Università
di Genova

Department of Electrical, Electronic and
Telecommunication Engineering and Naval Architecture
(DITEN)

PhD in Science and Technology for
Electronic and Telecommunication Engineering
(STIET)

The Facets of Edge AI in Automotive: Exploring Embedded Frameworks, Voice Assistants, and Deep Reinforcement Learning

Doctoral Thesis

Tutor

Prof. Riccardo Berta

Author

Luca Lazzaroni

Coordinator of the PhD Program

Prof. Maurizio Valle

2024

XXXVI Cycle

Gotta catch 'em all!

Acknowledgments

Desidero esprimere la mia più sincera gratitudine ai Proff. Riccardo Berta e Francesco Bellotti che mi hanno guidato in questi tre anni e insegnato ad apprezzare questo percorso.

Ringrazio poi tutti i componenti del laboratorio, (o meglio, gli abitanti). In ordine, per così dire, di apparizione, ringrazio Alessio *vodars* Capello, per tutti gli aiuti ricevuti e le innumerevoli partite su Showdown; Marianna *xmarrix* Cossu, per aver sempre portato il buon umore in laboratorio; Alessandro *pigo* Pighetti e Luca *forno* Forneris (ai più noti come *pornerisfighetti*), fedelissimi compagni di padel che sarà dura rimpiazzare durante il loro soggiorno londinese; Matteo *malefretto* Fresta, maestro di Make it Meme.

Un ringraziamento va poi alla mia famiglia, che mi ha sempre supportato (anche quando non torno il weekend) e motivato.

Ringrazio infine Valentina, che mi ha accompagnato in questo percorso (e non solo) rendendolo leggero e piacevole.

Abstract

This thesis presents a comprehensive exploration of the synergistic relationship between edge computing and AI, with a particular focus on the automotive sector. As technology rapidly evolves, edge computing emerges as a paradigm shift, especially significant in automotive applications. The shift from cloud-centric approaches to decentralized computation enhances real-time processing capabilities and reduces latency, enabling intelligent decision-making at the network's edge. Central to this investigation is Edgine, a versatile, non-vendor-locked framework tailored for heterogeneous IoT applications. This work not only evaluates Edgine's adaptability but also innovatively applies it in developing tools for performance assessment in the automotive industry. Indeed, Edgine proved useful even in most of the later developed tools, both as an evaluation and measurement tool. Another significant contribution of this research pertaining to edge AI vehicular technology is the development of embedded voice assistants optimized for vehicles. The thesis details the creation of an end-to-end voice assistant system capable of operating offline, emphasizing privacy and security concerns inherent in cloud-based systems. The system supports the Italian language and, in a vehicular-only context, can achieve results comparable to cloud-connected solutions, demonstrating the feasibility of integrating advanced AI methods in embedded systems and the application of

transfer learning. The potential of edge computing in overcoming the limitations of traditional cloud-connected solutions is also examined, alongside future research directions for enhancing voice assistants in terms of latency, language, and domain support. The thesis then shifts focus to deep reinforcement learning (DRL), specifically its application to automated driving for low-speed maneuvering. The effectiveness of DRL is explored through experiments in both Unity and CARLA -simulated environments. Key factors for successful DRL training, such as curriculum learning and simulation parameter tuning, are discussed. Results, in both environments, are promising, paving the way to possible future research directions in dynamic scenarios and real-world vehicle implementations. The final area of exploration is the explainability of DRL models, a critical aspect in domains like automated driving where safety is paramount. A novel approach for interpretability analysis is presented, combining episode timelines, frame-by-frame analysis, and aggregated statistical analysis. This investigation provides insights into the decision-making processes of DRL models and highlights future research opportunities in areas such as temporal correlations and more complex vehicular models. In summary, this thesis links advancements in edge computing, embedded voice assistants, DRL in automated driving, and DRL model explainability. This integration shapes a dynamic and evolving landscape, fostering a foundation for innovative developments within the automotive industry.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
AD	Automated Driving
ADF	Automated Driving Function
AI	Artificial Intelligence
AMP	Automatic Mixed Precision
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ASR	Automatic Speech Recognition
CER	Character Error Rate
CL	Curriculum Learning
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
D3QN	Dueling Double Deep Q-Network
DDQN	Double Deep Q-Network
DIET	Dual Intent Entity Transformer
DL	Deep Learning
DNN	Deep Neural Network
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning

DSP	Digital Signal Processor
EV	Ego Vehicle
GAN	Generative Adversarial Network
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
ICER	Intent Classification Error Rate
IDM	Intelligent Driver Model
IoT	Internet of Things
IRER	Interpretation Error Rate
kNN	k-Nearest Neighbors
KWS	Keyword Spotting
MAV	Max Attention Vehicle
MDP	Markov Decision Process
ML	Machine Learning
MSV	Max SHAP Vehicle
NLP	Natural Language Processing
NLU	Natural Language Understanding
NN	Neural Network
NPV	Non-Player Vehicle
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
REST	Representational State Transfer

RF	Reward Function
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SB3	Stable-Baselines3
SC	Speech Classification
SER	Slot Error Rate
SLU	Spoken Language Understanding
SPI	Serial Peripheral Interface
SSID	Service Set Identifier
TD3	Twin Delayed Deep Deterministic policy gradients
TOPS	Tera Operations Per Second
TPU	Tensor Processing Unit
TRPO	Trust Region Policy Optimization
TSV	Tab-Separated Values
TTS	Text to Speech
UI	User Interface
VA	Voice Assistant
VAD	Voice Activity Detection
WER	Word Error Rate
XAI	Explainable Artificial Intelligence

1

Introduction

In the rapidly changing world of technology, this thesis conducts an extensive investigation into the mutually beneficial correlation between the edge computing paradigm and AI, specifically in the automotive sector.

In the constantly evolving field of IoT, edge computing presents itself as a paradigm shift that reshapes standard models, especially in the field of automotive applications. Unlike the conventional cloud-centric method, edge computing distributes computation and data storage, placing computing resources adjacent to the data source (Figure 1). This architectural change reduces delays and improves real-time processing power, thus converting network peripheral devices into intelligent entities capable of self-governing data processing and decision-making abilities [1].

Edge computing's distributed nature opens the way for a new era of connectivity and responsiveness. The capability to perform on-

device data processing enhances decision-making speed, rendering it critical in situations where low latency is crucial.

The importance of edge computing in the automotive industry goes beyond increased operational efficiency. Instant decision-making on the vehicle or its surroundings enhances operational efficiency and ensures safety, making edge computing a crucial linchpin. This becomes especially important in applications like automated driving (AD), where split-second decisions have significant consequences.

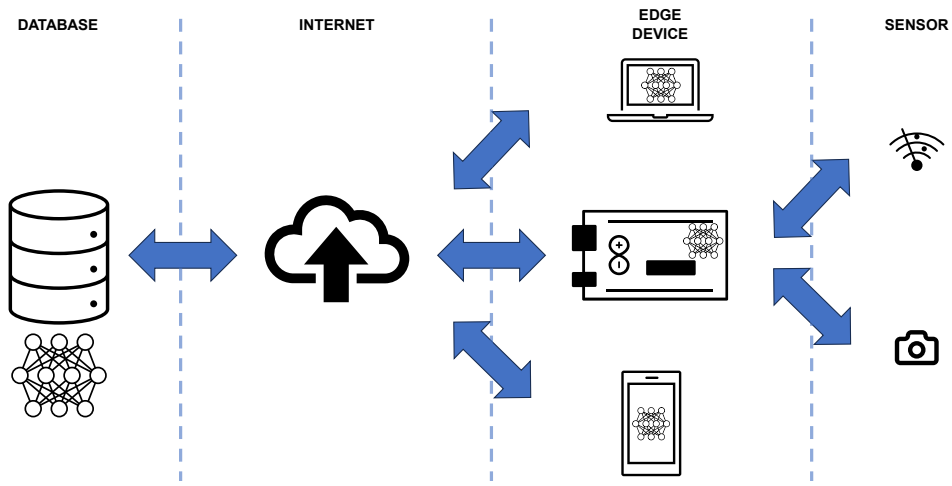


Figure 1: The edge computing paradigm schematic

At the heart of the transformative power of edge computing lies a system able to process and manage data at the network periphery. This should be positioned to perform computation and data processing tasks closer to the source of data generation, which, in the automotive context, often involves a myriad of sensors and devices embedded in the vehicle. By bringing computational capabilities

closer to the data source, these systems minimize latency and enhance the speed at which data is processed, enabling real-time decision-making.

Our exploration begins with a thorough analysis of edge computing systems and their crucial function in processing data from the edge. Central to the investigation is a versatile framework, Edgine, a non-vendor-locked, interoperable framework designed to support the IoT paradigm in diverse application domains [2], including the automotive industry. The first scope of our study involves evaluating the adaptability of Edgine in the creation of a range of applications across diverse fields such as business, environment, and sports. For our analysis, undergraduate students pursuing a Bachelor's degree in Electronic and Information Technology Engineering engaged in their final thesis have been selected. The primary objective of this target was to evaluate the ease of development, taking into consideration the subjects' basic professional proficiency. Additionally, it emphasizes the significance of didactics and didactic tools in cultivating a new cohort of electronic system designers who possess practical expertise in end-to-end IoT applications, rather than solely focusing on specific aspects.

By analyzing the architecture, real-world applications, and user feedback of Edgine, the flexibility and user-friendly design that distinguishes this comprehensive system are presented. In addition, given the versatile nature of the system, it will also prove useful for

the development and testing of the models and frameworks later discussed in this thesis.

Subsequently, another facet of edge computing applied to automotive is explored, such as embedded Voice Assistants (VAs). One significant advantage of VAs is their capacity to control diverse IoT devices through voice commands, including those integrated within intelligent vehicles. Privacy and security pose significant concerns, particularly in relation to cloud-based solutions [3], [4], [5]. Smart speakers are in a constant state of listening while they await the wake-word that triggers the dialogue management system of a VA. Subsequently, upon activation, a VA commences the process of capturing audio segments of the user's utterances. Consequently, an uninterrupted flow of clips is consistently sent to a remote server, where the audio data is analyzed to comprehend the user's spoken language and generate a suitable reply. This leads to the exposure of user data, thereby rendering the system susceptible to cyber-attacks. In addition, the processing of data necessitates a reliable and high-performing Internet connection. The inclusion of a real-time response is imperative to mitigate potential driver frustration and ensure safety, as certain situations may pose inherent risks. The aforementioned requirements, which hold significant importance in the automotive industry, indicate the significance of offline VAs in ensuring privacy, enhancing resilience, and ensuring uninterrupted functionality in all situations. Certain online VAs, such as Apple Siri and Google

Assistant, possess the additional functionality of offline voice assistance [6]. This includes features like setting alarms, sending text messages, and playing music. Nevertheless, the scope of supported functionalities is significantly constrained in comparison to the comprehensive online operability. In contrast, fully offline VAs are not as prevalent in usage and possess a more limited range of functionalities.

The utilization of the edge computing paradigm has therefore the potential to effectively address the aforementioned issues. Furthermore, the utilization of edge devices not only mitigates or minimizes the necessity for transmitting sensitive data but also offers advantages in terms of latency, energy efficiency, and bandwidth utilization [7], [8], [9]. Cloud-based solutions can leverage significantly larger computational and storage resources, thereby enabling the attainment of enhanced performance and versatility. However, the requirement for an embedded system to effectively manage and analyze data on-site presents a significant opportunity with the emergence of new generation edge devices. These devices possess substantial computational capabilities, including the inclusion of GPUs [10]. Furthermore, the utilization of cloud computing enables the training of Machine Learning (ML) and Deep Learning (DL) models on large datasets, which can subsequently be deployed on embedded devices with enhanced efficiency through optimized techniques [11]. This technology holds the potential to develop offline

VAs that can attain performance comparable to cloud-connected solutions, including the ability to process a wide range of commands.

Significant emphasis is placed on three primary novel contributions. Our primary objective is to attain offline performance that is on par with cloud-based systems, representing a noteworthy progression in the respective domain. Furthermore, our attention is directed towards providing support for non-mainstream languages such as Italian, acknowledging their significance in catering to a substantial portion of potential users on a global scale. In conclusion, we conduct a thorough examination of system modules, machine learning models, and training methodologies, contributing valuable insights and enhancing the existing knowledge in the field of offline virtual assistants. Given the embedded nature of the developed VA, for this use-case, Edgine is used to calculate and automatically log to the cloud all the evaluation metrics.

Afterward, the research moves from the study of the classic DL paradigm to investigate another crucial aspect of automotive AI, i.e., the use of Deep Reinforcement Learning (DRL) to build an autonomous agent able to drive in a vehicular context such as a low-speed maneuvering environment. An emerging phenomenon in the advancement of Automated Driving Functions (ADFs) involves, indeed, the utilization of DRL agents. These agents acquire task-specific knowledge by engaging with the designated environment through a trial-and-error approach, facilitated by a Deep Neural

Network (DNN) architecture [12]. Given the inherent characteristics of this methodology, it is advantageous to provide training within virtual driving simulators that closely replicate real-world conditions. This facilitates the establishment of secure operational conditions in which the agent can acquire the intended behavior, which is influenced by the allocation of rewards and penalties contingent upon the agent's position within the given environment. The primary goal of a DRL agent is to optimize the total expected rewards it receives throughout its entire lifespan. This is commonly referred to as the cumulative reward. Through the strategic utilization of acquired knowledge pertaining to the anticipated utility, which encompasses the summation of projected forthcoming rewards discounted by a specific factor, the agent possesses the capability to enhance its overall reward in the extended duration. This suggests that the components comprising the Reward Function (RF) must be intentionally crafted to facilitate the agent's attainment of the target policy. Experiments conducted in the Unity game simulator and CARLA reveal the effectiveness of DRL in real-time path planning and trajectory tracking, paving the way for real-world implementation. Again, Edgine is used for model testing and evaluation metrics gathering, showing the versatility and potential of the system.

Concluding with a critical examination of the explainability of DRL models, this study recognizes the significance of transparency in ML and DL, an especially crucial facet of an industry, such as the

automotive one, where safety standards take precedence. Our research aims to investigate the correlation between attention [13] and interpretability. SHAP is used for interpretability justified by its robust theoretical underpinnings in game theory ([14], [15]), as well as its extensive adoption in general contexts [16] and specifically within our field of study [17]. The SHAP method has indeed been recently introduced in the field of DRL as a foundational approach for elucidating the decision-making process of a trained agent in the specific context of the Gymnasium LongiControl environment [17]. The SHAP framework, which is grounded in game theory, provides a comprehensive approach for interpreting predictions post-hoc [14]. The proposed framework integrates six established methods, namely LIME [18], DeepLIFT [19], and Layer-Wise Relevance Propagation [20], which have been demonstrated to employ identical explanation models [16]. According to the provided prediction, SHAP assigns an importance measurement to each feature, enabling a quantitative explanation of the output generated by any machine learning model [14]. In the aforementioned study pertaining to the longitudinal control of a vehicle [17], the researchers computed SHAP values for various input features. These values were used to determine the impact of each feature on the chosen action, specifically the degree of acceleration. The findings are visually presented in a novel RL-SHAP diagram representation, organized along a timeline.

Furthermore, this research focuses on comprehending the methodologies for analyzing and visualizing attention and SHAP values within a 2D spatial highway setting. Specifically, we restrict our analysis to individual frames, disregarding any temporal correlations, thereby facilitating a clearer interpretation of the complex relations between SHAP and attention in isolation before advancing to more intricate (3D) temporal dynamics. Another question that is being considered is whether the abstract information derived from the neural model, such as attention and SHAP values, is adequate for achieving interpretability. Alternatively, it is being debated whether a more comprehensive domain-specific analysis should be conducted to ensure a thorough functional verification.

The simulation environment utilized in this study is highway-env [21]. It incorporates the Bicycle Kinematic Model motion model [22], which is a linear acceleration model inspired by the Intelligent Driver Model (IDM) [23]. Additionally, it incorporates a lane-changing behavior based on the MOBIL model [24]. Although this platform is less complex compared to widely-used vehicle simulators like CARLA [25] or SUMO [26], it has a strong track record in evaluating innovative decision-making control policies based on DRL (e.g., [27], [28], [29]). It is therefore highly suitable for creating a framework to analyze high-level decision-making in highway scenarios without any interference due to excessive environment complexity.

In summary, this study endeavors to explore and elucidate the synergistic potential of edge computing and AI within the automotive sector. While the diverse subjects addressed - Edgine, offline VAs, and DRL applications - may appear unrelated, they are intrinsically linked by the common thread of enhancing automotive intelligence through edge computing technologies.

Firstly, Edgine serves as the foundational framework that underpins this exploration. Its adaptable nature and relevance across various IoT applications, including automotive, provide a practical demonstration of how edge computing can reshape data processing and decision-making paradigms in vehicles.

Secondly, the development of offline VAs within vehicles showcases a specific application of edge computing, addressing critical concerns such as privacy, security, and real-time responsiveness. This area not only highlights the practicalities of implementing edge computing solutions but also delves into the challenges and opportunities of maintaining high performance in offline settings, a crucial aspect for automotive applications.

Finally, the application of DRL for the development of an ADF such as parking, demonstrates the role of edge computing in facilitating complex AI tasks. By training and deploying these models on edge devices, this research underscores the feasibility and benefits of localized, real-time data processing for critical vehicular functions.

This is also complemented by an in-depth analysis of Explainable Artificial Intelligence (XAI) applied to another key ADF such as highway driving.

Collectively, these topics do not simply represent discrete investigations; rather, they form a cohesive storyline that emphasizes the transformative impact of edge computing in the automotive sector. This thesis, therefore, aims to chart a path for future innovations in intelligent vehicular systems, demonstrating how these seemingly diverse elements blend to advance the field of automotive technology. This is essential to illustrate the multi-faceted influence of edge computing and AI in the automotive industry, paving the way for integrated, intelligent vehicular systems of the future.

2

Related Work

In this chapter, related works in the field of edge computing are presented, with a specific focus on its application in the automotive industry. Our examination begins by analyzing edge computing engines, which efficiently process vehicle data and enable the transmission of pertinent information to dedicate cloud infrastructures. Subsequently, the elaborate and crucial realm of on-the-edge voice assistants, which are highly relevant to the automotive sector, is explored. The focus is on the use of DL models to enhance the functionality of voice assistants and the resulting requirement for specialized datasets. The exploration further extends to the domain of motion and path planning, which is critical in the context of AD. The emphasis moves away from conventional DL models and toward the use of DRL to tackle the intricacies of motion and path planning for autonomous agents. The chapter concludes by shifting focus to the issue of explaining DRL models. Ensuring the explainability of DRL models becomes crucial since they play a pivotal role in autonomous

agent development, especially in industries where strict safety standards are a mandatory requirement.

2.1 Edge computing engines

Edge devices have gained increasing significance within the IoT framework [2], serving as integral components in a seamless computational flow from the field to the cloud. The edge computing paradigm [7], strategically situating computation, including AI, in close proximity to data sources [1], strives to minimize latency, energy consumption, and bandwidth usage.

