen by humans. The red square labelled “2” displays the Q-values for each possible action. Thus, the redder the square, the higher the Q-value; the bluer the square, the lower the Q-value. The red square labelled “3” indicates the level of uncertainty associated with each Q-value. Therefore, the bars are longer and redder when there is more uncertainty and shorter and bluer when there is less uncertainty. The uncertainty in this work ranges from 0 (no uncertainty) to 1 (maximum uncertainty). Red square “4” provides an estimated value ($V(s)$) over

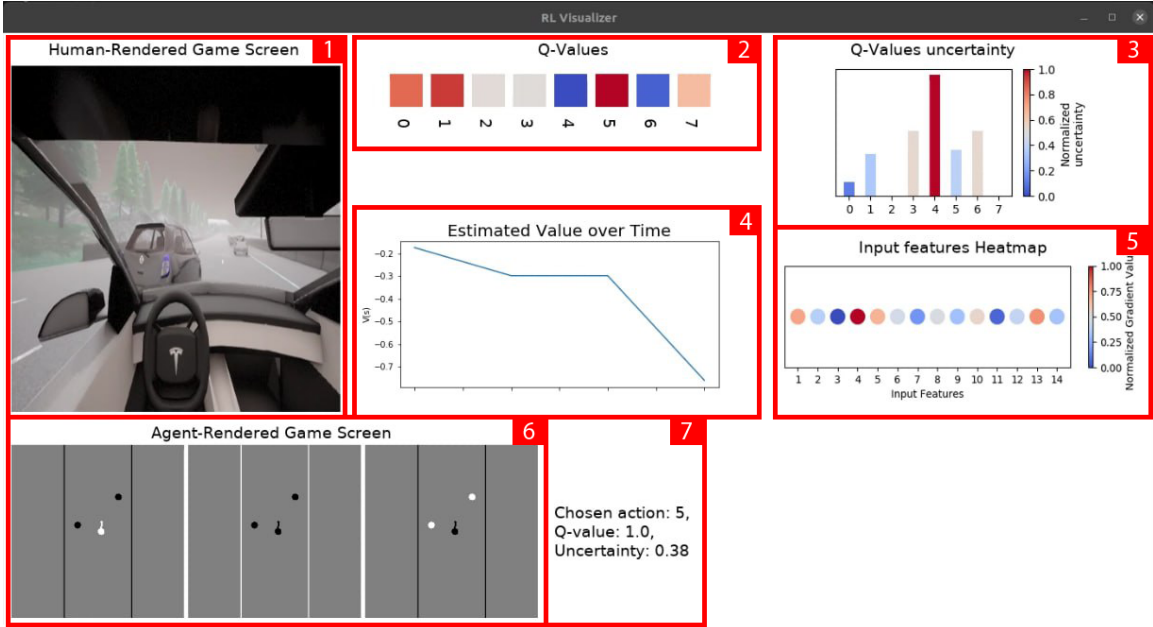


Figure 7.12: Details of the explainability screen in our proposal

time for the state, which describes the expected cumulative value of future rewards. The red square labelled “5” illustrates the input features that are most important when deciding on an action. The redder the circle, the more relevant the feature is, and the bluer the circle, the less relevant the feature is. The red square “6” presents the agent’s view of the environment in which it operates, so in the first figure of square “6”, from right to left, we have the surrounding detected obstacles represented as white circles. Next, we have information that defines drivable areas and lane markings represented by white lines. Moreover, in the last figure referring to square “6” from right to left, we see the ego vehicle as a white circle indicating its direction with a white arrow attached to that circle. Finally, the red square “7” gives the coming action, along with the Q and uncertainty values related to this desired action.

In Figure 7.12, square “6” represents how the agent views its surroundings. It

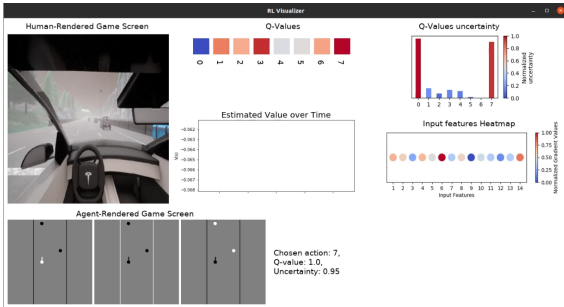
is divided into three smaller squares to show the agent’s attention order. First, the agent identifies obstacles (right frame), then the drivable area (middle frame), and finally, its own position and direction in the environment (left frame). The input features, represented by numbers 1 to 14, and actions, represented by numbers 0 to 7 in Figure 7.12, are explained below. Input features are the information the agent receives from the environment during each interaction.

input features = 1 : *Ego location (Y axis)*
= 2 : *Ego velocity (Y axis)*
= 3 : *Ego rotation (yaw)*
= 4 : *Ego angular velocity (Z axis)*
= 5 : *Left obstacle location (X axis)*
= 6 : *Left obstacle location (Y axis)*
= 7 : *Left obstacle location (Z axis)*
= 8 : *Right obstacle location (X axis)*
= 9 : *Right obstacle location (Y axis)*
= 10 : *Right obstacle location (Z axis)*
= 11 : *Left obstacle velocity (X axis)*
= 12 : *Left obstacle velocity (Y axis)*
= 13 : *Right obstacle velocity (X axis)*
= 14 : *Right obstacle velocity (Y axis)*

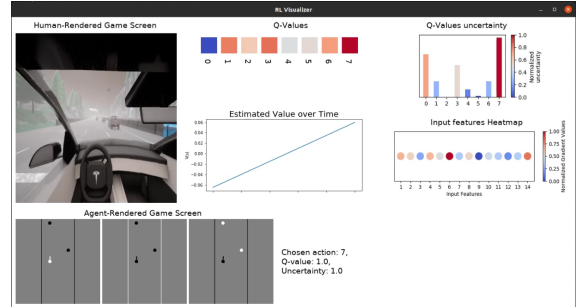
actions = 0 : [0.1, -0.5, 0.], *Turn Left*
= 1 : [0.1, 0.5, 0.], *Turn Right*
= 2 : [0.7, 0., 0.], *Forward*
= 3 : [0.5, -0.5, 0.], *Accelerate & Turn Left*
= 4 : [0.5, 0.5, 0.], *Accelerate & Turn Right*
= 5 : [0.5, 0.1, 0.], *Accelerate & Bear Right*
= 6 : [0.5, -0.1, 0.], *Accelerate & Bear Left*
= 7 : [0.5, 0., 0.], *Small Acceleration*

Figure 7.13 presents the result of viewing the behaviour of the agent ‘DDQN’ after training for 1000 episodes. Observing Figure 7.13a, we see that the first action to be taken is 7 (small Acceleration), with the maximum Q-value and uncertainty of 0.95. In addition, the agent focused mainly on features in order of priority 6, 14, 1 and 4 (left obstacle location (Y axis), right obstacle velocity (Y axis), ego location (Y axis) and ego angular velocity (Z axis)). Observing the figures from 7.13a to 7.13f, we realize that actions are taken based on the maximum value of Q and that the uncertainty of the Q-values is not taken into account. Another interesting aspect is that, based on the estimated value over time, we can follow the expected value of future rewards over time, noticing that this forecast decreases, indicating that the agent is not making the right decisions. Finally, in the last frame 7.13f of Figure 7.13, we see that even though the agent sees the vehicle in front of it, as it appears represented in the agent-rendered screen, the agent does not try to make a detour and then choose action 3 (Accelerate & Turn Left).

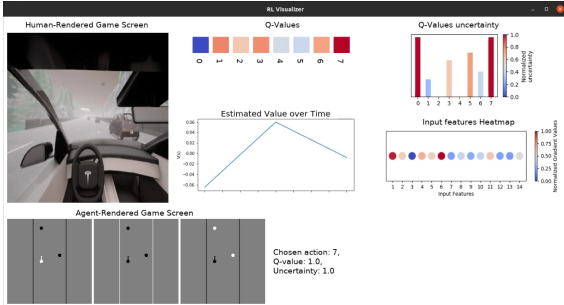
Figure 7.14 shows the behaviour and motivations of the agent ‘DDQN with uncertainty’ to avoid obstacles after training. Analyzing the pictures from 7.14a to 7.14e, we realize that the agent considers the highest value of Q, which also has the lowest uncertainty value. Observing the estimated value over time, which shows the expectation of future reward, we see that, initially in pictures 7.14b and 7.14c, this estimate of reward is decreasing but starts to grow from of the frame 7.14d. This fact means that the agent has improved its decision-making over time, as its objective is to learn the policy that maximizes the expected value of future rewards, and this does not



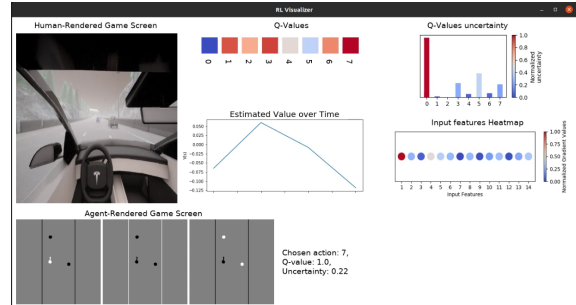
(a) Observation of initial action taken



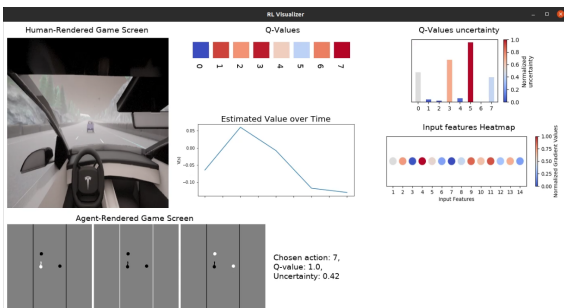
(b) Observation of the second action taken



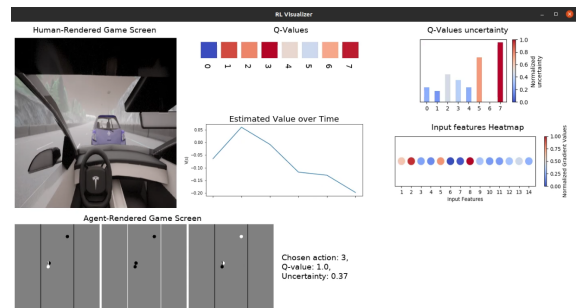
(c) Observation of the third action taken



(d) Observation of the fourth action taken



(e) Observation of the fifth action taken

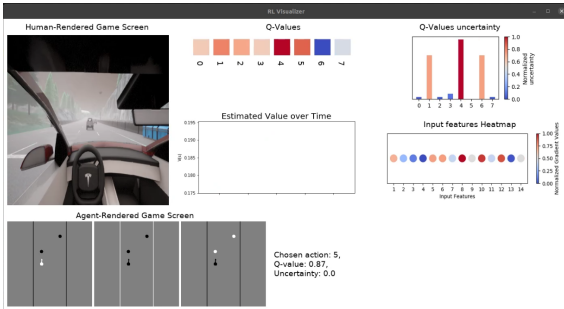


(f) Observation of the sixth action taken

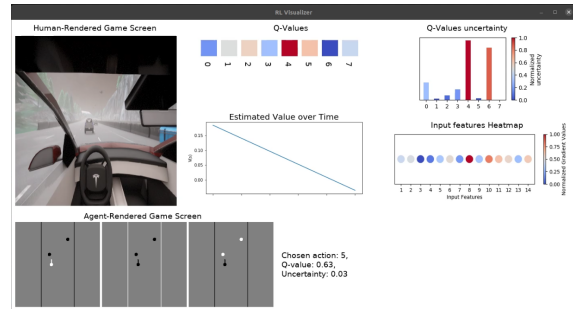
Figure 7.13: Explanation of the behaviour and actions of the DDQN agent in the obstacle avoidance scenario

always mean choosing actions that lead to states with the highest possible values in all situations. According to the information in the figure, we can see that our agent ‘DDQN with uncertainty’ can avoid the first obstacle that appears in front of it, but it has difficulties bypassing the second. Analyzing the frames 7.14d and 7.14e and comparing the human-rendered screen with the agent-rendered screen, we see that in the human-rendered screen, the agent does not seem to have deviated from the vehicle in front of it. However, on the agent-rendered screen, in addition to the space between the agent and the obstacle that seems to be bigger than it really is, we can see an attempt to rotate to the left with the move forward action, chosen in the last frame 7.14e.