Given its extensive application potential, major industry players are actively involved in crafting hardware and software solutions. Amazon provides the IoT solution Greengrass [30], streamlining local ML inference on devices using archetypes created, trained, and optimized in the cloud. AWS IoT Greengrass incorporates the Lambda runtime [31], a serverless computation service that eliminates the need for provisioning or managing infrastructure, automatically handling underlying compute resources with minimum hardware requirements of a 1 GHz processor frequency and 128 MB of RAM. On the other hand, Google introduced the Edge Tensor Processing Unit (TPU) [32] and Cloud IoT Edge [33]. The former, an Application-Specific Integrated Circuit (ASIC), is tailor-made for AI execution at the periphery, while the latter serves as an edge computing platform extending Google Cloud's data processing and ML capabilities to edge

devices. The strategy involves developing AI models in the cloud and utilizing them on IoT edge cloud devices, leveraging the capabilities of the Edge TPU hardware accelerator. This hardware can also run TensorFlow Lite [34], a platform simplifying the conversion of TensorFlow [35] Neural Network (NN) models into streamlined versions suitable for edge devices. A more compact TensorFlow Lite variant, TensorFlow Lite Micro, is specifically designed for running ML models on devices with limited memory such as Digital Signal Processors (DSPs) and microcontrollers [36]. Also, QKeras [37] allows to transform Keras models [38] to their quantized version, thus reducing inference time and resource consumption. IBM developed Edge Application Manager [39], an intelligent, secure, and flexible platform providing a management tool for edge processing. This autonomous solution allows a single administrator to handle the scale, variability, and frequency of application environment changes across endpoints concurrently. Edge endpoints operate on Red Hat OpenShift containers [40], supporting AI tools for DL, voice, and image recognition, as well as video and acoustics analysis. Microsoft contributes Azure IoT Edge [41], enabling the distribution of cloud workloads to run on IoT peripheral devices. Local processing reduces latency, with the option to utilize Microsoft's Project Brainwave [42], a DL platform for real-time AI inference in both the cloud and at the edge. Azure device management ensures functionality even in conditions of poor internet connection, automatically synchronizing device status upon reconnection. IoT Edge supports various

programming languages, including C, C#, Java, Node.js, and Python. Microsoft also launched EdgeML [43], a suite of ML algorithms for deployment in resource-constrained environments, with published results showcasing its effectiveness for training in conditions of limited computing power [44], [45], [46], [47], [48], [49].

In addressing cross-platform support, computational resource allocation algorithms were devised to enhance Vehicular Networks' performance [50], a crucial IoT application. The system uses the k-Nearest Neighbors (kNN) algorithm for platform selection (cloud computing, mobile edge computing, or local computing) and Reinforcement Learning (RL) for resource allocation, resulting in a significant 80% reduction in latency compared to basic algorithms. To tackle IoT device energy consumption [51], virtualization, particularly container-based virtualization, is suggested, addressing the multi-platform and multi-OS challenges [52]. A performance evaluation study explores the strengths and weaknesses of various low-power devices when handling container-virtualized instances versus native executions. Given its adaptability and close connection to IoT devices, edge computing presents diverse use cases. For example, in [53], a modular climatic enclosure through IoT device virtualization, enabling the application of common semantic rules for various users is proposed. Similarly, edge computing is employed in a smart IoT-based firefighting method [54], air pollution monitoring [55], edge video surveillance [56], and an IoT-based manufacturing context [57].

Table 1 summarizes the above-presented solutions and their main application fields.

Table 1: Summary of Edge Computing solutions

Provider	Solution Name	Key Features	Focus Area
Amazon	IoT Greengrass	Local ML inference, Lambda runtime for serverless computation, minimal hardware requirements	Cloud-integrated edge computing
Google	Edge TPU, Cloud IoT Edge	ASIC for AI execution, TensorFlow Lite for model optimization, extends cloud ML to edge devices	AI execution and model optimization at the edge
IBM	Edge Application Manager	Intelligent, secure management platform for edge processing, supports DL tools	Edge processing and management
Microsoft	Azure IoT Edge, EdgeML	Local processing to reduce latency, supports real-time AI inference, algorithms for resource-limited environments	Cloud workload distribution and AI inference
Research studies	Various	kNN for platform selection, RL for resource allocation, container-based virtualization for cross-platform support	Vehicular Networks, IoT smart applications

In this diverse landscape, the Edgine framework emerges as a pivotal innovation. Edgine is an open-source, platform-independent framework designed to streamline the development of edge computing applications across various domains. It distinguishes itself by its focus on abstraction and portability, aiming to facilitate code reuse and knowledge transfer across different application scenarios. Unlike vendor-specific solutions, Edgine's architecture is built to be cloud and edge-provider agnostic, enabling seamless integration in a wide range of IoT environments. This framework is not just a mere tool for application development; it represents a paradigm shift in how edge computing applications are conceptualized and executed. The detailed exposition of Edgine and its comprehensive capabilities will be further elaborated in Section 3.2.

2.2 On-the-edge voice assistants

This section delves into the landscape of speech processing, with a focus on the Italian language, designated as the target language for our system; and on English, which serves as the benchmark due to its extensive representation in scientific literature and availability in commercial products. All the presented solutions are then summarized in Table 2.

2.2.1 Datasets

The exploration into training the VA involved delving into the domain of open-source AI resources. This choice was motivated by the costs associated with proprietary datasets, often reaching tens of thousands of dollars. Italian open-source datasets, unfortunately, are somewhat limited, offering only a modest amount of speech hours. An emerging approach comes in the form of Mozilla Common Voice, a crowd-sourced, open-source, multi-language dataset [58]. Contributors participate by recording their voices through sentences provided on the Common Voice website [59]. The validation process employs a voting system, with each record subjected to assessment by three users. The Italian version of this dataset presently stands at 310 validated hours in MP3 format, boasting a mono configuration and a sample rate of 48 kHz. Manifest files, structured in Tab-Separated Values (TSV) format, are available for training, validation, test phases, and additional categories. The M-AILABS dataset [60], a freely usable multi-language resource, taps into LibriVox and Project Gutenberg [61], [62]. It is segmented by speaker name and sex, making it particularly suitable for tasks requiring a single speaker, such as Speech Synthesis. The Italian subset features two voices - male and female - totaling 18 hours of speech. Recordings, in WAV format with a mono configuration, are set at a 16 kHz sample rate. Notably, the texts originate from literature classics published between 1884 and 1964, imbuing them with a sometimes courtly and emphatic pronunciation. Manifest files are available in CSV and JSON formats.

Facebook AI contributes to the dataset repertoire with Multilingual LibriSpeech (MLS), an open-source collection spanning eight languages, including Italian [63]. Drawing from LibriVox audiobooks, MLS, akin to M-AILABS, boasts a larger scale. The Italian segment, enriched with 28 different speakers, accumulates to 279 hours of speech.

2.2.2 Models and Toolkits

The construction of speech recognition models is inherently tied to datasets and facilitated through speech toolkits. Recent developments have ushered in Italian language-enabled toolkits, some of which offer pre-trained models suitable for fine-tuning in specific applications. Vosk [64] emerges as a distinct speech recognition toolkit characterized by a substantial vocabulary transcription, reconfigurable vocabulary, and user-friendly installation. Notably, it provides a compact Italian model compatible with Raspberry Pi and Android devices. Mozilla DeepSpeech [65] leans on the extensive Common Voice Italian dataset for Automatic Speech Recognition (ASR), utilizing end-to-end DL with a Recurrent Neural Network (RNN) core. Facebook AI Research contributes wav2letter [66], an ASR tool accompanied by pre-trained models, including an Italian variant derived from Multilingual LibriSpeech [63]. NeMo by NVIDIA [67] extends beyond ASR, supporting Voice Activity Detection (VAD), Keyword Spotting (KWS), and Text to Speech (TTS) in a comprehensive package. Comparative assessments position

NVIDIA's NeMo toolkit with a slight edge [68]. Additionally, a DL-based speech recognition system is presented in [69], featuring a semantic communication model extracting text-related semantic features through a Convolutional Neural Network (CNN) and RNN-based encoder. These semantic features are then converted to text information through a decoder, followed by speech synthesis to regenerate the speech signals. Wav2vec 2.0 [70] introduces a framework for self-supervised learning of speech representations, achieving commendable Word Error Rate (WER) on the Librispeech dataset (4.8/8.2 on test-clean/other) [71].

Given the modest dimensions of Italian datasets compared to their English counterparts, especially in multi-speaker scenarios, a viable strategy involves leveraging transfer learning [72], [73]. Mozilla DeepSpeech offers a transfer learning version fine-tuned from the Common Voice English dataset to various languages, including Italian. Applications of fine-tuning, as demonstrated in [74] with the NeMo toolkit, reveal promising outcomes. Indeed, the QuartzNet 15x5 model, fine-tuned with Common Voice Spanish and Russian datasets, outperforms its trained-from-scratch counterparts. Challenges, however, lie in the demanding computing resources essential for the training phase, a subject addressed in [75] along with results achieved by the QuartzNet model within the NeMo framework.

Following the speech recognition stage, Natural Language Understanding (NLU) takes center stage in extracting meaning from the transcribed text. Established solutions requiring an Internet connection, such as Google Dialogflow [76], Amazon Lex [77], Facebook wit.ai [78], and Microsoft Bot Framework [79], coexist with off-line, open-source alternatives. Rasa [80], a Python module, stands out for its capability in ML-based dialogue management and language understanding. It provides pre-defined pipelines, including the integration of spaCy [81], an open-source Natural Language Processing (NLP) tool boasting pre-trained models in 64 languages. This combination, supported by Rasa, facilitates offline NLU, with an available default pipeline for training models from scratch, catering to domain-specific applications. DeepPavlov [82], another open-source library in Python, specializes in developing dialogue agents. It features three models: intent classification, entity recognition, and spelling correction. While resembling Rasa in its pipeline-based approach, it exhibits less customizability. Recent developments showcase also end-to-end solutions for ASR and NLU. An English, transformer-based [13] model for Spoken Language Understanding (SLU) is proposed in [83], achieving impressive accuracy on the Fluent Speech Commands dataset [84]. Snips [85], a dedicated SLU platform for IoT microprocessors, stands out for its offline functionality with low resource consumption. However, its recent transition to private ownership has rendered the code non-public and no longer open-source. Paval [86] introduces a virtual personal assistant focusing on

suggesting local points of interest and services. In a domain-specific context, Paval outperforms general-purpose systems like Google Assistant, Apple Siri, and Microsoft Cortana, leveraging not only NLP but also semantic technologies and external knowledge for geo-located data retrieval.

The final stage of building a VA involves speech synthesis, commonly known as TTS. The section presents two fundamental types of solutions: a two-stage pipeline and an end-to-end approach. The two-stage pipeline involves generating a spectrogram (mel or Hz scale) initially, followed by a voice-encoder (vocoder) producing audio based on the spectrogram. Numerous models adopting this approach are available, including the renowned Tacotron2 [87], which maps characters to mel-scale spectrograms through a recurrent sequence-to-sequence feature prediction network. Tacotron2 is the most well-known spectrogram generator, and numerous other contributors have adopted its implementation [88], [89]. Notable vocoders include glow-based (WaveGlow [90], SqueezeWave [91], UniGlow [92]) and Generative Adversarial Network (GAN) architectures (MelGAN [93], HiFiGAN [94]). On the contrary, the end-to-end approaches utilize a single model to generate audio directly from the text. Several models exemplifying this approach are available in the literature. FastPitchHifiGAN [95] combines a spectrogram generator (FastPitch [96]) and the HiFiGAN vocoder for waveform generation from the text. Similarly, FastSpeech2HiFiGAN [97]

combines the FastSpeech2 [98] spectrogram generator and HiFiGAN into a unified model, trained end-to-end. All the TTS models discussed, including those employing the two-stage pipeline and end-to-end approaches, are accessible within the NeMo toolkit. Additional end-to-end TTS systems, such as NaturalSpeech [99] and FastDiff-TTS [100], are introduced. NaturalSpeech exploits a variational autoencoder for direct waveform generation, achieving human-level performance on the LJ Speech dataset [101]. FastDiff-TTS stands out for both high-quality speech synthesis and impressive inference speed, enabling real-time speech synthesis. Although the source code for these end-to-end TTS solutions is yet to be released officially, their potential in enhancing the VA experience is acknowledged.

Table 2: Summary of state-of-the-art speech-processing datasets, models, NLU solutions, and TTS solutions

Category	Name	Description	Specifics / Highlights
Datasets	Mozilla Common Voice	Crowdsourced, multi-language dataset	Italian version: 310 hours, 48 kHz, MP3
	M-AILABS	Multi-language dataset, using LibriVox and Project Gutenberg	Italian: 18 hours, 16 kHz, WAV format
	Multilingual LibriSpeech	Open-source collection, multi-language	Italian: 279 hours, diverse speakers

Models & Toolkits	Vosk	Speech recognition toolkit with reconfigurable vocabulary	Compact Italian model, Raspberry Pi/Android compatible
	Mozilla DeepSpeech	End-to-end DL with RNN for ASR	Utilizes Common Voice Italian dataset, appears to be no longer maintained
	wav2letter	ASR tool by Facebook AI with pre-trained models available in many languages	Italian variant available
	NeMo	NVIDIA's toolkit for ASR, VAD, KWS, and TTS	Comparative edge in performance
	Wav2vec 2.0	Self-supervised learning framework for speech representations	Low WER on Librispeech dataset
NLU Solutions	Rasa	ML-based dialogue management and language understanding in Python	Offline functionality, high versatility, supports spaCy
	DeepPavlov	Python library for dialogue agents	Features intent classification, entity recognition, and spelling correction

	Tacotron2	Two-stage pipeline model mapping characters to mel-scale spectrograms then to audio	Widely adopted spectrogram generator
	FastPitchHifiGAN	Combines FastPitch spectrogram generator with HiFiGAN vocoder	End-to-end model for text to waveform generation
TTS Solutions	FastSpeech2HiFiGAN	Combines FastSpeech2 with HiFiGAN into a single model	Trained end-to-end
	NaturalSpeech	Utilizes a variational autoencoder for direct waveform generation	Human-level performance on LJ Speech dataset
	FastDiff-TTS	Notable for high-quality speech synthesis and impressive inference speed	Enables real-time speech synthesis

2.3 DRL for motion and path planning

In literature, numerous techniques are proposed to tackle the complexities of the motion and path planning domain. Global planning, a pivotal aspect, witnesses the influence of graph-search-based methodologies. Algorithms like Dijkstra [102], [103], A* [104],

Hybrid A* [105], and State Lattice [106] play a crucial role by mapping the state space of vehicles and other objects in the scene onto an occupancy grid. Sampling-based suboptimal planners, including the renowned Probabilistic Roadmap Method (PRM) [107] and the Rapidly-exploring Random Tree (RRT) [108], take a different approach by randomly sampling the configuration or state space, seeking connectivity [109]. To overcome the sub-optimality of RTT, [110] hybridized it with Ant Colony Systems (ACS) algorithms, resulting in good performance and fast convergence. Curve interpolation techniques, exemplified by Bezier [111] and Splines [112], as well as those inspired by biological systems such as genetic algorithms (GA) [113], ant colony algorithms (ACO) [114], and particle swarm algorithms [115], have gained prominence due to their rapid convergence and robust characteristics.

For what concerns local path planning, the Artificial Potential Field (APF) algorithm, introduced by Khatib [116], stands as a pioneering solution. By combining continuous attractive fields related to goal-reaching tasks and repulsive fields generated by obstacles, the APF algorithm navigates a vehicle through unknown environments. Also, variations of the APF algorithm have emerged, including methods for local minimum avoidance by anticipating future movements to preemptively bypass obstacles [117] and the incorporation of techniques like Simulated Annealing [118] and Fuzzy Logic [119]. Several algorithms have been presented to handle

dynamic obstacles. One such example is the dynamic window approach with the virtual manipulators (DWV) technique [120]. The DWV method produces modified candidate paths that are non-straight and non-arc by predicting the location of dynamic obstacles.

Turning our attention to the DRL paradigm, it is first necessary to introduce RL. RL, as a subset of Machine Learning (ML), takes center stage in this context. RL involves an agent learning a policy through trial and error, discerning which actions to take in diverse states. The learning process unfolds in an environment providing positive and/or negative rewards for each decision taken by the agent. Framed as the optimal control of a Markov Decision Process, RL boasts two primary approaches: value-based algorithms, focused on finding the action with the maximum expected overall value, and policy-based algorithms, aimed at determining the maximum reward policy [121]. The optimal action-value function $Q^* = \max_{\pi} Q^{\pi}(s)$ satisfies the Bellman Optimality Equation: $Q^*(s,a) = \mathbb{E}\{\max_{a' \in A}[R(s,a) + \gamma Q^*(s',a')]\}$, where s is a state, a an action, the apex denotes the next step, R is the reward and γ the discount factor [122]. The Q-learning algorithm iteratively calculates [123] Q^* through the application of a sampling version on a batch of collected experience. To tackle the challenge of continuous state spaces, the Deep Q-Network (DQN) algorithm [124] uses a NN model to represent the action-value function Q. Additional algorithms, including Double Deep Q-Network (DDQN) [125], Dueling DQN [126], and Dueling Double

Deep Q-Network (D3QN) [127], have emerged as pivotal solutions, contributing significantly to overcoming challenges related to convergence and stability.

In dynamic path planning within unknown environments, we witness a significant paradigm shift. The application of the DQN algorithm [128], a cornerstone in DRL, allows for learning successful policies directly from high-dimensional sensory inputs through end-to-end RL. In [129], a DRL agent is trained in a dynamic unknown environment using the DDQN method [130] and a CNN as a backbone, allowing it to reach the targeted position. In [131], the asynchronous advantage actor-critic (A3C) method [132] is utilized for training a four-wheeled rescue robot on difficult terrain within an urban environment.

Combining global and local planning strategies emerges as a promising approach for achieving superior overall performance. PRM-RL [133], a hierarchical method that merges sampling-based path planning with RL, exemplifies the power of combining short-range, point-to-point navigation policies learned by RL agents with the guidance of PRM planning for long-range navigation. Similarly, the integration of RRT and a DRL agent (RL-RRT) presented in [134] for long-range planning demonstrates the fusion of planning and control functionalities. The agent serves as the local planner during planning and as the controller during execution. The system shows its capability of being transferable to previously unobserved

experimental environments, thus making RL-RRT faster by replacing expensive computations with simple neural network inference. In [135], a new path planner (PRM+TD3) that combines the Twin Delayed Deep Deterministic policy gradients (TD3) DRL algorithm with the RPM global path planning method is presented. The proposed method demonstrates promising results in terms of both development efficiency and model generalization.

The integration of DRL in various applications extends its reach to autonomous underwater vehicles (AUVs) and unmanned aerial vehicles (UAVs). Techniques like DRL path planning based on DDQN prove effective in enhancing the planning of autonomous underwater vehicles navigating through ocean currents [136]. Similarly, the application of DDQN for path planning in UAVs, particularly those leveraging edge servers for computing tasks [137], showcases the adaptability of DRL across diverse scenarios.

2.3.1 Driving simulators

Simulation tools have become integral in testing and refining complete DRL pipelines before deploying learned agents in real-world scenarios. This is even more important considering AD contexts, in which realism and availability of real cars' sensors is key to simulate ADFs. Highway-env [138], implemented in Python and relying on the Gymnasium toolkit [139], offers a collection of open-source environments for autonomous driving and tactical decision-making

tasks. The environments are presented in a bird's-eye view and serve as a robust platform for the development, deployment, testing, and comparison of RL methods and models. Stable-Baselines 3 [140], [141] is employed as the library, providing implementation support for state-of-the-art DRL algorithms. Another notable simulation tool is Unity, a widely adopted game engine, cross-platform in nature, that enables the streamlined development of games and simulations, applied across multiple industrial sectors [142]. It can be utilized with ML-Agents [143], an open-source toolkit that enables the usage of Unity as a simulation setting to fashion and educate self-governing agents. ML-Agents also offers a Python Application Programming Interface (API) for the implementation of main RL algorithms, which is based on the PyTorch library. Finally, CARLA [25] is an open-source simulator specifically designed for developing, testing, and evaluating AD systems within a realistic urban environment. CARLA stands out for its feature-rich framework, incorporating physically accurate vehicle models, highly detailed maps reflecting real-world road networks, accurate traffic flow, infrastructure information, weather conditions, and pedestrian simulation. The framework provides a comprehensive set of APIs and ancillary tools to facilitate the development and evaluation of autonomous systems.

These simulation tools serve as invaluable assets in bridging the gap between theoretical developments and practical implementations,

ensuring the robustness and efficacy of autonomous systems in diverse and dynamic environments.

A summary of the comparison between the simulators presented is shown in Table 3.

Table 3: Driving simulators summary

Simulator	Features	Strengths	Application Focus
Highway-env	Open-source environments for AD and tactical decision-making, bird's-eye view	Robust platform for RL methods and model testing	Development and testing of ADFs and tactical decision-making tasks
Unity 3D	Cross-platform game engine for simulation development, applied in multiple sectors	Versatile, widely adopted, allows for detailed customization	Development of games and simulations, including automotive applications
CARLA	Realistic urban environment simulator for AD systems	Detailed maps, realistic simulated sensors, comprehensive simulation of traffic and pedestrians	Developing, testing, and evaluating ADFs in a realistic urban setting

2.4 DRL models explainability

The AD field poses a unique set of challenges, that require sophisticated decision-making abilities in constantly changing and highly uncertain environments. In our pursuit to attain elevated stages of automation, it is crucial to establish effective strategies for high-level decision-making [144]. The task of driving is appropriately formalized as a Partially Observable Markov Decision Process (POMDP), while carefully taking into account the stochastic nature of various road actors and the inherent uncertainties associated with perception systems [145].

The key component lying at the heart of decision-making within behavioral planning is the prediction of trajectory. Traditional approaches to trajectory prediction mainly focus on spatial interaction modeling, with landmark models such as the Social Force model [146]. This model combines attractive forces from a goal vehicle with repulsive forces generated by other vehicles. Nevertheless, the trajectory prediction landscape is evolving, with attention-based architectures trained through DRL steadily increasing in popularity. Pioneering research, exemplified by [28], [147], and [148], highlights the advantages of such architectures in dealing with varying numbers of nearby vehicles, ensuring invariance to the chosen order of feature representation, and inherently accommodating interactions between the Ego Vehicle (EV) and other traffic participants [149]. Recent advancements, such as the Hierarchical Spatio-Temporal Attention

architecture (HSTA) [150], introduce innovative elements by incorporating spatial interactions with varying weights and considering temporal interactions across multiple time steps involving all agents. The work of [28], which integrates attention modules into a hierarchical control structure of a D3QN (D3QN-DA), stands out for achieving superior safety rates and average exploration distances. Concurrently, [27] presents an analogous hierarchical control framework. In this setup, the upper-level is dedicated to managing driving decisions in a highway environment, while the lower-level governs speed and acceleration. By integrating the D3QN DRL algorithm within this hierarchical framework the convergence rate and control performance of the highway decision-making strategy are significantly improved.

The field of XAI comes into play with [151], providing the foundational principles of interpretable data science for decision-making. To visualize feature interactions and importance, model-agnostic approaches like LIME [18], SHAP [14], partial dependence plots (PDP) [152], and permutation feature importance scores are widely applied and recognized in recent reviews [16], [153]. Acknowledging the challenges in producing highly-performing white-box models, particularly in domains like computer vision and natural language processing, [16] highlights the persistent gap in performance against DL models. Furthermore, [154] draws attention to the several open research points in XAI, citing challenges ranging

from defining explanations to general interpretation pitfalls. In this expansive field, [155] warns against typical errors in interpreting ML models, such as utilizing interpretation techniques inappropriately or drawing unwarranted causal conclusions. Within this broad landscape, SHAP emerges because of the ability to analyze individual predictions by quantifying the contribution of each feature to the overall prediction. Rooted in the concept of Shapley values [156] for model feature influence scoring, SHAP offers desirable properties such as consistency, missingness, and local accuracy, albeit with a high computational cost. It operates as an additive feature attribution method, meaning that the sum of SHAP values aligns with the model's prediction. This approach adheres to efficiency properties, guaranteeing a fair distribution of effects (i.e., the prediction) among the features.

3

Edgine

The idea of Edgine arose from the necessity for a flexible and adaptable framework that facilitates the IoT paradigm across diverse application domains, ranging from agriculture to sports to automotive. To achieve this goal, three essential components are necessary:

- An edge device that collects environmental data through sensors;
- An execution engine that operates on the edge, interprets and processes the data captured by the device, and subsequently sends it to the cloud in a structured manner;
- A cloud server that stores the data and can be queried for visualizing and extrapolating information.

This chapter primarily delves into the architecture of the cloud server (referred to as Measurify [157]), followed by the presentation of

the execution engine, Edgine. Then, this study demonstrates various real-world use cases of the system utilizing different edge devices, highlighting the user-friendly design that sets apart this end-to-end system, as well as its versatility. Finally, some users' feedback is reported, concluding presenting the ongoing work on new features for Edgine.

3.1 Measurify

Measurify is a cloud-based platform that has been developed to effectively manage intelligent objects inside IoT ecosystems. It is characterized by its abstract nature and its focus on measurement-oriented functionalities. Measurify employs a methodology wherein it represents these objects as online resources, making them accessible through APIs that adhere to the principles of Representational State Transfer (REST) architecture. The process involves the utilization of a platform-agnostic HTTPS interface to enable remote access to data and resources, facilitating the creation of apps that may effectively utilize these entities.

Table 4 presents a comprehensive overview of the primary resources employed by Measurify. Further information can be obtained from [157], where the Measurify framework is presented.

Table 4: Outlook of the main Measurify resources [9]

Element	Description
Measurement	Value of a feature measured by a device for a specific thing
Thing	A generic object target of a measurement (I.e., within which a measurement is performed)
Device	A sensor that provides measurements about a thing
Feature	A physical dimension measured by a device
Script	A JSON string that contains information on how to manipulate, store, and transmit streams of measurements coming from devices. This is the program to be executed by a field device
Tag	Labels attachable to resources, to better specify them
Users	Users (with roles and rights) that have access to the resources of the current application

In addition to gathering all the data transmitted from the field, Measurify also offers developers a remote programming interface for configuring a deployed field execution engine. By providing its credentials, each edge engine can access a specific tenant area within a Measurify cloud installation. The schematic representation of the edge-cloud system is presented in Figure 2. This picture illustrates the arrangement and progression of both the configuration and execution stages of a typical IoT ecosystem application, which is facilitated by the Measurify platform. During the configuration phase, the developer establishes the specifications for various elements such as things, devices, features, user roles, and permissions, as well as scripts that are relevant to the new application. During the execution phase, the devices gather, analyze, and transmit measurements to the cloud, which may then be accessed and queried by authorized users. The two phases are not temporally distinct, as a developer can include new elements such as features, scripts, and other components throughout the execution phase.

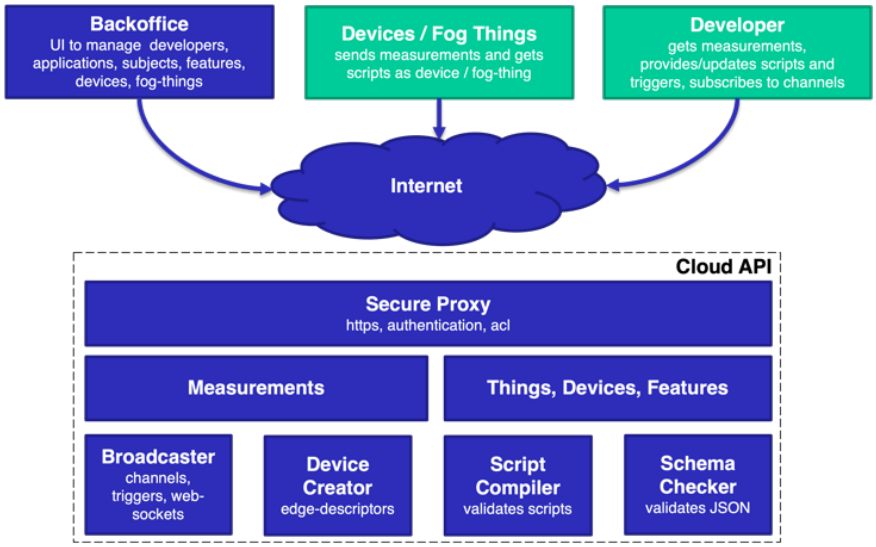


Figure 2: Measurify cloud API schematic [9]

3.2 Edgine

Edgine is a cross-platform edge system that offers the capability to retrieve scripts from cloud-based sources and execute operations locally (Figure 3). From the standpoint of edge-to-cloud continuum computing, the system possesses the capability of being configured remotely, encompassing both settings and executable scripts.

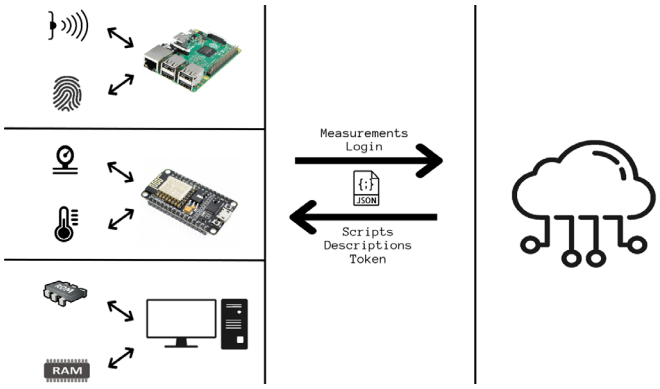


Figure 3: Block diagram of Edgine in the IoT ecosystem [158]

Edgine relies on Measurify for its cloud infrastructure. Measurify not only gathers all the data transmitted from the field by an Edgine instance, but it also offers developers a remote programming interface for configuring a deployed Edgine. The runtime operation of Edgine includes two distinct components: an initialization phase and a continuous loop phase. At the beginning of the start-up process, Edgine establishes a connection with the API to retrieve its description. This description includes a comprehensive list of scripts that are to be executed, as well as the corresponding parameter values required for its configuration. During the iteration, Edgine sequentially executes each assigned script. Every device that is linked to a Measurify installation is characterized by a JSON descriptive scheme that encompasses the features and scripts fields. The former refers to the enumeration of measurement kinds, as stated in Measurify, which serves to ensure the integrity of input. On the other hand, the latter serves as a descriptor for the computing work that is necessary, as depicted in Figure 4. Specifically, the *code* field of the system describes the executable script, which consists of a series of functions (instructions) that the Edgine use to process its raw data before transmitting it to the cloud. This processing occurs in a continuous loop. Every instruction is implemented on its input data stream, which consists of the output from the preceding instruction. The initial phase of the sequence is employed to process the raw input data. The instructions depicted in Figure 4 pertain to the calculation of

the available ROM in GB, its subsequent conversion into MB, and the ultimate dispatch to Measurify.

```
{
    "visibility": "private",
    "tags": [],
    "_id": "rom-available-to-mb",
    "code": "available-rom().map(a*1024).send()"
}
```

Figure 4: JSON description of a script [158]

There are two potential methods for uploading data to the cloud: continuous uploading, where data is transmitted as soon as it is processed, or batch uploading, where a specified number of measurements are accumulated before being sent in bulk. This second option is particularly useful in the absence of internet connection or if the embedded system on which Edgine is installed is required to consume as little energy as possible.

The existing operation set is documented in Table 5. Table 6 presents a synthesis of the HTTP requests made during the two distinct phases of execution, namely authentication and script download (performed at start-up), and the subsequent infinite loop for uploading measurements. It is worth to highlight that this process is entirely automatically performed by Edgine and does not require any user intervention.

Table 5: Instructions currently available in Edgine [9]

Instruction	**Description**
send	Sends to the API all elements of the data stream
map	A new data stream is created by performing a simple arithmetic operation between two operands
max/min	A new data stream is created containing only the min/max value among the values in the input stream
window / slidingWindow	A new data stream is created by applying a two-operand function on an accumulator, initialized to the value of the second argument, and on each input element, for a number of values indicated by the size of the window/slidingWindow
filter	A new data stream is created letting using only the elements of its input stream that have a value within a specified range
average / median / stdDeviation	A new data stream is created by taking the average/median/stdDeviation of a specified number of samples in its input stream

Table 6: Edgine's HTTPS requests [9]

Request	Subject	Description
POST	Login credentials	JWT is received from the cloud
GET	Device description	Scripts are retrieved from the cloud
POST	Measurements	Edge-processed data are shipped the cloud

Another significant advancement pertained to the communication interface, which was intentionally designed to be highly abstracted from the underlying hardware to enhance portability. To facilitate the transition of developers from Windows, Linux, or macOS platforms to Arduino, classes have been developed that incorporate macros. The distinction between the two types of platforms pertains to the nature of their Internet connectivity. The Arduino system facilitates automatic connection to a specified Wi-Fi network as indicated in the code and is additionally programmed to initiate a reconnection process in the event of signal disruption. In contrast, network connection (including reconnection) on PC-type devices is not automated, as users can utilize the User Interface (UI) to manually select a network of their choice. In addition, the automation of the

connection necessitates distinct implementations for various operating systems, thereby introducing complexity to the structure of Edgine. The transmission of data to the cloud is facilitated by a thread that is created during each cycle, ensuring that the edge device is not obstructed by communication tasks. Furthermore, regardless of the specific platform being used, a thread queue is utilized to enqueue threads when the network connection is unavailable. This ensures that data delivery remains accurate even under such circumstances.

By leveraging switchable network connection classes, the Edgine platform has been successfully implemented on major PC operating systems, namely Windows, Linux, and macOS. Additionally, Edgine has been extended to support various Arduino and Arduino-style boards, including Arduino MKR WiFi 1010, Arduino UNO WiFi Rev.2, Arduino NANO 33 IoT, Arduino NANO 33 BLE, Arduino MKR VIDOR 4000 WiFi, Espressif ESP32-WROVER, and Espressif ESP8266.

Figure 5 illustrates the schematic structure of the overall end-to-end system architecture of the Edgine-Measurify. At a broad level, the image depicts a sequence of commands flowing from the cloud to the edge, representing the downstream direction. Conversely, measurements are observed to flow from the edge to the cloud, representing the upstream direction. A running example is provided below.

A plant (*Thing*) is monitored through a humidity *Sensor*. Data are collected through an embedded *Board* in which Edgine (*Edge runtime*) is running. *Measurements* are shipped to the *Measurify server* whenever an internet connection is available. The stored information can be consulted and processed by front-end *Developers* (e.g., statistics can be collected and visualized) and is also available to the embedded board which can use it exploiting scripts. *Scripts* allow to perform operations on the previously collected measurements. For instance, if the humidity average level falls below a certain threshold, the water pump (*Actuator*) can be enabled to water the plant.

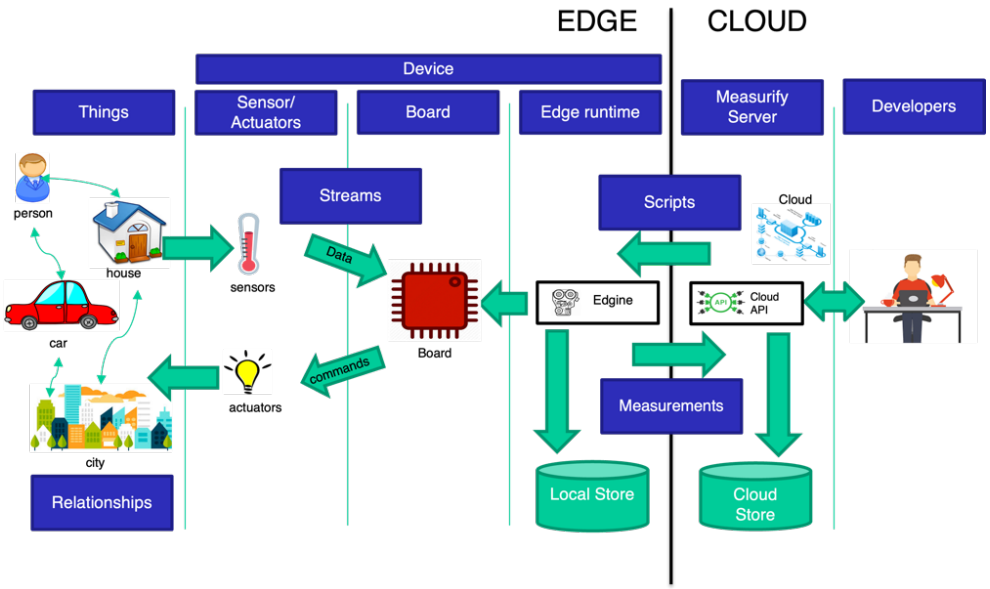


Figure 5: High-level Edgine-Measurify system architecture [9]

3.3 Experiments

The purpose of this section is to evaluate the capability of the presented system to meet various requirements in diverse application contexts. To achieve this objective, a comprehensive analysis of six practical applications utilizing the Edgine-Measurify framework is presented. These applications span across three significant domains, namely business, environment, and sport. The applications have been developed with the help of third-year students enrolled in an Electronic and Information Technology Engineering BSc program, working on their final thesis, who focused on the data collection aspect. This underscores the system's inherent simplicity, which can be leveraged for practical applications even by technicians lacking specialized professional expertise.

3.3.1 Industrial use-cases

In the following subsection, two scenarios illustrating the implementation of Edgine inside the realm of industries are delineated. The advantages obtained include real-time performance and the capability to operate offline (e.g., edge devices deployed in remote locations with inconsistent connectivity). These advantages are in contrast to the conventional data flows of cloud computing [159].

3.3.1.1 Shock monitoring

The initial application pertains to the implementation of an embedded system designed to assess shocks and bumps. The present study focuses on the application of logistics and transportation industries, where ensuring the preservation of goods during shipment is of utmost importance. The sequential processes involved in the implementation of the proposed system are illustrated in Figure 6.

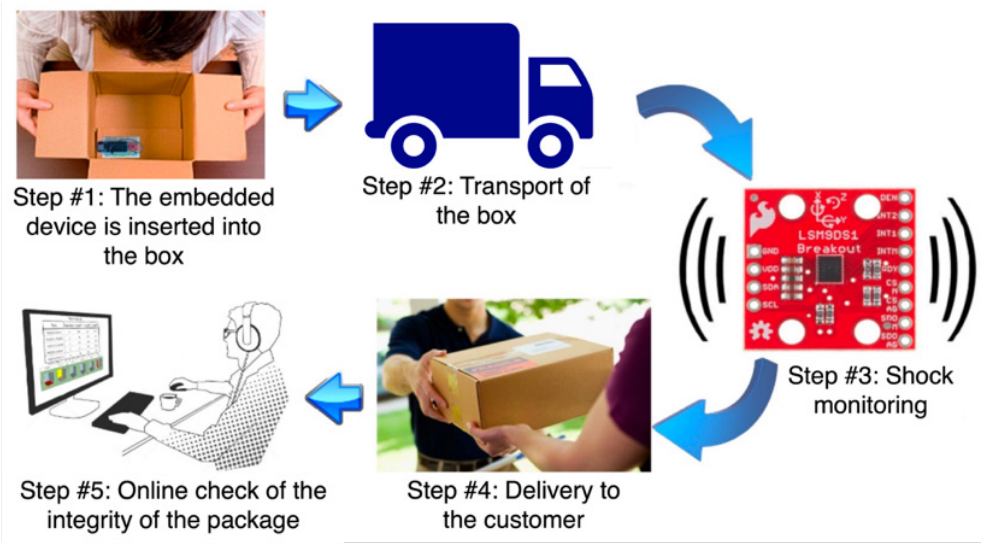


Figure 6: Stages of the shock monitoring system for the transport of goods [9]

From a developer's standpoint, the system comprises two unique phases: the shock monitoring phase, where data from sensors is collected and analyzed, and the package integrity check phase, which entails examining the recorded history of detected bumps stored in a cloud database.