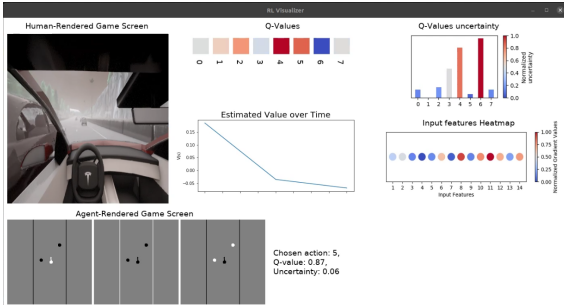
Comparing Figures 7.13 and 7.14, we can say that agent ‘DDQN with uncertainty’ in Figure 7.14 performed better than agent ‘DDQN’ in Figure 7.13, as this last one was unable to dodge any obstacles in front of it. Meanwhile, agent ‘DDQN with uncertainty’ managed to avoid the obstacle by considering only high Q-values whose uncertainty value is less than or equal to 0.6. Moreover, the input features considered most relevant by the ‘DDQN with uncertainty’ agent are consistent with its chosen actions. On the other hand, some input features considered relevant to the ‘DDQN’ agent do not match the action. For example, the relevance that the ‘DDQN’ agent gives to input feature 1 (Ego location (Y axis)) when choosing action 7 (small acceleration) in Figure 7.13d. With the proposed explanation of the agent’s behaviour and actions, we can understand how the agent thinks, which helps identify and correct possible errors or biases in the model, thus improving its performance



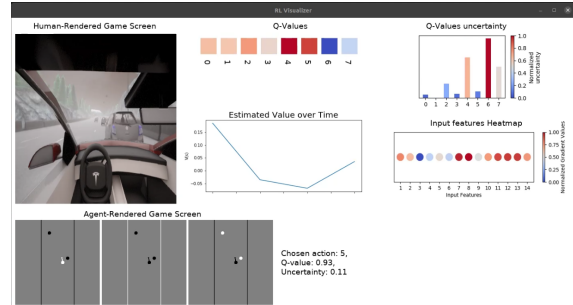
(a) Observation of initial action taken



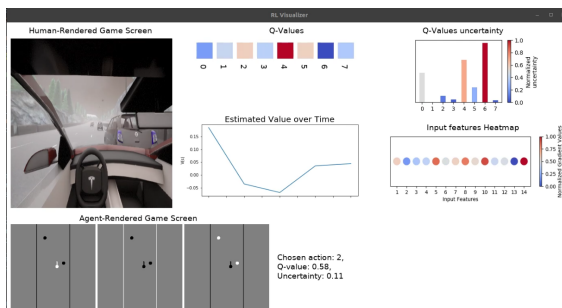
(b) Observation of the second action taken



(c) Observation of the third action taken



(d) Observation of the fourth action taken



(e) Observation of the fifth action taken

Figure 7.14: Explanation of the behaviour and actions of the DDQN with uncertainty agent in the obstacle avoidance scenario

and effectiveness.

In Figure 7.15, we can see why the agent made the wrong decision. The human-rendered screen shows that the agent was too close to the vehicle in front. However, when looking at the agent-rendered screen, we see that there was actually more space between the obstacle and the agent, making it appear closer to the side of the obstacle rather than behind it. By analyzing the heatmap of input features, we can see that the most relevant feature for decision-making was the Ego angular velocity (Z axis), followed by the Right obstacle velocity (X axis), Ego location (Y axis), and Left obstacle location (X axis). It shows that the left obstacle was given the most minor importance in the decision-making process. Despite the chosen action (accelerate & bear right) having a maximum Q-value and an uncertainty value lower than 0.6, it can still result in a negative outcome: a collision with the obstacle ahead.

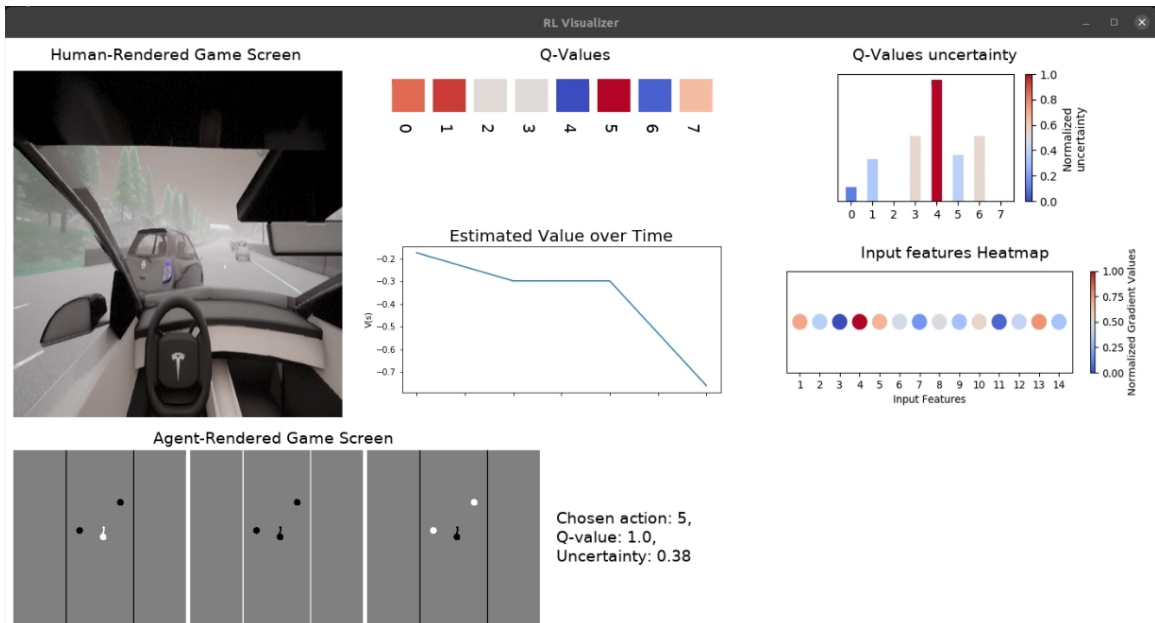


Figure 7.15: Agent error explanation

Table 7.3 compares the works related to our proposal. The points considered

for the analysis of each work are whether the approach offers a visualization of the original environment (as seen by human beings), a visualization of what the agent’s view of that same environment is like, a presentation of the action to be taken by the agent, explanation of this action performed and the type of environment used in each proposal (if the environment is more realistic or some game).

Table 7.3: Comparison of our approach and related works

Study	Original env.	Agent vision of the env.	Taken action	Taken action explanation	Test environment
[55]	✓	✗	✓	✓	Grid-world, Seaquest (Atari), HalfCheetah (MuJoCo)
[56]	✗	✗	✗	✗	12 different games
[57]	✓	✗	✗	✓	Chess, Atari and Go games
[114]	✓	✗	✗	✓	Ms. Pacman game
[58]	✓	✓	✗	✗	CARLA simulator
Our	✓	✓	✓	✓	CARLA simulator

Analyzing Table 7.3, we noticed that most of the approaches used games to test their proposals, only the work [58] and our approach used more realistic scenarios, CARLA simulator. Most works also do not show how the agent sees the environment, as it does not see it like humans. In summary, only our proposal satisfies all the points analyzed in the table.

Chapter 8. Conclusion and Future Works

8.1 Conclusion

In conclusion, this dissertation results from extensive research, experiments and evaluations that resulted in the development of eight scientific articles. These publications discuss the significance of context awareness in enhancing the effectiveness of autonomous systems. They also explore research and application of various techniques to increase context awareness. Furthermore, our studies investigate effective and efficient ways of processing tasks remotely, knowing that some autonomous systems, such as autonomous vehicles, have limited local resources. Another significant contribution of this research is the proposal to quantify uncertainty in predictions made by [ML](#) algorithms and how we can use this uncertainty to explain the decision-making and behaviour of these algorithms, contributing to raising human confidence in [ML](#) predictions.