The deployed edge system comprises a SparkFun 9DoF IMU Breakout-LSM9DS1 sensor, which incorporates a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer. The sensor is attached to an Arduino MKR WiFi 1010 board, which is equipped with the MKR MEM Shield to expand the device's memory capacity, enabling the storage of data locally in the case of connectivity issues with the cloud, which is very likely considering that the vehicle carrying the package will almost always be moving. The LSM9DS1 sensor is interfaced to the board via the Inter-Integrated Circuit (I2C) serial protocol, while the MKR MEM Shield is connected through Serial Peripheral Interface (SPI).

Based on the workflow outlined in the preceding section, the system autonomously initiates a connection to a designated Wi-Fi network and authenticates itself on Measurify using a username, password, and tenant credentials. Subsequently, relevant data pertaining to the object being measured is obtained from the cloud via a GET request, accompanied by executable scripts. After being stored locally, measurements are transmitted to the server, depending on the availability of network connectivity. The login and information retrieval phases are executed singularly within the initialization process, specifically within the Arduino setup function. In contrast, the cyclic execution of the Arduino loop function facilitates the processing of sensor data and the subsequent transmission of the results to the cloud. The code for monitoring shocks is derived from a

previous study ([160]), which employed a two-dimensional shock detection method in the context of vehicle crashes to determine the precise location of impact on the car. A three-dimensional approach to suit the transports specific use-case had been implemented. If the size of the shock surpasses the predetermined threshold, referred to as sensitivity, the shock value is transmitted to Measurify as an indication of a collision having taken place. The executable script utilizes the filter operation described in Table 5 to choose values that exceed a specified threshold. The position of the point of impact is also calculated and transmitted to Measurify to give a clearer idea about the extent of possible damage.

The visualization of data for the purpose of online package integrity verification involves the creation of a web page that incorporates a table displaying the most recent five occurrences of package effects together with their corresponding locations. An illustrative sample can be observed in Table 7. In order to enhance comprehension, a set of six images is presented, with each image depicting a different facet of the packaging (Figure 7). The graphic representation illustrates the magnitude of the impact by the utilization of a color scale consisting of four distinct keys: grey to indicate the absence of impact, green to represent a minor disturbance, yellow to signify a moderate jolt, and red to indicate the detection of a significant shock.

Table 7: Report of the five most recent shocks [9]

Time	Magnitude	Angle XY	Angle YZ	Angle XZ
12/09/2021, 16:45:50	3	251	230	254
12/09/2021, 16:45:50	6	270	185	268
12/09/2021, 16:45:50	3	92	353	252
12/09/2021, 16:45:50	3	93	341	260
12/09/2021, 16:45:50	4	67	287	277

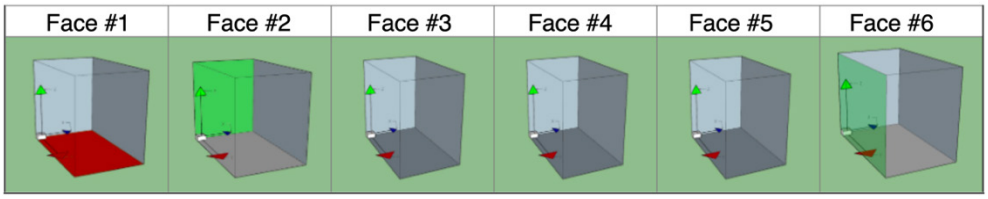


Figure 7: Graphical interface of the shock monitoring web page [9]

3.3.1.2 Tank level monitoring

The second application pertains to a monitoring system designed to measure the quantity of rainwater in a tank utilized as a storage facility for an aqueduct. The embedded system is dependent on the utilization of an Arduino MKR GSM 1400 board in conjunction with an HC-SR04 ultrasonic sensor. In addition, the program has been enhanced by the incorporation of an Arduino MKR SD Proto Shield,

thereby enabling the expansion of memory capacity. This augmentation facilitates the storage of data samples as a precautionary measure in the event of connectivity disruptions. The fundamental operational principle of the gadget is illustrated in Figure 8.

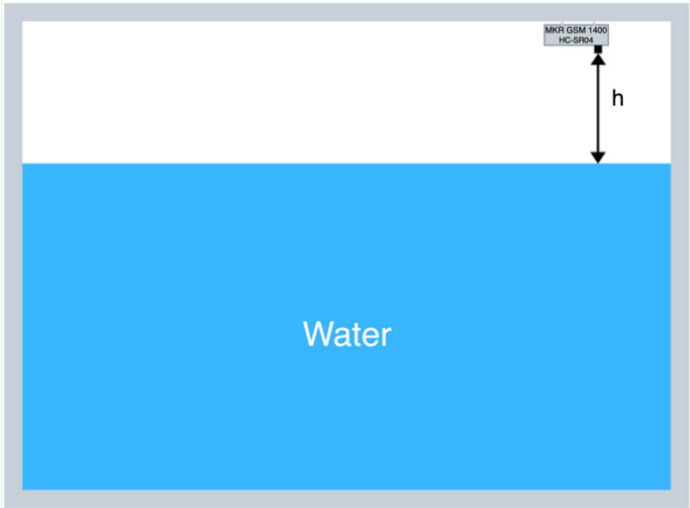
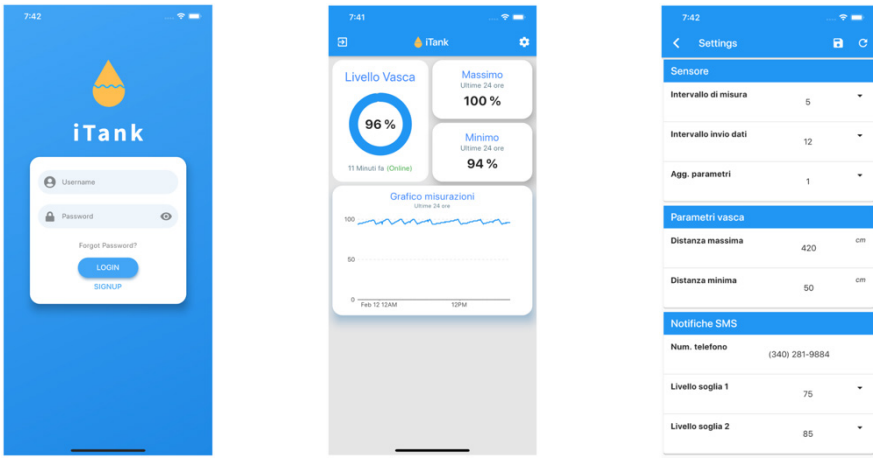


Figure 8: Schematic of the tank-level monitoring system [9]

Like the other embedded applications, there exists an initial setup step that is subsequently followed by a loop phase. The sole distinction is to the script that is linked to the utilized resource. In this scenario, the application relies on a filter operation. This operation selectively sends only those samples to the cloud whose values surpass either the upper threshold or fall below the lower threshold. Due to the remote location of the tank within a wooded region, the system has a GSM module integrated into the circuit board. This

module facilitates the transmission of SMS notifications to alert the tank maintainer of any potential risks.

To facilitate the monitoring and visualization of the collected data, a mobile application has been constructed using Flutter, an open-source framework that enables the creation of natively compiled, cross-platform applications for both iOS and Android devices [161]. Figure 9 displays the three primary pages of the application.



(a) (b) (c)

Figure 9: Snapshots from the tank level monitoring app. (a) Login page; (b) Tank monitoring info; (c) App settings [9]

3.3.2 Environmental use-cases

One example of an IoT application domain is environmental monitoring, as demonstrated in previous studies [162], [163], [164], [165]. Next, an overview of two use-cases from this perspective is provided. The first application is designed to monitor the air quality within an enclosed space, while the subsequent system enables the

assessment of a plant's condition and the environmental factors surrounding it.

3.3.2.1 Air-quality monitoring

The scenario in question relates to the implementation of a system designed to monitor the levels of noxious gasses within an enclosed area. The gases under surveillance include carbon monoxide (CO), nitrogen dioxide (NO2), and methane (CH4). The microcontroller employed in this study is the Arduino MKR WiFi 1010, while the gas sensor utilized is the MiCS-6814. The communication between the gas sensor and the microcontroller is established via the I2C protocol. The application life cycle remains consistent with earlier instances. The data samples are regularly transmitted to the cloud, as specified in the script, together with the identifier corresponding to the specific gas being measured.

In line with previous instances, a webpage has been developed with the purpose of presenting data in both graphical and tabular representations. Table 8 presents a representative illustration of carbon monoxide readings, whereas Figure 10 exhibits a visual representation wherein the values of the samples are graphically depicted in a temporal view.

Table 8: Measurements of the air quality system in terms of CO concentration [9]

Date	Time	Concentration (ppb)
21-1-2020	19:47	6044.16
21-1-2020	19:18	6044.16
21-1-2020	18:48	6193.68
21-1-2020	18:18	6472.43
21-1-2020	17:48	7155.01
21-1-2020	17:18	8858.53

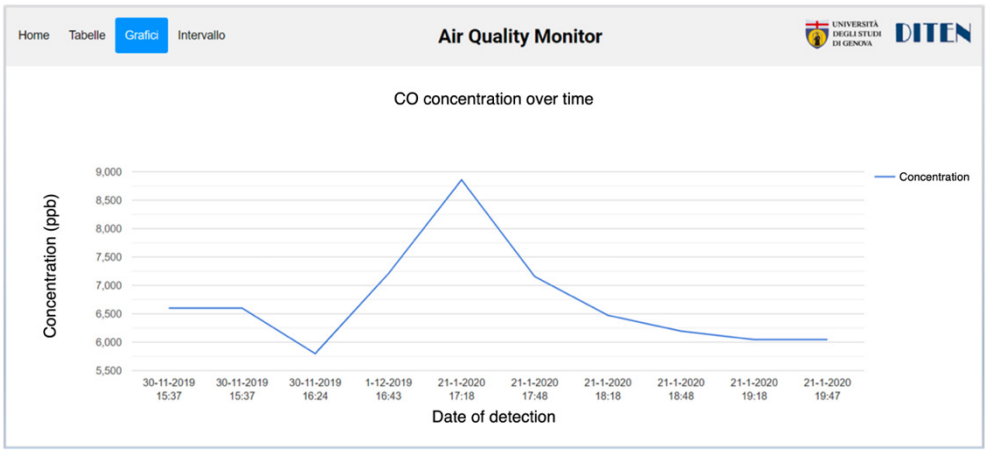


Figure 10: Measurements of the air quality system related to CO concentration [9]

3.3.2.2 Plant monitoring

The objective of this application is to facilitate the remote monitoring of plants and flowers. The selected microcontroller for this project is the Arduino Uno WiFi Rev 2. It is equipped with three sensors: the Sparkfun TSL2561 brightness sensor, the DHT22 Pro v1.3 air-humidity and temperature sensor, and the DFRobot SEN0193 soil moisture capacitive sensor. The luminance sensor offers data in Lux units, enabling the user to determine whether the plant is receiving adequate light. Additionally, the air-humidity and temperature sensor delivers samples in °C and %RH, respectively, offering information on the surrounding environment in which the plant is cultivated. The soil moisture sensor facilitates determining whether the plant requires additional water. Furthermore, an Arduino device has been equipped with a TFT screen to facilitate direct visualization of data from the source, as depicted in Figure 11.

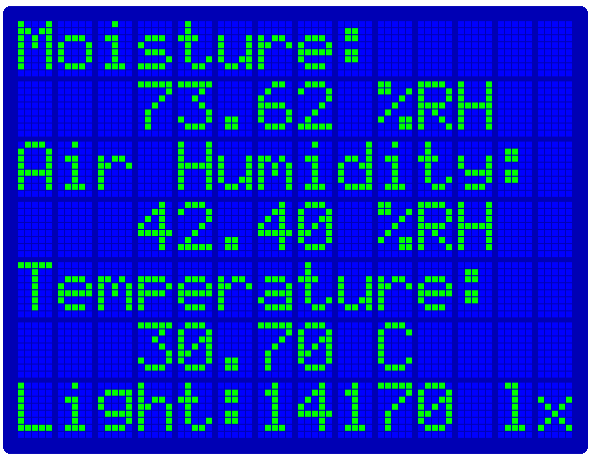
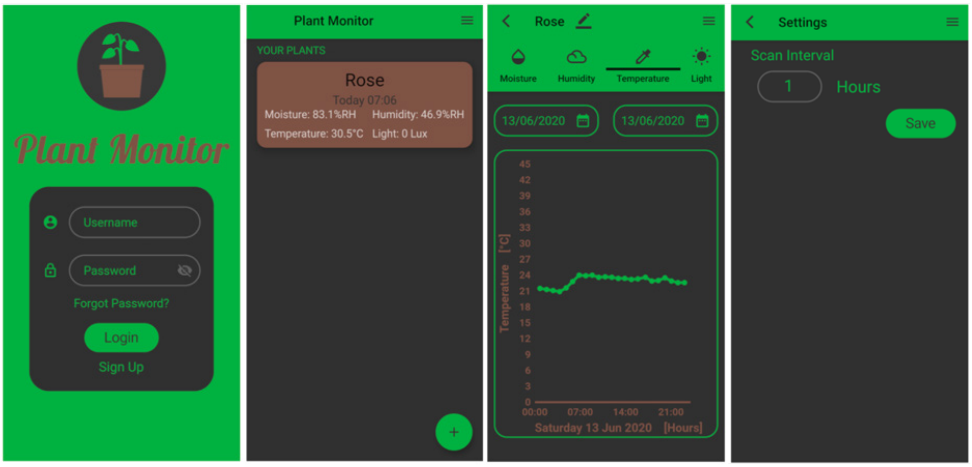


Figure 11: The plant monitoring system screen for local inspection [9]

Given the presence of diverse measurements in the application, Measurify captures four distinct features, each corresponding to a specific type of measurement. The accompanying scripts facilitate the specification of various sampling intervals and processing procedures for these distinct physical variables.

A mobile application has been built using the Flutter framework to enhance the usability of data interaction. The primary screens of the application are depicted in Figure 12. The application enables users to obtain a comprehensive overview of the plants being monitored, as well as access graphs that depict the changes in various features over a period. Additionally, the scan interval can be adjusted within the application.



(a) (b) (c) (d)

Figure 12: The plant monitoring app. (a) Login page; (b) Report on the monitored plants; (c) Temperature graph over time; (d) Scan interval settings [9]

3.3.3 Sports use-cases

The advancement in miniaturization of edge devices has facilitated the integration of digital technologies into sports activities through the use of wearable devices such as wristbands, cardio-bands, and smartwatches. Additionally, some devices can be attached to sports equipment such as shoes, tennis rackets, motorbikes, and cars. These developments have been documented in various studies (e.g., [166], [167], [168], [169], [170]). This subsection describes two sports applications that were developed by utilizing the Edgine-Measurify platform.

3.3.3.1 Smart bike

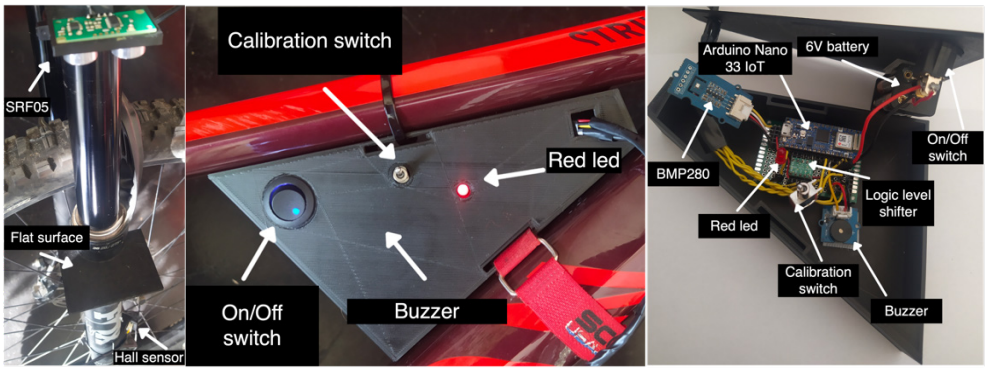
The objective of this work is to observe and track the performance of an enduro mountain bike throughout a descent over a designated trail, with the purpose of enabling athletes to assess their own performance. The dimensions that have been considered are as follows:

- The temporal evolution of speed;
- The vertical profile of the route;
- The duration of travel;
- The maximum lean angle of the bicycle;
- The maximum gradient of the route;
- The number of front fork compressions;

- The maximum velocity achieved.

The utilization of a Grove – Hall sensor has been employed for the purpose of obtaining speed and time data. The device, attached to the bicycle's front fork, facilitates the measurement of wheel revolutions, enabling the determination of speed, time, and distance covered. The DFRobot SRF05 ultrasonic sensor has been employed to measure the number of front fork compressions. Additionally, this device, which is also mounted on the fork, quantifies the displacement from the front hub, resulting in a reduction in distance during compressions. The measurement of the elevation profile of the route was conducted using a Grove – Barometer BMP280, a device capable of detecting atmospheric pressure in hectopascals (hPa). The measure is subsequently transformed into an altitude value, as detailed in [171]. In order to determine the lean angle of the bicycle and the slope of the route, the utilization of an accelerometer and a gyroscope was necessary. Given the specified criteria, an Arduino Nano 33 IoT board was selected due to its inclusion of a built-in LMS6DS3 sensor, eliminating the need for any further installation on the MTB. In order to mitigate the risk of system failures or damages, a logic level shifter, namely the Pololu 4-channel, was utilized. This was necessary due to the utilization of sensors, such as the hall sensor and ultrasonic sensor, which operate at a voltage above 3.3 V. These sensors function within the voltage range of 0-5 V. Figure 13 depicts the ultimate iteration of

the prototype, which encompasses the board and the sensors attached to the mountain bike.

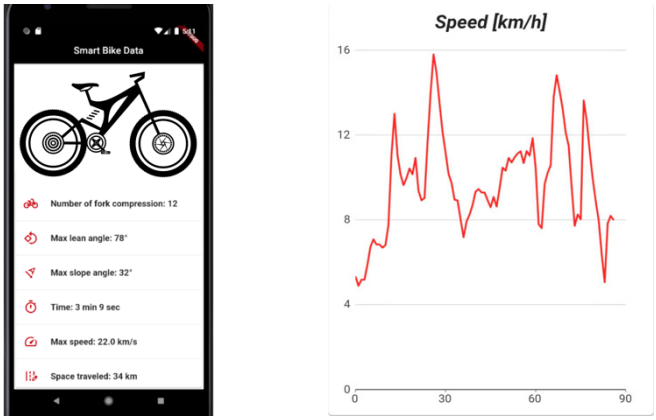


(a) (b) (c)

Figure 13: The smart bike main components. (a) The front fork with the ultrasonic and the hall sensors; (b) The main core of the system with a led and a buzzer to check if the system is running correctly; (c) The inside of the core, and a battery that powers the system [9]

Within the execution loop, the edge system acquires data samples from the sensors and establishes a connection with Measurify by utilizing the Edgine library. Subsequently, the system transmits the script-processed outcomes to the cloud for each individual characteristic. The scripts in question prescribe the acquisition rate of different signals and the desired computation, such as the maximum value inside a given sliding window, by utilizing the actions outlined in Table 5. Similar to the previous use-cases, a mobile application using the Flutter framework has been developed. The user is able to collect real-time data regarding the smart bike, as well as access instructive graphs illustrating information about the tour. The main

page of the application, as depicted in Figure 13, displays a graphical representation illustrating the progression of speed during a tour.



(a) (b)

Figure 14: The smart bike app layout. (a) The bike data; (b) Speed evolution over the tour [9]

3.3.3.2 Smart racket

The final application case within the domain of sports pertains to the development of a tennis racket assistance system that gathers data pertaining to an athlete's strokes. The project entails the development and deployment of a system capable of quantifying velocity, rotation, and point and angle of impact, with the objective of facilitating the differentiation of distinct stroke types. A total of six distinct stroke types have been taken into consideration: serve, forehand groundstroke, backhand groundstroke, overhead smash, forehand volley, and backhand volley.

The utilization of accelerometer and gyroscope sensors is crucial for this use-case. The Arduino Nano 33 IoT board was selected as the

development board because of its integrated LMS6DS3 sensor. The raw data from the sensors alone do allow to differentiate between the six distinct types of strokes. Hence, the primary objective of the application is to facilitate the generation of a dataset for every individual class. Subsequently, the provided information will be utilized to train a neural model using an edge learning strategy [172], with the objective of recognizing strokes. The objective of the suggested methodology is to enhance the approach presented in [173]. In this previous study, a motion sensor was attached to the racket to categorize the stroke type into three distinct classes: serve, groundstroke, and volley. In addition, the serve stroke involves the proposal of a regression model to estimate the speed of the ball. Similarly, for the groundstroke and volley, two models are suggested: a regression model and a physical model. The physical model is most suitable for proficient players who consistently execute stroke gestures, whereas the regression model is better suited for novice players who exhibit more variability in their stroke gestures. The Edgine system can be used to perform local data processing and transmit information to Measurify with minimal post-processing requirements. An application will be designed for mobile devices with the purpose of displaying real-time aggregated statistics from a tennis match. This data will be collected through specifically constructed scripts for the task, which will include information such as the count of forehand groundstrokes, the maximum speed of serves, and the percentage of forehand groundstrokes in comparison to backhand

groundstrokes. The integration of Edgine with the results of an integrated neural network, which classifies the raw sensor data into the six specified categories, is currently work in progress. The anticipated construction of the final prototype is projected to resemble the configuration depicted in Figure 15. The board will be fitted into the throat of the racket, which is protected by a pitted casing, because of the reduced diameter of the racket handle.

Figure 15: The smart racket project [9]

3.4 Users' feedback

As per the accounts provided by the students who actively participated in the above-mentioned projects, the utilization of the Measurify framework for the development of a comprehensive IoT application yielded numerous advantages. The system's user-friendly interface and straightforward installation process enabled the developer to prioritize dataflow design over cloud interfacing, resulting in a significant reduction in development time. Moreover, the utilization of scripts that sequence a certain number of instructions

facilitated the attainment of the projects' goals in a coherent and accurate manner. The main challenges faced revolved around the necessity for the developer to embed the Service Set Identifier (SSID) credentials within the code and a certain level of complexity in comprehending the interplay between the edge system and the cloud. Specifically, since the HTTPS queries initiated by Edgine are concealed within the library, there is a lack of quick indication for the user regarding the successful storage of data in the cloud server.

3.5 New features

Currently, Edgine's capabilities are being expanded by adding support to ML and DL models locally, with subsequent automated logging of results and evaluation metrics to Measurify using scripts. NN's configurations and weights are stored in the cloud database; Edgine is exploited to automatically GET the desired model and its description, to then perform on-the-edge inference and subsequently POST the results on Measurify. This extends Edgine's field of action from simple operations to the DL domain, allowing for a more in-depth data elaboration. This functionality is still work-in-progress but currently under test, with promising results, in the context of DRL models for low-speed maneuvering, which will be presented in Chapter 5.3. Also, future works with Edgine will include its evaluation in terms of performance, efficiency, latency, and resilience, particularly in executing ML and DL tasks.

4

Embedded Voice Assistant

After testing the potential of edge computing through the Edgine experiment, a transition was made to explore on-the-edge VAs, a domain highly relevant to the automotive industry. To accomplish this, however, we must advance from executing basic edge-side scripts to implementing DL models that permit speech processing and subsequent response generation for the user. In this instance, Edgine can be utilized to securely store significant system performance and utilization data on the cloud server. Being interested in developing an end-to-end VA system to be used on high-end embedded devices, first the needed modules are investigated, along with their interconnections, and performance. Then, the implementation of each block is presented, and experimental results are finally provided.

To provide an overview of the intended scope of the system, Table 9 outlines a compilation of voice command instances categorized based on their respective subsystems. Although the dialogue management system is primarily crafted for offline functionality, it is

also supposed to handle commands that necessitate an internet connection, such as downloading a song or map, engaging with an online navigator, or accessing weather information. Consequently, the development of the system has considered intents that involve an internet connection.

Table 9: Car VA use-cases

Subsystem	Examples
Phone	• Please, call John • Call mum via WhatsApp
Heating, Ventilation, and Air Conditioning (HVAC)	• Set the temperature to 22 degrees • Turn off the heating of the front left seat
Vehicle state	• How many liters are consumed per 100 km? • Tell me the tire pressure
Drive mode	• Select sport mode • Enter eco mode
Location-based services (offline)	• Take me to Rome • Find the nearest Italian restaurant
Media	• Stop the track • Fast forward the song by 15 seconds

4.1 Methodology

This section presents a comprehensive examination of the workflow, aimed at providing an overview of the various components, interfaces, and overall design. The subsequent section will provide an overview of the implementation details pertaining to the individual modules.

End-to-end VAs necessitate a thorough process that encompasses various stages, starting from the reception of incoming speech and concluding with the provision of feedback regarding the execution of the intended action. Consequently, a workflow was devised enabling the execution of the subsequent procedures:

- Human voice detection;
- Activation keyword detection;
- Speech recognition and conversion into text;
- Text conversion into meaningful data;
- Text conversion into speech to give feedback to the user.

The logical sequence of steps in a conversation with a VA is as follows. Initially, speech is detected, and if the spoken word is identified as a wake-word, the process of speech recognition is initiated. Subsequently, significant information, i.e., intents and entities, is derived from the inferred statement, with the purpose of

comprehending the user's query. The request that has been detected is ultimately carried out by the system, and the assistant provides feedback in the form of a vocal response.

The implementation of the conversational process can be achieved by employing a series of dedicated modules that are responsible for managing distinct aspects of the conversation.

The VAD module is employed to discern human speech from ambient noise of non-human origin. The module functions as a binary classifier with two distinct classes: "voice" and "noise". The system should remain in an active listening state at all times and should only be deactivated upon detection of a specific keyword. Reactivation of the system should occur at the conclusion of the conversation.

The activation of a KWS block triggers the interaction with the VA exclusively when a specific wake-word is pronounced. In the absence of this wake-word, no further steps are executed. This module also functions as a binary classifier, specifically designed to differentiate the wake-word from all other verbal utterances produced by humans.

When the user utters the designated wake-word, an ASR module is triggered, initiating the process of transcribing the spoken sentence into written text. The complexity of this module surpasses that of the preceding two modules, as it undertakes a multi-class classification task on the input speech. In this task, the classes correspond to the characters of the target alphabet as well as the punctuation marks.

The acquired text string is subsequently analyzed by an NLU module, which extracts intents and associated entities. Intent refers to a set of utterances that share similar meanings, while entities represent data that hold important values. These values are treated as parameters by the system when executing user requests, such as numerical values or geographical locations. The inclusion of this step is imperative to acquire a semantic interpretation that is comprehensible by the VA system, thereby enabling it to execute the user's requested task.

Following the execution of the intent, a TTS module is tasked with audibly informing the user regarding the outcome, whether it be a successful or unsuccessful execution of the provided command. Consequently, this stage entails employing a spectrogram generator to transform the raw textual output of the system into a visual depiction of the frequency spectrum of the audio signal across time. Subsequently, a vocoder is utilized to convert this visual representation into an audible speech waveform.

The sequence of tasks presented above is challenging, especially in an embedded setup, and the modules involved require substantial resources to attain a performance level comparable to that of cloud-based applications. Therefore, an estimation of the minimum specifications has been conducted, which include a memory capacity of at least 4 GB. While a GPU is not mandatory, it is recommended in order to achieve faster inference, with a minimum performance of 1

Tera Operations Per Second (TOPS). Additionally, the power consumption should not exceed 30 W. The latest generation of high-end embedded devices fulfills these requirements. The NVIDIA Jetson AGX Xavier was selected as the development board for our project. The selection of this particular system was based on its compact dimensions (105 mm × 105 mm × 65 mm) and its notable computational capabilities, as well as the availability of a powerful graphics processing unit (GPU) [10]. Table 10 presents the specifications of the board, along with the specifications of Amazon Alexa's cloud infrastructure (Amazon EC2 Inf1 [174]). This comparison aims to emphasize the gap in computing capabilities between cloud-based and edge solutions.

Table 10: Target embedded device vs Amazon Alexa, specifications comparison

Feature	**NVIDIA Jetson AGX Xavier**	**Amazon EC2 Inf1**
Memory	32 GB	Up to 192 GB
GPU	512-core NVIDIA Volta GPU with 64 Tensor Cores (32 TOPS)	Up to 16 AWS Inferentia chips (128 TOPS each)
CPU	8-core NVIDIA Carmel ARM v8.2 64-bit	Up to 96 2nd gen Intel Xeon (x86 64-bit)
Power	Between 10 W and 30 W	Not specified

The proposed workflow, as depicted in Figure 16, is designed to utilize the VA in a setting where reliable Internet access is not available and there is a need for minimal delay. Furthermore, in order to maintain a significant degree of extensibility, it has been strategically devised with the intention of incorporating additional languages in the future. The conversation steps mentioned above have undergone minor modifications in order to enhance the system's compatibility for embedded deployment. For instance, it was decided to integrate the VAD and KWS tasks into a unified module, referred to as Speech Classification (SC), motivated by the shared characteristic of binary classification. This unit conducts a binary classification task, where the two classes under consideration are the wake-word and its negation. The negation class encompasses non-human background noises as well as any other human-uttered words that are not the wake-word. The decision to activate the ASR block only after the wake-word is recognized enables faster inference while maintaining user privacy. This approach ensures that unintended information from the user cannot be gathered by the system.

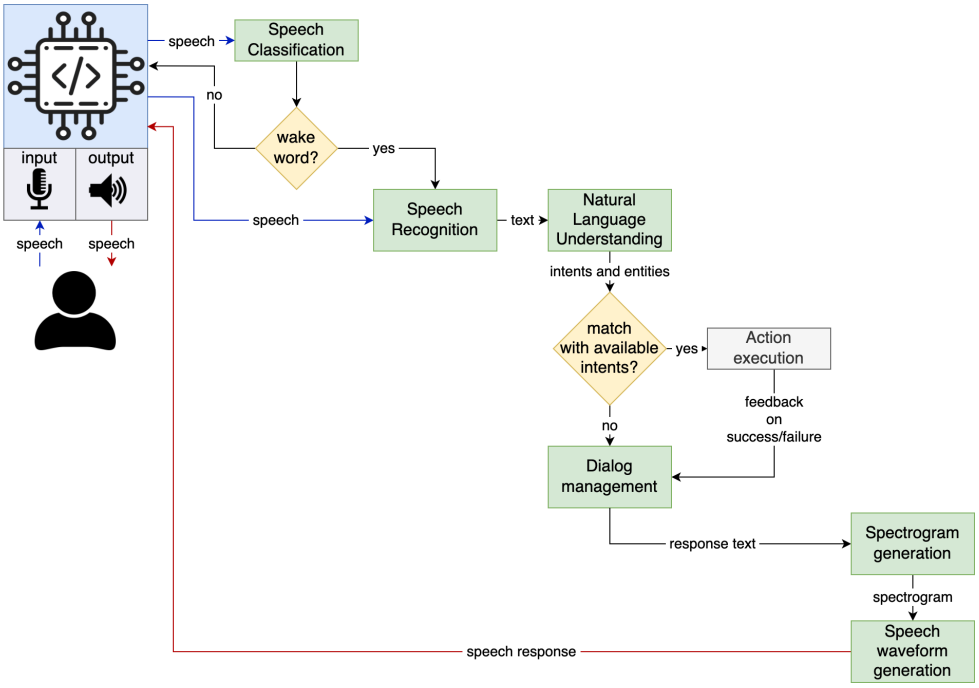


Figure 16: VA workflow

4.2 Automotive embedded VA implementation

Following workflow presentation, we now proceed to conduct an examination of the implementation of the in-car embedded VA.

In order to mitigate the issue of overfitting and promote generalization, a strategy of early stopping was implemented during the training phase of each model. Early stopping [175] involves the monitoring of a validation metric, such as validation loss or accuracy, during the training process. If the metric does not show improvement over a specified number of iterations, the training process is halted. The implementation of early stopping resulted in a substantial reduction in the training time of each model.

4.2.1 Speech classification

As depicted in Figure 16, the SC module is located at the forefront of the toolchain and operates continuously to detect the wake-word uttered by the user. The module in question is exclusively deactivated for the duration of the live interaction between the user and the VA. For the implementation of this task MarbleNet model [176] has been chosen, which is a deep neural network consisting of 1D time-channel separable convolution blocks, batch normalization, Rectified Linear Unit (ReLU), dropout layers, and Cross Entropy loss. The model described in this study, which is based on the QuartzNet model (section 4.2.2.1), was developed by NVIDIA and is accessible through the NeMo toolkit. The choice of this solution is because it can achieve comparable performance to state-of-the-art VAD models while utilizing significantly fewer parameters (88K compared to the 738K of the CNN-TD model proposed in [177]). The architecture of the system, similar to other VAD architectures, is designed to carry out a binary classification task. This task involves discerning the segments of an audio signal that contain human speech from those that do not. As anticipated in the previous section, to enhance the efficiency of our pipeline and accelerate inference, this architecture has been employed in an alternative manner. The two class labels were modified in order to differentiate between the genuine wake-word and other words spoken by humans, as well as the ambient noise present in the background. The dataset utilized for the wake-word class in our study was obtained from Lifetouch, an Italian company specializing in high-

technology solutions for the automotive and transportation industry [178]. In contrast, for the counterpart dataset, a collection of human-uttered words from the Lifetouch dataset has been generated and combined them with common background noises encountered in-car environments. These background noises were sourced from Freesound [179] using its API. These sounds encompass traffic sounds, such as those produced by cars and buses, as well as human conversations and other related noises. By employing a singular DL model, it was possible to successfully execute both VAD and KWS.

The model was trained for a total of 97 epochs, i.e., approximately 14,000 steps. During this training process, 93% accuracy was achieved. Figure 17 displays the training and validation Cross Entropy losses during training. The narrow discrepancy between the training and validation loss indicates a strong fit of the model.

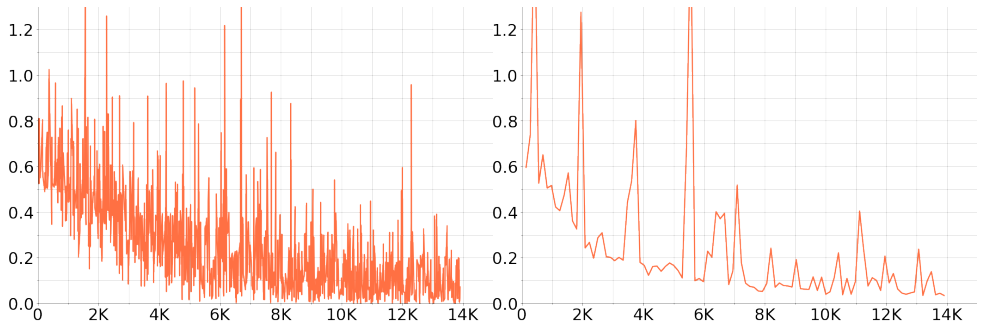


Figure 17: SC model training (a) and validation (b) losses over 97 training epochs

4.2.2 Automatic speech recognition

ASR plays a pivotal role in the overall process, alongside TTS, as these layers serve as the primary means of interaction between the system and the user. In order to achieve state-of-the-art performance, it is necessary to utilize high-dimensional datasets containing substantial quantities of speech data lasting several hours. As previously indicated, this issue relates to languages other than English or Chinese. In the context of ASR, the inclusion of multiple-speaker datasets is deemed necessary. To address the challenge, the NeMo toolkit provides the option of employing transfer learning. This approach allows us to capitalize on the knowledge embedded in a pre-trained English model, enabling the utilization of a smaller Italian dataset to fine-tune the model's weights. In this study, the initial model utilized for fine-tuning was the NVIDIA QuartzNet 15x5 model [180], pre-trained in the English language. Subsequently, fine-tuning was conducted on this model using the Common Voice Italian dataset. The QuartzNet 15x5 model was chosen due to the NeMo support for transfer learning and the model's relatively low parameter count (19M) in comparison to the current leading models [180].

4.2.2.1 ASR model

The QuartzNet model (Figure 18) is composed of blocks that are interconnected with residual connections. Each block consists of modules that incorporate 1D time-channel separable convolutional layers, batch normalization, and ReLU layers. The network consists of

an encoder and a decoder. The encoder is responsible for processing acoustic signals and generating a latent representation of the captured voice. The proposed approach can be interpreted as an acoustic model utilized to extract speech features, which are subsequently fed into a decoder responsible for generating textual output. The decoder utilizes the given representation to produce letters based on the alphabet of the target language. Consequently, the encoder exhibits the potential for cross-linguistic reusability, whereas the decoder's adaptability is contingent upon the specific target alphabet.

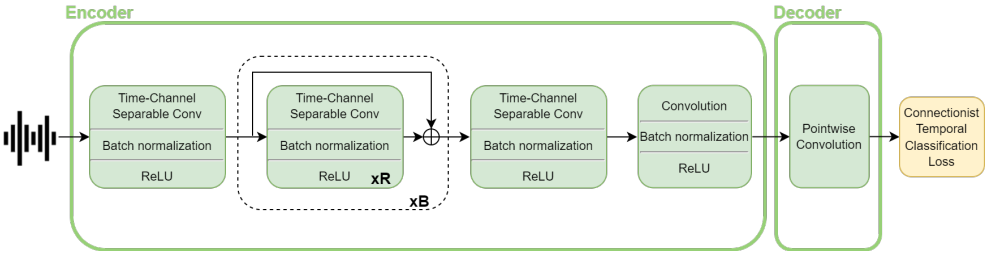


Figure 18: QuartzNet B×R architecture

The QuartzNet 15x5 model, trained on multiple datasets (namely: LibriSpeech [71], Mozilla Common Voice [58], WSJ [181], Fisher [182], Switchboard [183] and NSC Singapore English [184]) using Connectionist Temporal Classification (CTC) Loss [185], achieves a WER of 3.79% on LibriSpeech *dev-clean*, and a WER of 10.05% on LibriSpeech *dev-other*. It is worth to highlight that WERs under 5% are considered to be professional level [186]. Notably, the model has a relatively low parameter count of 18.9M [187]. This model is part of the NVIDIA NGC collection [188], available in the NeMo toolkit.