We start presenting the use of noise as a valuable and efficient resource to encourage the exploration of autonomous agents. That is crucial because the [RL](#) approach helps solve machine learning problems through self-exploration and when training samples are not provided. In this way, knowing more and better its environment, the autonomous agent becomes effectively aware of its context, which enables it to make better decisions. Thus, we fuse the application of [OU](#) and Gaussian noises with the [RL](#) agent policy network. This approach's advantage is the decrease in the autonomous agent's action certainty for the training phase, which will take more

stochastic actions and, consequently, better exploit its training environment. Furthermore, that fact favours a better response of the agent to unusual situations in the environment. We present the potential and superiority of stochasticity in exploring using autocorrelated temporal noise generated by the *Ornstein-Uhlenbeck* process, improving the autonomous vehicle's exploration, as well as other autonomous agents' exploration, in the control tasks that provided better rewards.

Continuing with our study, we developed an edge-centric workload orchestration proposal that takes advantage of the benefits offered by machine learning. In this way, we managed the limited onboard processing capacity and minimized the response time for connected autonomous vehicles. After this, we progressed our research toward the development of a collaborative task management system for [Connected Autonomous Vehicles \(CAVs\)](#) using edge, fog, and cloud computing. Then, we compared our proposed approaches with other methods and found that our techniques performed best in predicting the server service time and executing tasks within the deadline. We also concluded that our proposed distributed approach had a better performance when compared to our previous centralized one.

Also, we developed a model that considers [ML](#), sensitivity analysis and uncertainty measurements for complex events composition for classification and regression problems. We have found that conformal prediction with machine learning models and sensitivity analysis allows for more accurate predictions of potential failures. We also proposed enhancing the transparency and interpretability of decision-making processes in autonomous systems by integrating uncertainties in [XAI](#). We achieve this

by combining Bayesian deep learning with uncertainty-aware planning. To demonstrate the impact of these uncertainties on the [RL](#) agent’s choices, we use the [DDQN](#) algorithm as a reference and present graphs that visually depict the uncertainty estimates.

This research has resulted in several significant contributions to the field of autonomous systems. The proposed methodologies and models increase not only the context perception of these systems but also present effective ways of managing remote tasks and improving the decision-making process by incorporating uncertainties. We demonstrate how the use of noise can benefit the learning of autonomous agents and how a workload mapping approach can optimize the efficiency of connected autonomous vehicles. Furthermore, our conformal prediction model and sensitivity analysis, combined with integrating uncertainties into explainable [AI](#), have proven to be effective in predicting potential failures and improving the transparency and interpretability of autonomous systems. We believe that these advances will substantially contribute to the advancement and widespread acceptance of autonomous systems in several practical applications, and we trust that this work will serve as a solid basis for future investigations in the area.

8.2 Future Work

As the work in [\[115\]](#) highlights, efficient exploration remains one of the most significant issues in reinforcement learning, and every improvement made contributes to

increasing performance in this field. Therefore, in future works, more tests and experiments need to be performed to analyze the power of noise generated by the [OU](#) process in increasing [RL](#) agents' performance in both continuous and discrete action space. For the next steps, we should study how noise volatility acts during agent training in different contexts and domains, such as autonomous vehicles. Besides, we need to verify how the mean reversion speed variation can positively or negatively affect the agent.

One of our limitations is related to the fact that we have not tested in real-world environments. However, instead, we use simulators with different types of simulations containing tasks with different difficulty levels. For example, we could have developed an autonomous car prototype using technologies such as Arduino, raspberry and Jetson, and different environment models where this car could perform some tasks. Nonetheless, developing these tools would take too much time, limiting the time for research application. Therefore, we used different simulators with tasks of varying levels of complexity, ranging from pendulum and Atari games to humanoids and vehicles.

In order to use our work in the real world, it is indispensable to use real-world task datasets from automobiles in actual conditions, like trips from point A to point B, including scenarios like traffic jams and accidents. Additionally, we can expand our approach to other types of non-terrestrial automobiles, like drones and ships, when considering mapping and scheduling tasks.

It is important to note that our work does not cover cybersecurity in [MEC](#) net-

works because it is outside our scope. However, we stress the significance of implementing strong security measures in all system components, including mobile devices, edge, fog, and cloud. This includes ensuring secure communication between them and implementing measures like data encryption, user authentication, access control, and consistent monitoring for any suspicious or malicious activity. The papers [103], [104] have highlighted a handful of measurements as essential.

To quantify uncertainty and its use in explaining algorithm actions and behaviour, we plan to explore how to incorporate other types of data, such as camera footage, social media, and sensor data, into our proposal. Another way to expand on this research is by exploring other methods of uncertainty, like hierarchical deep Bayesian neural networks. These techniques can help improve the accuracy of uncertainty predictions. Additionally, we plan to test our proposal with real users to confirm that they understand the explanations provided.

Another limitation of our work to be raised is the fact that we have yet to test our visual representation model that shows how uncertainty influences the agent's behaviour and decisions with other types of machine learning algorithms, such as supervised and unsupervised learning. Our research mainly focuses on contextual learning and how autonomous agents interact with their environment. Because of this, we primarily consider reinforcement learning and its challenges. However, we can evaluate our approach to other types of learning in future studies.

References

- [1] H. Zhu, K.-V. Yuen, L. Mihaylova, and H. Leung, “Overview of environment perception for intelligent vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 10, pp. 2584–2601, 2017. DOI: [10.1109/TITS.2017.2658662](https://doi.org/10.1109/TITS.2017.2658662).
- [2] P. Lindemann, T.-Y. Lee, and G. Rigoll, “Catch my drift: Elevating situation awareness for highly automated driving with an explanatory windshield display user interface,” *Multimodal Technologies and Interaction*, vol. 2, no. 4, 2018, ISSN: 2414-4088. DOI: [10.3390/mti2040071](https://doi.org/10.3390/mti2040071). [Online]. Available: <https://www.mdpi.com/2414-4088/2/4/71>.
- [3] M. Plappert, R. Houthoof, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, “Parameter space noise for exploration,” Jun. 2017. [Online]. Available: <http://arxiv.org/abs/1706.01905>.
- [4] Z. Ahmed, N. L. Roux, M. Norouzi, and D. Schuurmans, “Understanding the impact of entropy on policy optimization,” Nov. 2018. [Online]. Available: <http://arxiv.org/abs/1811.11214>.
- [5] M. Li, N. Mao, X. Zheng, and T. R. Gadekallu, “Computation offloading in edge computing based on deep reinforcement learning,” in *Proceedings of International Conference on Computing and Communication Networks: ICCCN 2021*, Springer, 2022, pp. 339–353.
- [6] L. Wells and T. Bednarz, “Explainable ai and reinforcement learning—a systematic review of current approaches and trends,” *Frontiers in artificial intelligence*, vol. 4, p. 550 030, 2021.
- [7] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
- [8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” Sep. 2015. [Online]. Available: <http://arxiv.org/abs/1509.02971>.
- [9] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, “Reinforcement learning with deep energy-based policies,” Feb. 2017. [Online]. Available: <http://arxiv.org/abs/1702.08165>.
- [10] Y. Li, Y. Duan, A.-B. Spulber, H. Che, Z. Maamar, Z. Li, C. Yang, *et al. Et al.*, “Physical artificial intelligence: The concept expansion of next-generation artificial intelligence,” *arXiv preprint arXiv:2105.06564*, 2021.
- [11] G. Cugola and A. Margara, “The Complex Event Processing Paradigm,” in *Data Management in Pervasive Systems*, ser. Data-Centric Systems and Applications, F. Colace, M. De Santo, V. Moscato, A. Picariello, F. A. Schreiber, and L. Tanca, Eds., Cham: Springer International Publishing, 2015, pp. 113–133, ISBN: 978-3-319-20062-0. DOI: [10.1007/978-3-319-20062-0_6](https://doi.org/10.1007/978-3-319-20062-0_6).

- [12] A. Kendall and Y. Gal, “What uncertainties do we need in bayesian deep learning for computer vision?” *Advances in neural information processing systems*, vol. 30, 2017.
- [13] A. Leite, M. Candadai, and E. J. Izquierdo, “Reinforcement learning beyond the bellman equation: Exploring critic objectives using evolution,” *Artificial Life Conference Proceedings*, no. 32, pp. 441–449, 2020. DOI: [10.1162/isal_a_00338](https://doi.org/10.1162/isal_a_00338).
- [14] A. K. Dey, “Understanding and using context,” *Personal and Ubiquitous Computing*, 2001.
- [15] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan, “Context is key,” *Communications of the ACM*, 2005.
- [16] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, “End-to-End Deep Reinforcement Learning for Lane Keeping Assist,” no. Nips, pp. 1–9, 2016. arXiv: [1612.04340](https://arxiv.org/abs/1612.04340). [Online]. Available: <http://arxiv.org/abs/1612.04340>.
- [17] H. Kamalzadeh and M. Hahsler, “POMDP: Introduction to Partially Observable Markov Decision Processes,” pp. 1–10, 2019.
- [18] A. R. Cassandra, “A Survey of POMDP Applications,” *Uncertainty in Artificial Intelligence*, pp. 472–480, 1997.
- [19] K. S. Luck, M. Vecerik, S. Stepputtis, H. B. Amor, and J. Scholz, “Improved exploration through latent trajectory optimization in deep deterministic policy gradient,” Nov. 2019. [Online]. Available: <http://arxiv.org/abs/1911.06833>.
- [20] N. Bougie1 and R. Ichise, *Fast and slow curiosity for high-level exploration in reinforcement learning*, 2021. arXiv: [1511.06581 \[cs.LG\]](https://arxiv.org/abs/1511.06581). [Online]. Available: <https://doi.org/10.1007/s10489-020-01849-3>.
- [21] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine, “Skew-fit: State-covering self-supervised reinforcement learning,” Mar. 2019. [Online]. Available: <http://arxiv.org/abs/1903.03698>.
- [22] C. E. Rasmussen, *Gaussian Processes in Machine Learning*, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–71, ISBN: 978-3-540-28650-9. DOI: [10.1007/978-3-540-28650-9_4](https://doi.org/10.1007/978-3-540-28650-9_4). [Online]. Available: https://doi.org/10.1007/978-3-540-28650-9_4.
- [23] G. E. Uhlenbeck and L. S. Ornstein, “On the theory of the brownian motion,” *Phys. Rev.*, vol. 36, pp. 823–841, 5 Sep. 1930. DOI: [10.1103/PhysRev.36.823](https://doi.org/10.1103/PhysRev.36.823). [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRev.36.823>.
- [24] K. Bartoszek, S. Glémin, I. Kaj, and M. Lascoux, “Using the ornstein–uhlenbeck process to model the evolution of interacting populations,” *Journal of Theoretical Biology*, vol. 429, pp. 35–45, Sep. 2017, ISSN: 10958541. DOI: [10.1016/j.jtbi.2017.06.011](https://doi.org/10.1016/j.jtbi.2017.06.011).

- [25] J. Nauta, Y. Khaluf, and P. Simoens, “Using the ornstein-uhlenbeck process for random exploration,” 2019, pp. 59–66.
- [26] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, *Noisy networks for exploration*, 2019. arXiv: [1706.10295](https://arxiv.org/abs/1706.10295) [cs.LG].
- [27] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, “Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 38–67, 2020. DOI: [10.1109/COMST.2019.2943405](https://doi.org/10.1109/COMST.2019.2943405).
- [28] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, “Cloud, fog, or edge: Where to compute?” *IEEE Internet Computing*, vol. 25, no. 4, pp. 30–36, 2021. DOI: [10.1109/MIC.2021.3050613](https://doi.org/10.1109/MIC.2021.3050613).
- [29] C. Sonmez, A. Ozgovde, and C. Ersoy, “Fuzzy workload orchestration for edge computing,” *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 769–782, 2019. DOI: [10.1109/TNSM.2019.2901346](https://doi.org/10.1109/TNSM.2019.2901346).
- [30] C. Anagnostopoulos, T. Aladwani, I. Alghamdi, and K. Kolomvatsos, “Data-driven analytics task management reasoning mechanism in edge computing,” *Smart Cities*, vol. 5, no. 2, pp. 562–582, 2022, ISSN: 2624-6511. DOI: [10.3390/smartcities5020030](https://doi.org/10.3390/smartcities5020030). [Online]. Available: <https://www.mdpi.com/2624-6511/5/2/30>.
- [31] V. Nguyen, T. T. Khanh, T. Z. Oo, N. H. Tran, E.-N. Huh, and C. S. Hong, “Latency minimization in a fuzzy-based mobile edge orchestrator for iot applications,” *IEEE Communications Letters*, vol. 25, no. 1, pp. 84–88, 2021. DOI: [10.1109/LCOMM.2020.3024957](https://doi.org/10.1109/LCOMM.2020.3024957).
- [32] M. D. Hossain, T. Sultana, M. A. Hossain, M. I. Hossain, L. N. T. Huynh, J. Park, and E.-N. Huh, “Fuzzy decision-based efficient task offloading management scheme in multi-tier mec-enabled networks,” *Sensors*, vol. 21, no. 4, 2021, ISSN: 1424-8220. DOI: [10.3390/s21041484](https://doi.org/10.3390/s21041484). [Online]. Available: <https://www.mdpi.com/1424-8220/21/4/1484>.
- [33] G. Gürsel, “Healthcare, uncertainty, and fuzzy logic,” *Digital Medicine*, vol. 2, pp. 101–112, 2016.
- [34] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, “Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework,” *IEEE Communications Magazine*, vol. 57, no. 3, pp. 56–62, 2019. DOI: [10.1109/MCOM.2019.1800608](https://doi.org/10.1109/MCOM.2019.1800608).
- [35] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, “Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading,” *IEEE Wireless Communications*, vol. 27, no. 1, pp. 92–99, 2020. DOI: [10.1109/MWC.001.1900232](https://doi.org/10.1109/MWC.001.1900232).