4.2.2.2 ASR dataset

To train the model, the Mozilla Common Voice dataset [58] has been employed, specifically the Italian variant known as Common Voice Corpus 8.0. This dataset encompasses a comprehensive collection of 310 validated hours of audio recordings in MP3 format. The decision to select this specific dataset for the transfer learning task was influenced by several factors. These include the dataset's size, the availability of labels for both the training and testing phases and the lower quality of other open-source datasets.

4.2.2.3 ASR transfer learning experiment

After the English pre-trained model and the Italian dataset were established, the process of transfer learning for the model was initiated. The NeMo toolkit has been exploited, leveraging the QuartzNet 15x5 model.

Prior to training the English model, it was essential to perform pre-processing on the audio files. This was required because the training data consisted of WAV files with a sampling rate of 16 kHz, while the Common Voice clips were in MP3 format with a sampling rate of 48 kHz. Consequently, the Common Voice clips were converted to match the training data of QuartzNet 15x5. In this phase, JSON manifests have been generated for the training, validation, and test phases. These manifests contain information such as the clip name, duration, and the sentence spoken. Additive features that were included in the

original TSV manifests but lacked relevance for our specific objective were eliminated. Furthermore, all the characters were converted to lowercase.

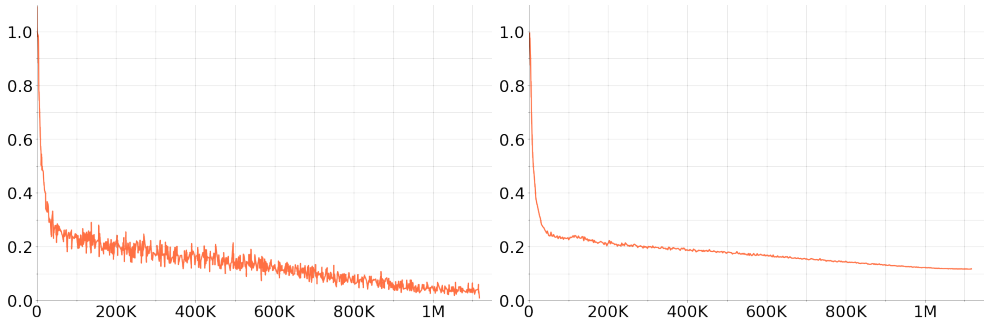
The model was fine-tuned using the NovoGrad optimizer, as recommended in [74] β_1 and β_2 were set to 0.95 and 0.25, respectively [189]. Furthermore, the learning rate was set to 0.001, utilizing a Cosine Annealing policy and a warm-up ratio of 12% [190]. The labels of the decoder were modified to correspond with the characters of the Italian alphabet. The network underwent training using the PyTorch Lightning tool [191] for a total of 256 epochs. The training process involved utilizing the CTC Loss, similar to the original QuartzNet 15x5 model. A batch size of 32 was employed, along with Automatic Mixed Precision (AMP) O1 [192].

Following the completion of 256 epochs and approximately 921,200 training steps, the obtained outcomes were deemed satisfactory. Figure 19 illustrates the WER of the model on both the training and validation datasets. The WER, which serves as the established benchmark for evaluating the accuracy of ASR models [193], is calculated in the following manner:

$$WER = \frac{(i_w + s_w + d_w)}{n_w} \quad (1)$$

where n_w is the number of words in the reference text, s_w is the number of words substituted (in the inferred text), d_w the number of

words deleted, and i_w the number of words to be inserted to transform the hypothesis text into the ground truth. Therefore, the WER is a numerical value ranging from 0 to 1. The final validation WER stands at 11.7%. In the context of speech transcription, it is widely acknowledged in the literature that WERs ranging from 20% to 25% are generally considered to be the upper limit of acceptable performance [194].



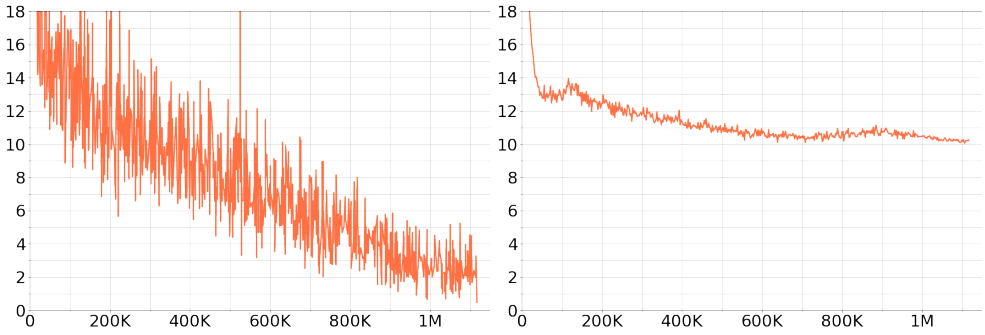
(a) (b)

Figure 19: Training (a) and validation (b) WERs over 256 epochs of training

In the subsequent section, a comparative analysis of these WER findings with alternative open-source offline solutions will be conducted. This analysis will demonstrate that the aforementioned results are comparable to, if not surpassing, the current cutting-edge implementations of cloud-based and desktop computing systems.

The provided visual representation in Figure 20 illustrates the progression of training and validation loss throughout 256 training epochs. The training process lasted 5 days and 21 hours, utilizing two 24 GB NVIDIA GeForce RTX 3090 GPUs. The model successfully

acquired knowledge of the fundamental patterns and correlations within the dataset, as evidenced by the training and validation loss plots. The training process was halted at 256 epochs due to the observation that the validation loss did not exhibit any further improvement.



(a) (b)

Figure 20: ASR model training (a) and validation (b) losses over 256 epochs of training

4.2.2.4 ASR models comparative

A comparative analysis between the trained model and the current state-of-the-art solutions has been conducted, focusing on metrics such as WER, Character Error Rate (CER), and transcription time. The comparison was conducted using the Vosk and DeepSpeech models, both of which are open-source and do not necessitate a connection to any cloud service. The DeepSpeech model utilized in this study was acquired through the application of transfer learning. The process involved initially training the English DeepSpeech model with the

Common Voice English dataset, followed by further training with the Common Voice Italian dataset, similar to our own model.

To ensure equitable conditions, the Common Voice test set has been employed, which had not been previously utilized by any of the models. The dataset consists of a total of 12,928 utterances, which corresponds to approximately 12 hours of speech. The calculation of WER and CER was conducted employing the Python JiWER tool [195]. Findings are presented in Table 11.

Table 11: WER, CER, and transcription times

Model	**Dataset**	**WER**	**CER**	**Transcription time** [a]
Ours (based on NeMo)	Common Voice Italian test set	11.7%	3.12%	0.215 s
Vosk	Common Voice Italian test set	29.8%	12.5%	0.464 s
DeepSpeech	Common Voice Italian test set	45.8%	13.24%	1.778 s

[a] The audio file considered is 5.269s long.

The presented table demonstrates that our model attains WER and CER values that are deemed satisfactory based on reference [194], which sets an upper limit of acceptability at 20-25%. Furthermore, our model outperforms both the Vosk and DeepSpeech models in terms

of WER and CER. In contrast, the QuartzNet 15x5 English model, which underwent training using over 3,300 hours of spoken English language data, demonstrates a WER of 3.79% on the LibriSpeech dev-clean dataset. However, its performance on the dev-other sets yields a WER of 10.05%, indicating notable variability in performance based on the specific test set utilized.

Considering the diverse range of word lengths present in the Italian lexicon, the performance of the model has been also assessed using the CER metric, which measures errors at the character level rather than the word level. Based on the WER, a word is deemed to be inaccurately recognized if a single letter within it is not correctly identified. The CER is derived using the subsequent equation:

$$CER = \frac{(i + s + d)}{n} \tag{2}$$

where n is the number of characters, while i, s, and d the insertions, substitutions, and deletions necessary to convert the inferred text into the ground-truth text, respectively. The obtained CER from our model validates the observations made regarding the WER. However, it also demonstrates that the DeepSpeech model exhibits a higher degree of closeness to the other two models when it comes to this kind of error. The observed behavior aligns with the findings presented in reference [196], which, in the first figure, shows the nonlinear correlation between WER and CER.

When evaluating the precision of the model, it is important to consider the nature of the test set being employed. Upon examining the individual utterances, it has come to our attention that they contain a significant number of archaic terms, as well as certain non-Italian words or proper nouns. Sentences such as *"Tom Sawyer and his friend Huckleberry Finn are witnesses of a homicide"* pose challenges in accurately identifying and transcribing them for various Italian models. These difficulties can result in increased WER and CER levels. This phenomenon is also observed in the training dataset, where utterances of this nature, although present in small quantities, do not significantly contribute to the effective training of the model for its application in a general language context. If the dataset had excluded utterances with uncommon usage, it is likely that the performance of all three models, as measured by WER and CER, would have improved. It is argued that introducing this change into the dataset would be particularly advantageous for tasks such as in-vehicle VAs, as they typically involve minimal usage of uncommon words by the user.

For the purpose of comparison, Figure 21 presents WER benchmarks provided by Picovoice [197] for the English idiom, along with the results for Italian (as previously displayed in Table 11). The Common Voice dataset was utilized for both tests, with one test conducted on the English version and the other on the Italian version. The depicted figure illustrates cloud ASR solutions, namely Amazon

Transcribe, Azure STT, Google STT (two versions), and IBM Watson STT, represented by orange bars. In contrast, the blue bars represent non-cloud-connected models, including the two Picovoice models, the two DeepSpeech models, the Vosk model, and our model. The performance of our model in terms of WER is similar to that achieved by cloud-based models in the English language. It is important to note that our model has undergone fine-tuning using a relatively limited Italian dataset, making the fact that its performance is comparable, if not superior, to English and cloud-connected models noteworthy.

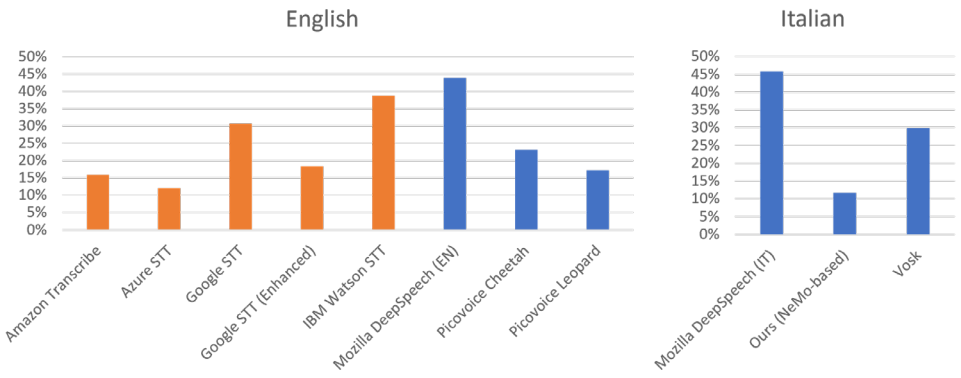


Figure 21: Comparative of WERs on the English **[197]** and Italian portions of the Common Voice dataset. Cloud-based solutions are in orange and embedded solutions are in blue

The final factor taken into account during the assessment pertains to the transcription time. This aspect holds significant importance when employing the model in ASR systems that necessitate a balance between accuracy and low latency. This is particularly relevant for time-sensitive applications like real-time navigation. The results presented in Table 11 demonstrate the superior performance of our

model compared to the other two models, particularly in terms of transcription time, for an audio file with a duration of approximately 5 seconds. The DeepSpeech model exhibits a computational time that is approximately eightfold greater than that of our NeMo model, thereby indicating its lack of suitability for real-time applications.

4.2.3 Natural language understanding

The NLU module is responsible for processing unprocessed text and transforming it into structured information, specifically intents and entities. These data are then utilized by the system to categorize the sentence according to its content. NLU and NLP are distinct in their approaches. NLU focuses on comprehending the meaning and significance of a sentence, while NLP primarily involves the conversion of an entire text into its constituent semantic elements. Rasa [80] has been selected as a tool for NLU due to its notable flexibility and customizability. This is due to its ability to create and utilize tailored NLU pipelines with desired modules. The components comprising our NLU pipeline are depicted in Figure 22 and elaborated upon in the subsequent discussion.

- WhitespaceTokenizer: divides a sentence into individual words based on the presence of whitespace characters;
- RegexFeaturizer: generates features for entity extraction and intent classification by identifying regular expressions that have been defined in the training dataset;

- LexicalSyntacticFeaturizer: iterates through the sentence using a sliding window and derives lexical and syntactic features;
- CountVectorsFeaturizer: converts a given text into a vector by considering the frequency of each word present in the text. Word token counts are employed as features;
- Dual Intent Entity Transformer (DIET) Classifier: an architecture based on transformers [13] capable of performing both intent classification and entity recognition [198];
- Entity Synonym Mapper: ensures that entity values detected in the training data are mapped to the same value if they can be defined as synonyms of other words;
- FallbackClassifier: responsible for classifying a message as the intent *nlu_fallback* if the confidence score is below a specified threshold, which has been set to 0.3. A fallback will also arise in the event that the two highest-ranked intents exhibit comparable levels of confidence.

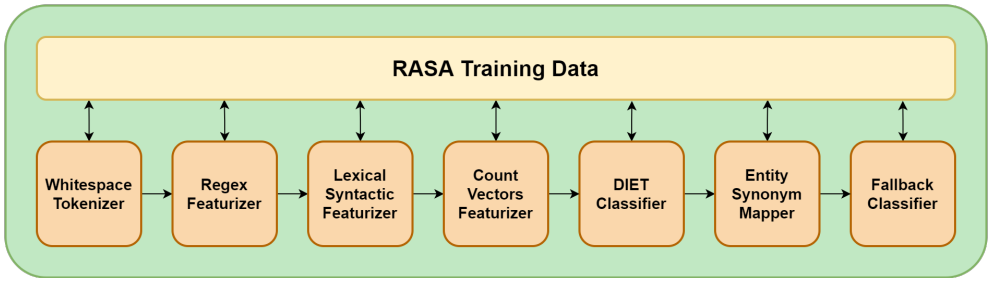


Figure 22: NLU model training pipeline

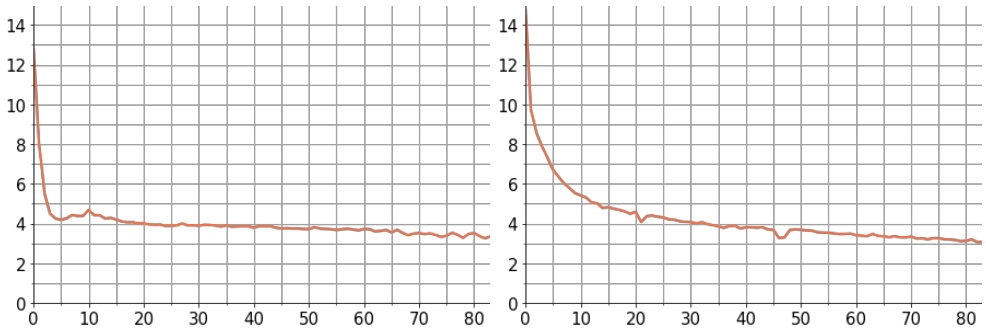
To explore different approaches, a spaCy pre-trained model has been investigated, leveraging its associated pipeline. However, this approach yielded inferior performance. It is argued that this phenomenon can be attributed to the inherent characteristics of our specific application scenario. Considering our requirement to create an NLU engine that focuses on specific terminology, specifically in-car intents as outlined in Table 9, it is likely that a generic pre-trained model is not suitable for this particular scenario. This assertion is supported by the Rasa documentation as well [199].

In accordance with the specifications described in Table 9, a total of 85 intents has been established. For each intent, an average of five exemplary sentences to be included in the training dataset has been curated, and formatted in YAML. Figure 23 presents an illustrative instance of adjusting the volume of a radio. The reporting of significant values, such as numerical quantities, follows a specific syntax that enables the system to recognize them as entities. This allows the system to extrapolate the value of these entities and subsequently execute the requested action. In the case under examination, the radio volume value that has been requested is identified as an entity, enabling the system to establish the desired value accordingly.

The DIET classifier was trained for 83 epochs, reaching an accuracy of 98%. Figure 24 reports the training and validation losses over the training.

```
- intent: set_volume
  examples: |
    - imposta volume a [3](volume_level)
    - imposta volume radio a [4](volume_level)
    - imposta volume musica  a [5](volume_level)
    - imposta a [3](volume_level) volume
    - imposta a [4](volume_level) volume radio
    - imposta a [5](volume_level) volume musica
```

Figure 23: Intent declaration for the training phase



(a) (b)

Figure 24: NLU model training (a) and validation (b) losses over 83 training epochs

4.2.4 Speech synthesis

The TTS module is responsible for generating speech based on an input text. It consists of two submodules: a spectrogram generator, which produces a mel (a perceptual scale of pitches judged by listeners to be equal in distance from one another) or Hz spectrogram; and a vocoder, which converts the spectrogram into audible speech. The two submodules have the option to undergo separate or joint training.

The Tacotron2 model [89] was chosen for its high reliability, open-source availability, and its success demonstrated in terms of Mean Opinion Score (MOS) [87]. Additionally, its integration within the

NeMo toolkit made it the preferred choice for our spectrogram generation needs.

Regarding the vocoder, empirical testing on various models within the NeMo framework has been conducted, including WaveGlow, SqueezeWave, UniGlow, MelGAN, and HiFiGAN. The selection was made based on the criterion of obtaining the most intelligible output voice, leading us to choose MelGAN.

The final component that was selected was the dataset. Regarding the ASR case, it is worth noting that the availability of open-source Italian datasets is quite limited. However, M-AILABS [60] has proven to be a dependable resource due to its provision of a substantial amount of speech data from a single speaker (18 hours in total for the male speaker). Figure 25 illustrates the architecture of the resulting TTS module obtained.