- [36] N. A. Abu-Taleb, F. Hasan Abdulrazzak, A. T. Zahary, and A. M. Al-Mqdashi, “Offloading decision making in mobile edge computing: A survey,” in *2022 2nd International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, 2022, pp. 1–8. DOI: [10.1109/eSmarTA56775.2022.9935407](https://doi.org/10.1109/eSmarTA56775.2022.9935407).
- [37] C. Sonmez, C. Tunca, A. Ozgovde, and C. Ersoy, “Machine learning-based workload orchestrator for vehicular edge computing,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2239–2251, 2021. DOI: [10.1109/TITS.2020.3024233](https://doi.org/10.1109/TITS.2020.3024233).
- [38] C. Quadri, V. Mancuso, M. A. Marsan, and G. P. Rossi, “Edge-based platoon control,” *Computer Communications*, vol. 181, pp. 17–31, 2022, ISSN: 0140-3664. DOI: <https://doi.org/10.1016/j.comcom.2021.09.021>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0140366421003583>.
- [39] L. Alekszejenkó and T. Dobrowiecki, “Simulating urban traffic as a multilayered multiagent system,” in *27th PhD Minisymposium of the Department of Measurement and Information Systems*, Budapest University of Technology and Economics, 2020, pp. 8–11.
- [40] I. Srisomboon and S. Lee, “A round-robin position change algorithm in vehicle platooning,” in *2022 International Conference on Information Networking (ICOIN)*, IEEE, 2022, pp. 340–344.
- [41] H. Kawashima, H. Kitagawa, and X. Li, “Complex event processing over uncertain data streams,” in *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2010, pp. 521–526. DOI: [10.1109/3PGCIC.2010.89](https://doi.org/10.1109/3PGCIC.2010.89).
- [42] S. A. Bell, “A beginner’s guide to uncertainty of measurement.,” 2001.
- [43] M. Désenfant and M. Priel, “Road map for measurement uncertainty evaluation,” *Measurement*, vol. 39, no. 9, pp. 841–848, 2006.
- [44] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin, “Complex event processing over uncertain data,” in *Proceedings of the Second International Conference on Distributed Event-Based Systems*, ser. DEBS ’08, Rome, Italy: Association for Computing Machinery, 2008, pp. 253–264, ISBN: 9781605580906. DOI: [10.1145/1385989.1386022](https://doi.org/10.1145/1385989.1386022).
- [45] A. Bühler, A. Gaidon, A. Cramariuc, R. Ambrus, G. Rosman, and W. Burgard, *Driving through ghosts: Behavioral cloning with false positives*, 2020. DOI: [10.48550/ARXIV.2008.12969](https://doi.org/10.48550/ARXIV.2008.12969). [Online]. Available: <https://arxiv.org/abs/2008.12969>.
- [46] I. Sobol’, “Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates,” *Mathematics and Computers in Simulation*, vol. 55, no. 1, pp. 271–280, 2001, The Second IMACS Seminar on Monte Carlo Methods, ISSN: 0378-4754. DOI: [https://doi.org/10.1016/S0378-4754\(00\)00270-6](https://doi.org/10.1016/S0378-4754(00)00270-6). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0378475400002706>.

- [47] S. Tennøe, G. Halnes, and G. T. Einevoll, “Uncertainpy: A python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience,” *Frontiers in Neuroinformatics*, vol. 12, p. 49, 2018, ISSN: 1662-5196. DOI: [10.3389/fninf.2018.00049](https://doi.org/10.3389/fninf.2018.00049). [Online]. Available: <https://www.frontiersin.org/article/10.3389/fninf.2018.00049>.
- [48] A. Saltelli, “Making best use of model evaluations to compute sensitivity indices,” *Computer Physics Communications*, vol. 145, no. 2, pp. 280–297, 2002, ISSN: 0010-4655. DOI: [https://doi.org/10.1016/S0010-4655\(02\)00280-1](https://doi.org/10.1016/S0010-4655(02)00280-1). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0010465502002801>.
- [49] A. Saltelli, *Global sensitivity analysis: the primer*. 2008, ISBN: 9780470059975. [Online]. Available: <http://books.google.at/books?id=wAssmt2vumgC>.
- [50] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, “Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index,” *Computer Physics Communications*, vol. 181, no. 2, pp. 259–270, 2010, ISSN: 0010-4655. DOI: <https://doi.org/10.1016/j.cpc.2009.09.018>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0010465509003087>.
- [51] P. P. Angelov, E. A. Soares, R. Jiang, N. I. Arnold, and P. M. Atkinson, “Explainable artificial intelligence: An analytical review,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 11, no. 5, e1424, 2021.
- [52] A. Das and P. Rad, “Opportunities and challenges in explainable artificial intelligence (xai): A survey,” *arXiv preprint arXiv:2006.11371*, 2020.
- [53] F. K. Došilović, M. Brčić, and N. Hlupić, “Explainable artificial intelligence: A survey,” in *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, IEEE, 2018, pp. 0210–0215.
- [54] P. Gerber, L. Jöckel, and M. Kläs, *A study on mitigating hard boundaries of decision-tree-based uncertainty estimates for ai models*, 2022. DOI: [10.48550/ARXIV.2201.03263](https://doi.org/10.48550/ARXIV.2201.03263). [Online]. Available: <https://arxiv.org/abs/2201.03263>.
- [55] S. V. Deshmukh, A. Dasgupta, B. Krishnamurthy, N. Jiang, C. Agarwal, G. Theocharous, and J. Subramanian, “Explaining RL decisions with trajectories,” in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: <https://openreview.net/forum?id=5Egggz1q575>.
- [56] M. Finkelstein, L. Liu, Y. Kolombus, D. C. Parkes, J. S. Rosenschein, S. Keren, *et al.*, “Explainable reinforcement learning via model transforms,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 34 039–34 051, 2022.