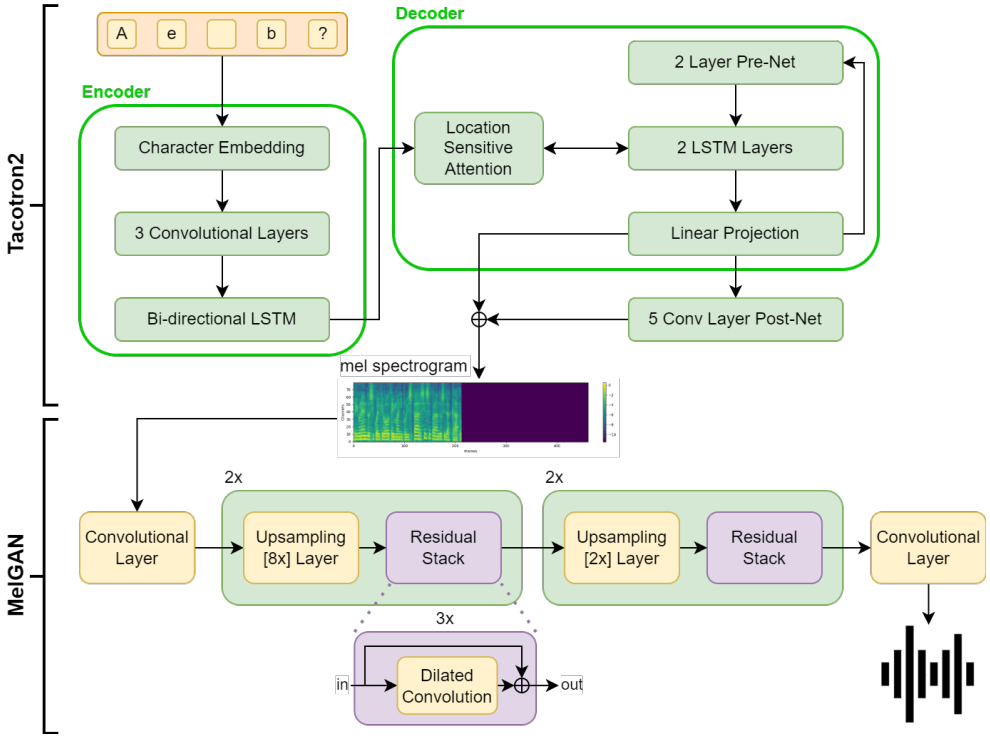
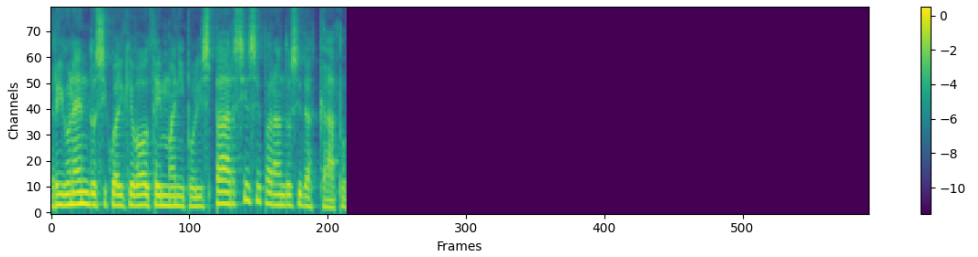


Figure 25: TTS architecture, including the Tacotron2 spectrogram generator and the MelGAN vocoder

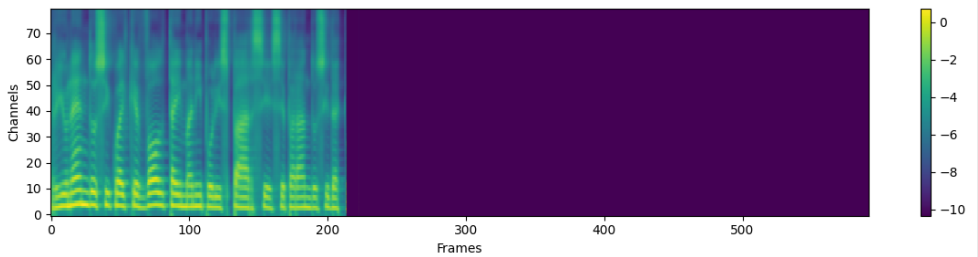
The Tacotron2 model is characterized by a sequence-to-sequence architecture. The system is composed of an encoder, responsible for generating a concealed representation of the characters in the input alphabet, and a decoder, which transforms this representation into a mel spectrogram. Once the inputs and location features have been projected into 128-dimensional hidden representations, the encoder output is then passed to an attention network, which serves to condense the encoded sequence into a context vector. Location-sensitive attention is employed [200], allowing to focus on specific portions of the encoder data that is to be used at each decoder step. The decoder is a type of autoregressive RNN that is designed to make

predictions of the mel spectrogram based on the encoded input sequence. The resulting output consists of an 80-dimensional audio spectrogram, where each frame is computed every 12.5 milliseconds. This spectrogram captures various aspects of speech, including word pronunciation, volume, speed, and intonation.

The Tacotron2 model underwent training for a total of 1,500 epochs, which is equivalent to approximately 73,500 steps. The findings are presented in Figure 26, which displays the ground truth and prediction of a sample mel spectrogram. Additionally, Figure 27 illustrates the training and validation loss curves, indicating a swift convergence of the model.

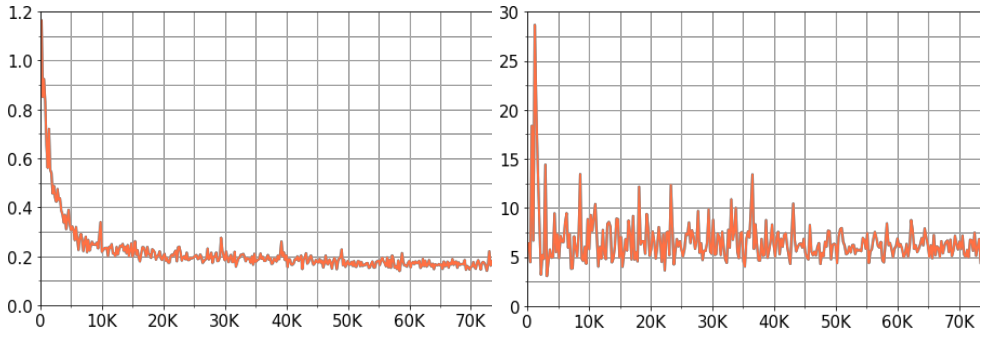


(a)



(b)

Figure 26: Target (a) and predicted (b) spectrograms by the Tacotron2 model



(a) (b)

Figure 27: Spectrogram generator model training (a) and validation (b) losses over 1,500 epochs of training

The two spectrograms that were generated exhibit a notable degree of similarity, which is further corroborated by the alignment plot depicted in Figure 28. The encoder, on the y-axis, receives an input character and its corresponding state at each iteration, producing a real vector that represents the network's current status at that particular time. Approximately 60 vectors are generated by the encoder. The decoder, represented on the x-axis, utilizes the vectors (y-axis), in order to generate audio spectrograms, specifically in the form of mel-spectrograms. The decoder also operates in a sequential manner, proceeding through approximately 200 steps. At each step, it determines the significance of each specific vector along the y-axis to generate audio frames at that particular time. An almost diagonal line results when audio frames are created by focusing on the correct input characters.

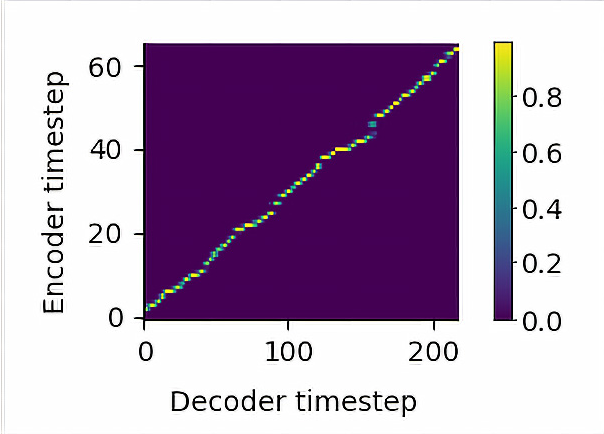
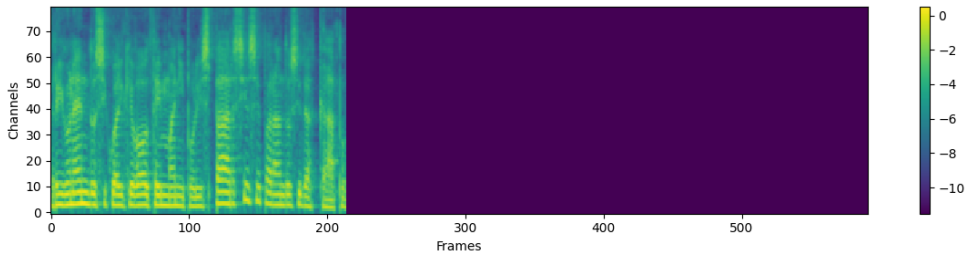


Figure 28: Tacotron2 model alignment plot after 1,500 epochs of training

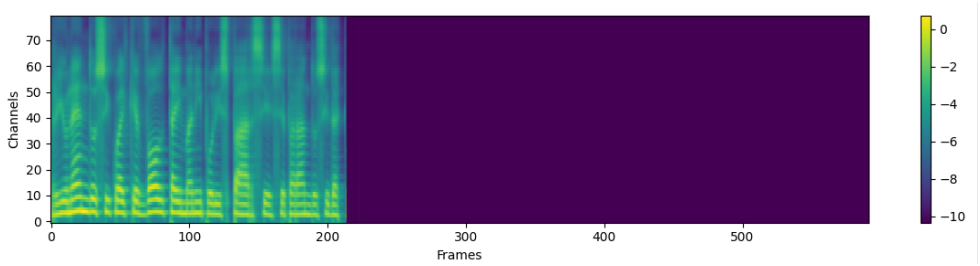
Regarding the vocoder, MelGAN is a non-autoregressive feed-forward CNN to transform mel spectrograms into time-domain waveform samples in a GAN setup. In contrast to conventional GANs, MelGAN diverges in its approach by excluding the utilization of a global noise vector as an input due to a difference in the generated waveforms when additional noise is fed to the generator. The generator is a feed-forward network that employs a fully convolutional architecture. It performs up-sampling on the input sequence, increasing its resolution by a multiplicative factor of 256. This up-sampling process occurs in four stages, with each stage increasing the resolution by factors of 8×, 8×, 2×, and 2×, respectively. This is necessary because the mel-spectrogram, which serves as the input, has a temporal resolution that is 256 times lower than the desired output resolution. The discriminator block exhibits a multi-scale architecture comprising three discriminators, each possessing an identical structure but functioning on distinct audio scales, namely

raw audio, raw audio down-sampled by a factor of two, and down-sampled by a factor of four. In this manner, each discriminator acquires discriminative characteristics by focusing on a particular target frequency range. For instance, the discriminator that operates on down-sampled audio lacks access to high frequencies, thereby specializing in the tuning of low frequencies exclusively.

The MelGAN model was trained for a total of 2,950 epochs, which corresponds to approximately 108,000 training steps. Figure 29 presents a comparative analysis of the target and predicted plots, illustrating the degree of similarity observed between the two. Figure 30 illustrates the training and validation losses of the model. The training loss of the generator exhibits a sharp decline around the 10,000th step, as depicted in Figure 30a. The presence of the discriminator (Figure 30b) is responsible for its capability to differentiate between waveforms generated by the generator and authentic waveforms. Figure 30c (validation) illustrates the evident convergence of the overall model. It is observed that the discriminator progressively improves its capacity to differentiate between real and generated waveforms, while the loss of the waveform generator remains constant, showing its robustness.

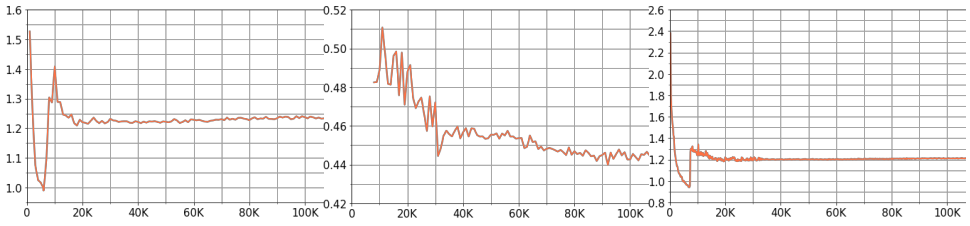


(a)



(b)

Figure 29: Target (a) and predicted (b) spectrograms by the Tacotron2 model



(a) (b) (c)

Figure 30: Vocoder training losses. (a) generator loss, (b) discriminator loss, (c) waveform generator validation loss

4.2.5 Toolchain

Following the determination of the components constituting the toolchain, they have been interlinked in accordance with the visual representation depicted in Figure 31.

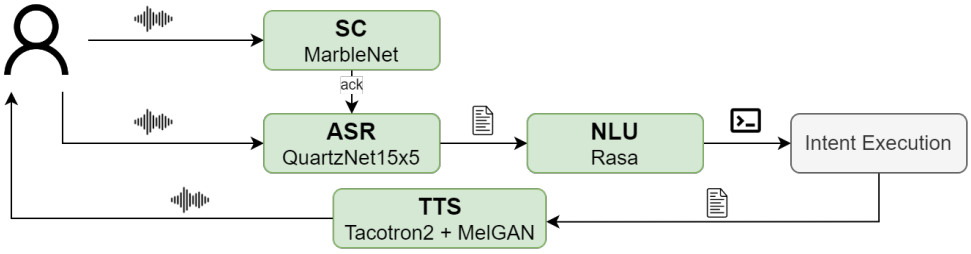


Figure 31: Block diagram of the toolchain

To ensure continuous activation of the SC module for detecting the wake-word from the user and subsequent activation of the following modules only upon wake-word detection, separate audio streams have been maintained. The activation of the ASR audio stream is contingent upon the utterance of the wake-word. The PyAudio library was utilized for this specific objective [201]. The PyAudio package offers Python bindings for PortAudio, an audio I/O library [202]. PortAudio provides a straightforward API that allows users to record and/or play sound by utilizing a basic callback function.

Upon detection of the wake-word, the system initiates the activation of the ASR audio stream. Subsequently, the user's command is transcribed into text in real-time. A Python library was developed to convert numerical values written in letters into digits, as the Rasa toolkit does not accept such values within sentences, as they are the output of the ASR model. The acceptable range of values is limited to 0 to 999 due to the specific requirements of the in-car use-case. This range is deemed sufficient as it covers numerical values such as radio volume or interior temperature, which are known to

never exceed 1,000. Consequently, the text is subsequently transmitted to the NLU engine, which transforms it into significant data, specifically intent and entities. In a practical application scenario, the intent execution block is responsible for executing the command, which is not addressed in this context. The user is provided with subsequent feedback that is determined by the recognized intent and the resulting execution. To fulfill this objective, a JSON file was generated that encompasses responses corresponding to every conceivable intent. The Rasa framework has the capability to handle responses, however, a dedicated JSON file has been developed to enhance its flexibility, specifically for the purpose of incorporating additional languages into the system. The inclusion of a new language is facilitated by this introduction, as it simplifies the process for the user. The user is only required to input the responses for each intent in the JSON file and provide sample sentences for the intents in the Rasa project for training purposes. Moreover, this approach enabled us to exclusively utilize the NLU component of Rasa, while disregarding the Core component, responsible for managing responses and story sequences (i.e., sequences of questions and answers). Consequently, this optimization streamlined the model and enhanced the efficiency of inference times.

In contrast to Rasa, our Tacotron2 model does not possess the capability to process numerical values represented in digit form, as the training dataset employed (specifically, M-AILABS) exclusively

contains numerical values expressed in alphabetic characters. Therefore, the process involved the utilization of the num2words open-source Python library [203], designed for converting numerical values into their corresponding textual representations. This library offers support for a wide range of languages, encompassing 38 different linguistic variations, including Italian. The user is subsequently provided with a response. In the event that the system is unable to interpret the command, or the intent is non-existent, a default response is given, prompting the user to rephrase their input. In this scenario, the VA will remain active rather than reverting to its inactive state, which involves waiting for the wake-word. Instead, it will remain in a state of readiness for further instructions from the user, with a timeout period of 10 seconds. After this duration, the VA will cease listening.

The models associated with the different components comprising the toolchain exhibit interchangeability, thereby facilitating the seamless replacement of these models with newly trained ones or the inclusion of models trained in different languages, without necessitating manual code modifications. The user is solely required to indicate the name of the new model in a JSON configuration file, alongside other significant parameters such as the anticipated sample rate or the language identification of the model (see Figure 32). This ensures enhanced ease of access. Furthermore, due to the system's

inherent capability to accommodate various languages, the language switch functionality is directly managed within the configuration file.

Figure 32: Set of parameters configurable within the JSON file

4.3 Results and discussion

This section presents the performance analysis results of the system that has been created to leverage the benefits of offline computation in an in-vehicle VA. The VA system was implemented on an NVIDIA Jetson AGX Xavier board, as described in section 4.1. Due to the adoption of a 64-bit ARM architecture by the board, it was necessary to build Rasa and NeMo from source as they lack native compatibility.

Our study involved the identification and training of neural models capable of attaining exceptional performance levels on the reference test sets. In this section, however, the system is evaluated in its entirety, first with an examination of the inference times, which are documented in Table 12. Outcomes indicate that the duration of the initialization process is considerable; however, it is important to note that initialization is solely necessary during system boot-up. Upon the

VA's execution, the cumulative duration of its operation amounts to around one second. This average value is derived from 20 distinct runs, encompassing a variety of commands with varying durations.

Table 12: VA execution times

Task	Description	Avg time
Initialization	Start-up of models (pre-downloaded) and audio streaming. Occurs once, at booting time	57.691 s
Speech Classification	Detection of the wake-word from the user	0.023 s
ASR transcription time	Time elapsed between when the user stops speaking and when the ASR model obtains the entire sentence (ASR works through PyAudio streaming while the user speaks)	0.132 s
Intent Recognition	Convert written text to meaningful data	0.071 s
TTS	Convert raw text to mel spectrogram, then to speech	0.790 s

The VA memory footprint within the Jetson AGX Xavier board is presented in Table 13. The disk space allocation for the VA amounts to approximately 4 GB, including the necessary Python libraries and the DL models, which occupy a relatively modest size of a few hundred MB (specifically, 236.7 MB). Regarding the RAM usage, 6 GB

is needed for the VA to perform inference, including the initialization of the system and the two PyAudio streams (one for SC, one for ASR). Hence, it is assumed that the system has the potential to function effectively on a less advanced board, albeit with the drawback of longer inference times likely attributable to limitations imposed by the lower-end GPU.

Table 13: VA memory usage

Model	Disk occupation	RAM occupation at runtime
SC	361.5 kB	0.4 GB
ASR	72.5 MB	1.2 GB
NLU	25.4 MB	1.4 GB
TTS	105.5 MB (Tacotron2) + 32.9 MB (MelGAN)	1.4 GB
Net total	236.7 MB	4.4 GB
Gross total	~ 4 GB (including required libraries and models)	6 GB (0.6 GB init + 1 GB PyAudio streams)

To comprehensively evaluate the overall system performance within the specified application domain, tests have been conducted using a custom dataset. This dataset comprised 135 sentences recorded in a noiseless environment and 135 in a noisy environment (i.e., amidst traffic), totaling 270 utterances from nine distinct speakers (three females and six males). Covering diverse use-cases outlined in

Table 9, speakers have been instructed to record application domain-related commands, with each user recording 30 sentences, evenly split between noiseless and noisy environments. These sentences were semantically aligned with one of the 85 intents in the Rasa training set, allowing us to assess all designated VA features with a relatively modest number of audio clips.

Table 14 presents sample sentences extracted from the dataset, along with their ASR transcriptions, the ground truth intent, NLU interpretation (i.e., inferred intent and, if present, entities), and the TTS-associated response. Noteworthy focus was given to critical cases, with errors highlighted in italics. For instance, the first sentence is correctly parsed. However, a common error observed involves recognizing an intent opposite to another, such as *"on"* instead of *"off"* or *"up"* instead of *"down."* This is evident in the second sentence where *"off"* is erroneously identified as *"on."* It is posited that the similarity between sentences, differing only in the verb, contributes to this error. While this can be mitigated by incorporating domain knowledge into the system, it underscores the challenge for the ASR module to recognize the [ɲ] phoneme (represented by the "gn" digram), present in Italian but absent in English. This suggests a potential weakness in the transfer learning approach from English to Italian, probably necessitating updates to the lower layers of the NN. Nevertheless, the third sentence demonstrates that the system can still extrapolate the correct intent even when a sentence is not parsed entirely accurately.