- [57] P. Gupta, N. Puri, S. Verma, D. Kayastha, S. Deshmukh, B. Krishnamurthy, and S. Singh, “Explain your move: Understanding agent actions using specific and relevant feature attribution,” in *International Conference on Learning Representations (ICLR)*, 2020.
- [58] J. Chen, S. E. Li, and M. Tomizuka, “Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 5068–5078, 2021.
- [59] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2012, pp. 5026–5033.
- [60] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, “Fog computing: Survey of trends, architectures, requirements, and research directions,” *IEEE Access*, vol. 6, pp. 47 980–48 009, 2018. DOI: [10.1109/ACCESS.2018.2866491](https://doi.org/10.1109/ACCESS.2018.2866491).
- [61] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC ’12, Helsinki, Finland: Association for Computing Machinery, 2012, pp. 13–16, ISBN: 9781450315197. DOI: [10.1145/2342509.2342513](https://doi.org/10.1145/2342509.2342513). [Online]. Available: <https://doi.org/10.1145/2342509.2342513>.
- [62] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog computing: A platform for internet of things and analytics,” in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Cham: Springer International Publishing, 2014, pp. 169–186, ISBN: 978-3-319-05029-4. DOI: [10.1007/978-3-319-05029-4_7](https://doi.org/10.1007/978-3-319-05029-4_7). [Online]. Available: https://doi.org/10.1007/978-3-319-05029-4_7.
- [63] Y. Jararweh, A. Doulat, A. Darabseh, M. Alsmirat, M. Al-Ayyoub, and E. Benkhelifa, “Sdmec: Software defined system for mobile edge computing,” in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, 2016, pp. 88–93. DOI: [10.1109/IC2EW.2016.45](https://doi.org/10.1109/IC2EW.2016.45).
- [64] P. Liu, J. Li, and Z. Sun, “Matching-based task offloading for vehicular edge computing,” *IEEE Access*, vol. 7, pp. 27 628–27 640, 2019. DOI: [10.1109/ACCESS.2019.2896000](https://doi.org/10.1109/ACCESS.2019.2896000).
- [65] A. Amini, W. Schwarting, A. Soleimany, and D. Rus, “Deep evidential regression,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 927–14 937, 2020.
- [66] Z. Ding, T. Yu, Y. Huang, H. Zhang, L. Mai, and H. Dong, “Rlzoo: A comprehensive and adaptive reinforcement learning library,” *arXiv preprint arXiv:2009.08644*, 2020.
- [67] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*, 2016. arXiv: [1602.01783](https://arxiv.org/abs/1602.01783) [cs.LG].

- [68] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: [1707.06347](https://arxiv.org/abs/1707.06347) [cs.LG].
- [69] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: [1801.01290](https://arxiv.org/abs/1801.01290) [cs.LG].
- [70] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016. eprint: [arXiv:1606.01540](https://arxiv.org/abs/1606.01540).
- [71] C. Sonmez, A. Ozgovde, and C. Ersoy, “Edgecloudsim: An environment for performance evaluation of edge computing systems,” in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 2017, pp. 39–44. DOI: [10.1109/FMEC.2017.7946405](https://doi.org/10.1109/FMEC.2017.7946405).
- [72] H. Yang and J. Moody, “Feature selection based on joint mutual information,” in *Proceedings of international ICSC symposium on advances in intelligent data analysis*, Citeseer, vol. 23, 1999.
- [73] M. Z. I. Chowdhury and T. C. Turin, “Variable selection strategies and its importance in clinical prediction modelling,” *Family medicine and community health*, vol. 8, no. 1, 2020.
- [74] **Maria J. P. Peixoto** and A. Azim, “Design and development of a machine learning-based task orchestrator for intelligent systems on edge networks,” *IEEE Access*, vol. 11, pp. 33 049–33 060, 2023. DOI: [10.1109/ACCESS.2023.3263483](https://doi.org/10.1109/ACCESS.2023.3263483).
- [75] H. Ouarnoughi, E. Grislin-Le Strugeon, and S. Niar, “Simulating multi-agent-based computation offloading for autonomous cars,” *Cluster Computing*, pp. 1–12, 2021.
- [76] N. Matloff, “Introduction to discrete-event simulation and the simpy language,” *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, vol. 2, no. 2009, pp. 1–33, 2008.
- [77] T. H. I. for Geoinformation Technology, *Openrouteservice*, Available at <https://openrouteservice.org/>.
- [78] E. Hüllermeier and W. Waegeman, “Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods,” *Machine Learning*, vol. 110, no. 3, pp. 457–506, Mar. 2021, ISSN: 1573-0565. DOI: [10.1007/s10994-021-05946-3](https://doi.org/10.1007/s10994-021-05946-3). [Online]. Available: <http://dx.doi.org/10.1007/s10994-021-05946-3>.
- [79] A. Kimmig, B. Demoen, L. De Raedt, V. S. Costa, and R. Rocha, “On the implementation of the probabilistic logic programming language problog,” *Theory and Practice of Logic Programming*, vol. 11, no. 2-3, pp. 235–262, Jan. 2011, ISSN: 1475-3081. DOI: [10.1017/s1471068410000566](https://doi.org/10.1017/s1471068410000566). [Online]. Available: <http://dx.doi.org/10.1017/S1471068410000566>.
- [80] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic Learning in a Random World*. Berlin, Heidelberg: Springer-Verlag, 2005, ISBN: 0387001522.

- [81] A. N. Angelopoulos and S. Bates, “A gentle introduction to conformal prediction and distribution-free uncertainty quantification,” *arXiv preprint arXiv:2107.07511*, 2021.
- [82] J. Alvarsson, S. Arvidsson McShane, U. Norinder, and O. Spjuth, “Predicting with confidence: Using conformal prediction in drug discovery,” *Journal of Pharmaceutical Sciences*, vol. 110, no. 1, pp. 42–49, 2021, ISSN: 0022-3549. DOI: <https://doi.org/10.1016/j.xphs.2020.09.055>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S002235492030589X>.
- [83] D. Boursinos and X. Koutsoukos, “Selective classification of sequential data using inductive conformal prediction,” in *2022 IEEE International Conference on Assured Autonomy (ICAA)*, 2022, pp. 46–55. DOI: [10.1109/ICAA52185.2022.00015](https://doi.org/10.1109/ICAA52185.2022.00015).
- [84] G. Cherubin, K. Chatzikokolakis, and M. Jaggi, “Exact optimization of conformal predictors via incremental and decremental learning,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, Jul. 2021, pp. 1836–1845. [Online]. Available: <https://proceedings.mlr.press/v139/cherubin21a.html>.
- [85] H. Linusson, U. Johansson, H. Boström, and T. Löfström, “Efficiency comparison of unstable transductive and inductive conformal classifiers,” in *Artificial Intelligence Applications and Innovations*, L. Iliadis, I. Maglogiannis, H. Papadopoulos, S. Sioutas, and C. Makris, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 261–270, ISBN: 978-3-662-44722-2.
- [86] T. Litman, *Evaluating accessibility for transport planning - measuring people’s ability to reach desired services and activities*, 2021.
- [87] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, “Microscopic traffic simulation using sumo,” in *The 21st IEEE International Conference on Intelligent Transportation Systems*, 2018. [Online]. Available: <https://elib.dlr.de/124092/>.
- [88] F. Arasteh, “Network-aware multi-agent reinforcement learning for adaptive navigation of vehicles in a dynamic road network,” M.S. thesis, York University, Toronto, Ontario, Canada, Sep. 2021.
- [89] C.-X. Zhang, J.-S. Zhang, and G.-W. Wang, “An empirical study of using rotation forest to improve regressors,” *Applied Mathematics and Computation*, vol. 195, no. 2, pp. 618–629, 2008, ISSN: 0096-3003. DOI: <https://doi.org/10.1016/j.amc.2007.05.010>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0096300307005917>.

- [90] Y. Liao and V. Vemuri, “Use of k-nearest neighbor classifier for intrusion detection,” *Computers & Security*, vol. 21, no. 5, pp. 439–448, 2002, ISSN: 0167-4048. DOI: [https://doi.org/10.1016/S0167-4048\(02\)00514-X](https://doi.org/10.1016/S0167-4048(02)00514-X). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S016740480200514X>.
- [91] E. D. C. Bezerra, A. S. Teles, L. R. Coutinho, and F. J. da Silva e Silva, “Dempster–shafer theory for modeling and treating uncertainty in iot applications based on complex event processing,” *Sensors*, vol. 21, no. 5, 2021, ISSN: 1424-8220. DOI: [10.3390/s21051863](https://doi.org/10.3390/s21051863). [Online]. Available: <https://www.mdpi.com/1424-8220/21/5/1863>.
- [92] U. Umoh, E. Udo, and E. Nyoho, “Support vector machine-based fire outbreak detection system,” *International Journal on Soft Computing, Artificial Intelligence and Applications*, vol. 08, pp. 01–18, May 2019. DOI: [10.5121/ijscai.2019.8201](https://doi.org/10.5121/ijscai.2019.8201).
- [93] R. Stekolshchik, “Noise, overestimation and exploration in deep reinforcement learning,” Jun. 2020. [Online]. Available: <http://arxiv.org/abs/2006.14167>.
- [94] W. Lim, S. Lee, M. Sunwoo, and K. Jo, “Hierarchical Trajectory Planning of an Autonomous Car Based on the Integration of a Sampling and an Optimization Method,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 613–626, 2018, ISSN: 15249050. DOI: [10.1109/TITS.2017.2756099](https://doi.org/10.1109/TITS.2017.2756099).
- [95] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.
- [96] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, *Mastering atari with discrete world models*, 2021. arXiv: [2010.02193](https://arxiv.org/abs/2010.02193) [cs.LG].
- [97] J. Ferret, O. Pietquin, and M. Geist, “Self-imitation advantage learning,” 2021. [Online]. Available: www.ifaamas.org.
- [98] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*, 2016. arXiv: [1602.01783](https://arxiv.org/abs/1602.01783) [cs.LG].
- [99] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, *Unifying count-based exploration and intrinsic motivation*, 2016. arXiv: [1606.01868](https://arxiv.org/abs/1606.01868) [cs.AI].
- [100] C. Stanton and J. Clune, *Deep curiosity search: Intra-life exploration can improve performance on challenging deep reinforcement learning problems*, 2018. arXiv: [1806.00553](https://arxiv.org/abs/1806.00553) [cs.AI].
- [101] R. A. Nafea and M. Amin Almaiah, “Cyber security threats in cloud: Literature review,” in *2021 International Conference on Information Technology (ICIT)*, 2021, pp. 779–786. DOI: [10.1109/ICIT52682.2021.9491638](https://doi.org/10.1109/ICIT52682.2021.9491638).

- [102] W. Ahmad, A. Rasool, A. R. Javed, T. Baker, and Z. Jalil, “Cyber security in iot-based cloud computing: A comprehensive survey,” *Electronics*, vol. 11, no. 1, 2022, ISSN: 2079-9292. DOI: [10.3390/electronics11010016](https://doi.org/10.3390/electronics11010016). [Online]. Available: <https://www.mdpi.com/2079-9292/11/1/16>.
- [103] P. Dini and S. Saponara, “Analysis, design, and comparison of machine-learning techniques for networking intrusion detection,” *Designs*, vol. 5, no. 1, 2021, ISSN: 2411-9660. DOI: [10.3390/designs5010009](https://doi.org/10.3390/designs5010009). [Online]. Available: <https://www.mdpi.com/2411-9660/5/1/9>.
- [104] P. Dini, A. Begni, S. Ciavarella, E. De Paoli, G. Fiorelli, C. Silvestro, and S. Saponara, “Design and testing novel one-class classifier based on polynomial interpolation with application to networking security,” *IEEE Access*, vol. 10, pp. 67 910–67 924, 2022. DOI: [10.1109/ACCESS.2022.3186026](https://doi.org/10.1109/ACCESS.2022.3186026).
- [105] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.
- [106] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.
- [107] **Peixoto, Maria JP** and A. Azim, “Improving environmental awareness for autonomous vehicles,” *Applied Intelligence*, vol. 53, no. 2, pp. 1842–1854, 2022. DOI: <https://doi.org/10.1007/s10489-022-03468-6>.
- [108] P. Sequeira and M. Gervasio, “Interestingness elements for explainable reinforcement learning: Understanding agents’ capabilities and limitations,” *Artificial Intelligence*, vol. 288, p. 103 367, 2020.
- [109] Y. Qing, S. Liu, J. Song, and M. Song, “A survey on explainable reinforcement learning: Concepts, algorithms, challenges,” *arXiv preprint arXiv:2211.06665*, 2022.
- [110] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, “Explainable ai: A review of machine learning interpretability methods,” *Entropy*, vol. 23, no. 1, p. 18, 2020.
- [111] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.
- [112] M. Désenfant and M. Priel, “Road map for measurement uncertainty evaluation,” *Measurement*, vol. 39, no. 9, pp. 841–848, 2006.
- [113] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “Carla: An open urban driving simulator,” in *Conference on robot learning*, PMLR, 2017, pp. 1–16.
- [114] R. Iyer, Y. Li, H. Li, M. Lewis, R. Sundar, and K. Sycara, “Transparency and explanation in deep reinforcement learning neural networks,” in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, 2018, pp. 144–150.

- [115] A. Lazaridis, A. Fachantidis, and I. Vlahavas, “Deep reinforcement learning: A state-of-the-art walkthrough,” *J. Artif. Intell. Res.*, vol. 69, pp. 1421–1471, 2020.