Table 14: VA end-to-end performance evaluation

Spoken command	**ASR transcription**	**Ground truth intent**	**Inferred intent**	**Entities**	**TTS response**
Alza la temperatura di otto gradi (Raise the temperature by eight degrees)	Alza la temperatura di otto gradi	turn_up_temperature	turn_up_temperature	temperature = 8	Alzo la temperatura dell'aria condizionata di otto gradi (Turning up the air conditioning temperature by eight degrees)
Spegni il riscaldamento sedile anteriore sinistro (Turn off the left front seat heating)	*Speni* il riscaldamento sedile *e* anteriore sinistro	turn_off_seat_heating	*turn_on_seat_heating*	seat_type = anteriore sinistro	Accendo il riscaldamento del sedile anteriore sinistro (Turning on the left front seat heating)
Quanti litri consumo ogni cento chilometri percorsi? (How many liters do you consume per hundred kilometers traveled?)	Quanti *detre* consumo ogni 100 chilometri *per corsi*?	show_fuel_consumption_100km	show_fuel_consumption_100km	None	Il consumo è di quattro litri ogni 100 chilometri (Consumption is four liters per 100 kilometers)

In the assessment process, apart from the WER and CER that specifically pertain to the performance of the ASR model, three supplementary metrics have been also taken into account [83]. These metrics consider the involvement of the NLU model, thereby encompassing the overall execution of the ASR+NLU system. The following additional metrics are taken into consideration:

- Intent Classification Error Rate (ICER): proportion of incorrect intent predictions to the total number of utterances;
- Slot Error Rate (SER): ratio of incorrect entity predictions to the total number of entities;
- Interpretation Error Rate (IRER): proportion of incorrect interpretations within a set of utterances. An incorrect interpretation refers to a sentence where either the entity or the intent prediction is inaccurate. IRER is calculated by dividing the number of incorrect interpretations by the total number of utterances. This metric is the most stringent and pertinent at both the application and system levels [83].

Table 15 presents the obtained results in all the metrics discussed. Higher values for both WER and CER in comparison to the values presented in Table 11 are visible. Our hypothesis is that this can be primarily attributed to the quality of the audio files being examined. Table 11 pertains to audio files sourced from the Common Voice dataset, which were chosen subsequent to a quality assessment phase

conducted through crowdsourcing. Conversely, Table 15 pertains to audio clips that were recorded within an ecological context. Regarding the IRER, it is noteworthy that our achieved result of 9.77% aligns with the user requirements outlined for the project. The aforementioned value decreases further when excluding the critical sentences discussed earlier, specifically those that can convey an opposing meaning based on a single word, typically the verb. In the absence of such sentences, the percentage decreases to 3.37% in the noiseless condition and 6.74% in the noisy condition. The IRER observed in this study is notably lower compared to the 22% error rate reported by [204] in their study on Alexa. It is worth noting that the task performed in [204] can be considered more complex, as it pertains to a wider domain. No other comparable system-level findings appear published in the literature. The values obtained under the presence of noise exhibit a comparable level of performance to the other conditions, thereby providing evidence for the resilience of the proposed system to background noise. This characteristic is particularly significant in the context of vehicular applications.

In conclusion, Table 16 provides a comprehensive comparison between our model and existing state-of-the-art solutions. Results demonstrate that our model is capable of offering complete offline end-to-end speech processing without compromising functionality, thus representing a significant advancement in the current state-of-the-art.

Table 15: Error rates in noisy and noiseless environments

Error type	Noiseless env	Noisy env
WER	17.71%	21.30%
CER	3.7%	5.1%
ICER	9.02%	9.35%
SER	3.09%	7.04%
IRER	9.77%	13.08%

Table 16: Feature comparison among VA systems

Model	Multi-language	Offline operation	SC	ASR	NLU	TTS
Google Assistant	☑	limited	☑	☑	☑	☑
Siri	☑	limited	☑	☑	☑	☑
Picovoice	☑	☑	☑	☑	☑	×
Vosk	☑	☑	×	☑	×	×
DeepSpeech	☑	☑	×	☑	×	×
Ours	☑	☑	☑	☑	☑	☑

All the metrics considered in this section have been measured through Edgine's scripts, which allowed for extrapolation and automatic storage into Measurify. In this scenario, Edgine served as a system evaluation tool, proving again its flexibility and wide range of potential use in the IoT context.

5

DRL for low-speed maneuvering

In the realm of automotive-applied artificial intelligence, an additional aspect of interest lies in the application of the DRL paradigm. This approach, which relies on an agent's direct exploration of the environment, proves to be highly efficient for tasks that demand learning a particular behavior (i.e., a policy) via interactions with the environment itself. An example of DRL's automotive application is in planning vehicle trajectories and executing parking maneuvers. This task synergizes well with DRL since it involves observing the environment through sensors and then responding accordingly by modulating the brake and accelerator pedals as well as the steering and gearbox.

The chapter is structured as follows. First, the RL paradigm is illustrated and then two experiments are proposed using the Unity game simulator [143]. Subsequently, a similar experiment is

performed using a more realistic engine, i.e., CARLA [25]. Finally, a comparison of the use between the two simulators is provided.

5.1 Deep reinforcement learning

In addition to supervised and unsupervised learning, RL constitutes an established paradigm within the ML field. RL is a technique used to train an autonomous agent to make optimal decisions within a specific environment. The acquisition of knowledge occurs through an iterative process of trial-and-error, facilitated by the agent's interactions with the surrounding environment. The interactions encompass periodic environmental observations as well as the corresponding actions executed by the agent. The rewards derived from the environment are gathered with the observations and actions, and then processed by an RL algorithm. This algorithm iteratively improves the agent's policy, which refers to its decision-making strategy, throughout the training process (see Figure 33).

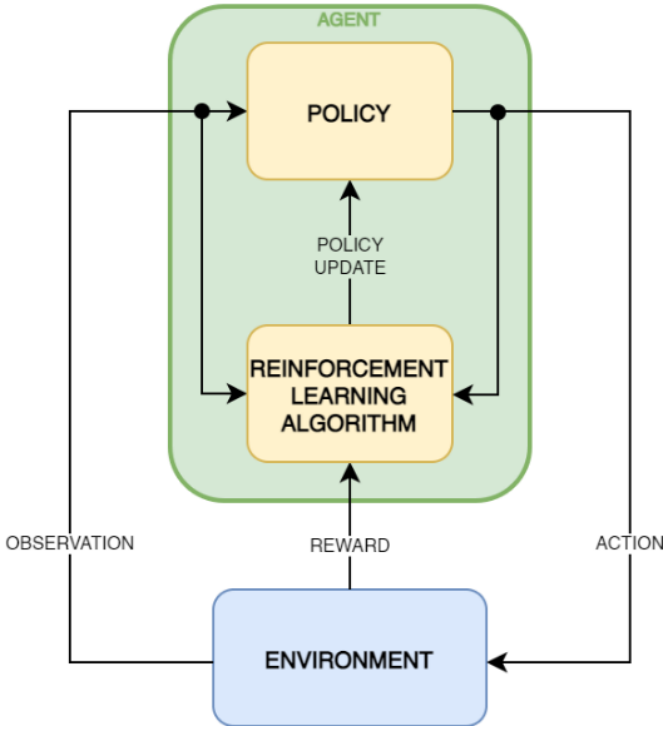


Figure 33: RL training loop

The policy represents a mapping between observations and actions. It can be realized in various forms, such as a look-up table, a complex function, or even a stochastic function that specifies a probability. During each training episode, the agent's objective is to optimize the overall cumulative reward to its maximum potential. In order to achieve this objective, the agent must prioritize actions that it has previously encountered and found to yield favorable outcomes within a given state. However, in order to ascertain such actions, it is necessary to experiment with actions that have not been previously selected. Therefore, it is essential for the agent to utilize existing knowledge while also engaging in exploration of novel actions, to

ensure optimal decision-making. The concept being referred to is the widely recognized trade-off between exploration and exploitation [205].

In the context of RL, it is common to represent a problem as a Markov Decision Process (MDP), which consists of a collection of states denoted as *S*, a transition function denoted as *T*, and an RF denoted as *R*. During each iteration, an agent is situated in a specific state *s*, performs an action *a*, and subsequently transitions to a new state *s'*. This transition is governed by a transition probability, denoted as *T(a, s, s')*, which is a value between 0 and 1. Additionally, the agent receives a reward, denoted as *R(s, a)*, based on the state and action taken. The agent learns a stochastic policy, which establishes a relationship between the state space and the set of available actions by assigning each of them a probability $p(a|s)$. The objective is to identify the optimal policy π^*, which aims to maximize the expected cumulative rewards within an episode, as represented by the following equation:

$$\pi^* = arg\max_{\pi} \mathbb{E}_{\pi}\left\{\sum_{k=0}^{H-1} \gamma^k r_{t+k} \middle| s_t = s\right\} \tag{3}$$

Where r is the reward, t the current timestep, and γ a discount factor ($\gamma \in [0,1]$) that controls how the agent values future rewards (i.e., low values encourage the agent to prioritize short-terms rewards, while large values give it a longer perspective). H is the horizon,

defined as the total number of steps in the MDP. It can be set to infinite or to a finite number if the episode ends after a certain number of steps or whenever a terminating condition is satisfied. In this case, γ values close to 1 are typically preferred to encourage the agent to actively pursue the objective. On the other hand, lower γ values allow for balanced long and short -terms rewards.

In conventional decision-making scenarios, the magnitude of the MDP state space is substantial, leading to the prevalent utilization of deep neural networks for modeling policies, referred to as DRL. Over the course of time, a multitude of DRL training methods have been devised, and they can be categorized from diverse viewpoints. One primary differentiation can be made between algorithms that are value-based and those that are policy-based. Value-based methods, such as Q-learning [205], employ a value network to approximate the Q-value. The Q-value represents the total expected reward for each individual *(s, a)* pair, assuming the agent consistently adheres to a policy π. The Q-table is initialized with random values and updated iteratively using a learning rate ranging from 0 to 1. The table update policy can be formulated as follows:

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a)\right) \tag{4}$$

where Q is the state-action value function, a the learning rate, r the reward, and γ the discount factor. The scalability of Q-learning is

limited by its structure, specifically in relation to the number of states and actions. This poses a significant challenge, especially in the context of continuous-action problems [205]. Furthermore, it has been observed that value-based RL algorithms are ill-equipped to handle scenarios involving stochastic policies, primarily due to their reliance on greedy action selection [206].

Policy-based methods, on the other hand, involve the direct learning of a policy, often achieved through the optimization of neural network weights using gradient descent. This optimization process aims to maximize the expected reward. The policy can be classified as either deterministic, such as the Deterministic Policy Gradient (DPG) proposed by Silver et al. [207], or stochastic, where actions are selected probabilistically, such as the Proximal Policy Optimization (PPO) introduced by Schulman et al. [208].

Actor-critic methods integrate the advantages of value-based and policy-based approaches [209], [210]. The actor network is responsible for executing the policy in a continuous action space, while the critic network is responsible for estimating the value function. The value function helps to update the policy using a historical sequence of states, actions, and rewards, resulting in reduced variance.

Another distinction can be made between on-policy and off-policy methods. In addition to the primary objective of optimizing the "target policy," off-policy methods employ a behavior distribution during

training to facilitate sufficient exploration of the state space. In the context of Q-learning, it is common to employ a $\mathcal{E}$-greedy strategy to determine the behavior distribution, which serves as the learnable policy. This strategy involves selecting actions according to a probability of 1-$\mathcal{E}$ for exploiting the current knowledge, and a probability of $\mathcal{E}$ for exploring randomly. This approach effectively balances the exploration-exploitation trade-off [128].

One final differentiation can be made between model-based and model-free approaches, which is contingent upon whether a model of the environment is built.

Figure 34 presents a non-comprehensive classification of RL algorithms.

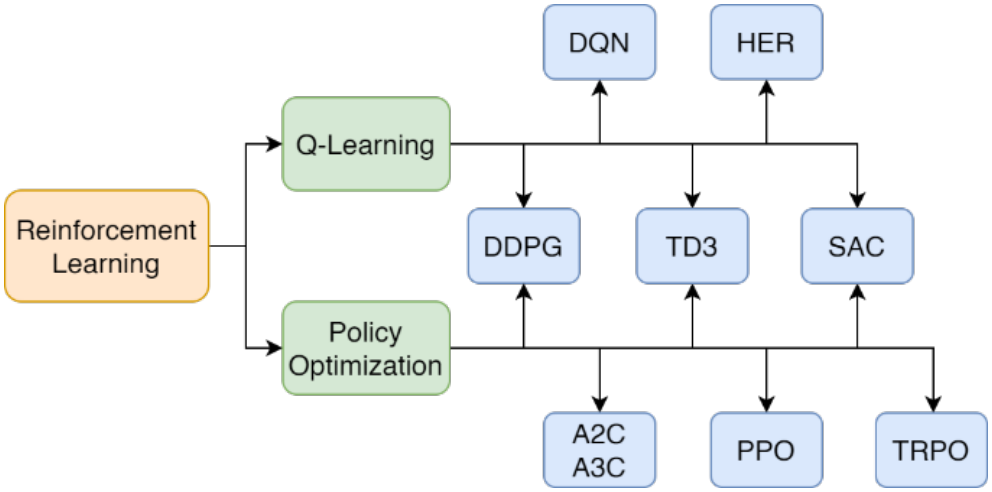


Figure 34: Taxonomy of DRL algorithms (non-comprehensive)

This study centers around the PPO algorithm, specifically designed to fit into a continuous action space in accordance with the

requirements of our problem. PPO, which was introduced by Schulman et al. in 2017 [208], has emerged as a prominent benchmark for addressing continuous control problems. Policy gradient methods introduce noise to the value estimation network due to its frequent updates with each experience sample. The Trust Region Policy Optimization (TRPO) method, as described by Schulman et al. [211] employs a strategy that constrains the policy gradient step in order to limit the variation of the policy. PPO, on the other hand, incorporates an objective function that facilitates the execution of multiple epochs of minibatch updates. This design choice allows PPO to harness some of the benefits associated with TRPO. However, PPO surpasses TRPO in terms of implementation simplicity, generalization ability, and sample efficiency.

5.2 Unity experiment

The first experiment involves a parking case-study in the Unity game engine, which is very popular in several domains, from video games to autonomous agent development [142].

5.2.1 Unity ML-Agents

The ML-Agents open-source toolkit [143] enables the utilization of Unity as a simulation environment for the development and training of autonomous agents. ML-Agents offers a Python API that facilitates the use of the major RL algorithms. This implementation is built upon

the PyTorch library. The key components of ML-Agents encompass the following:

- Learning environment: the Unity scene that serves as the setting in which the agent engages in observation, action, and learning. The ML-Agents Toolkit SDK facilitates the encapsulation of any Unity scene into a learning environment, enabling the specification of agents and their corresponding behaviors. It is feasible to concurrently train multiple agents, thereby substantially diminishing the training duration;

- mlagents-learn: the utility responsible for managing the training process. It is initiated by means of a .yaml configuration file. The file is organized into multiple sections, namely behaviors, environment, engine, checkpoint, and torch. The behaviors section outlines the training algorithm and its associated hyperparameters. The environment section specifies the path to the environment, any relevant environment arguments, and the number of parallelized environments. The engine section defines the rendering settings, including screen dimension, render quality, time scale, and whether to render the scene. The checkpoint section contains information regarding the creation of checkpoints during training. Lastly, the torch section determines whether the CPU or GPU will be used for training;

- Python low-level API: enables communication with the Unity scene during training;
- External communicator: an internal component within the learning environment that facilitates communication with the Python API;
- Python trainers: it comprises the algorithms necessary for training the agents and offers the "mlagents-learn" command-line utility and is exclusively integrated with the Python low-level API.

Figure 35 illustrates the architectural design of a representative learning environment within the ML-Agents framework. In this particular instance, the Python trainer is instructing two agents, specifically referred to as A1 and A2, through the utilization of the Python API. The Communicator module establishes a connection between the Agents and the Python API, facilitating the retrieval of essential environment parameters required for training purposes. These parameters include the target coordinates and the current location of the agent, among others. To ensure comprehensive analysis, the environment incorporates two additional agents. The first is an agent that incorporates a policy executed by a NN, which has been previously trained during a prior RL session. The second is a Heuristic agent, as it operates based on a predefined set of heuristic rules to determine its behavior.

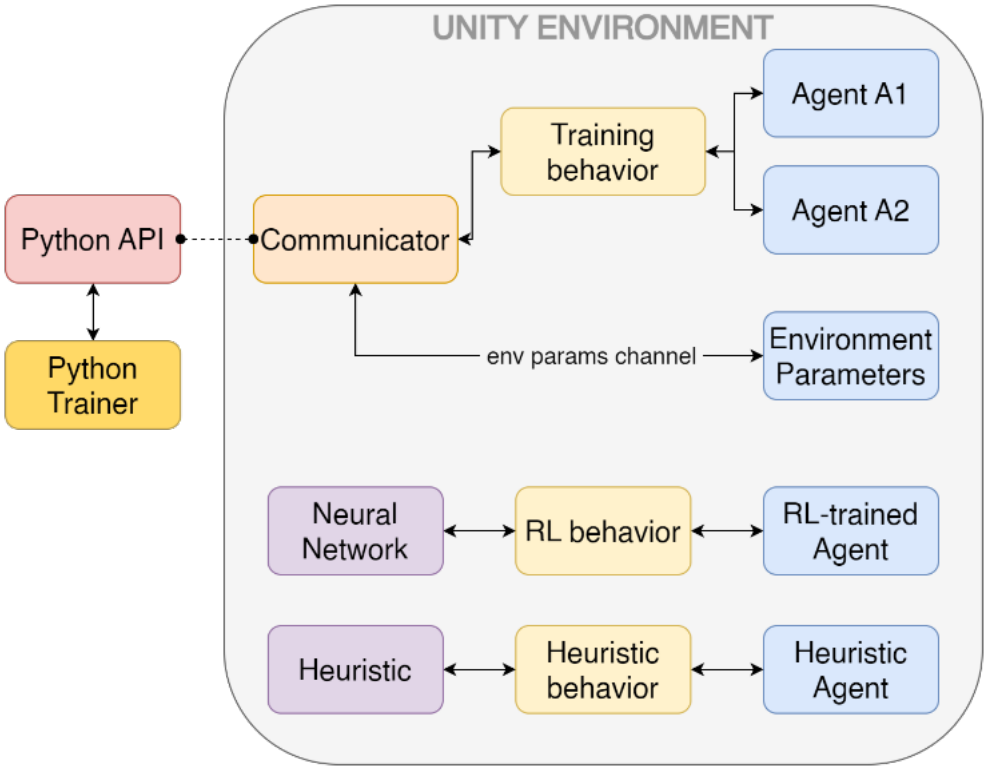


Figure 35: A potential ML-Agents learning environment

Figure 36 provides a comprehensive depiction of the standard workflow employed in ML-Agents projects. The primary phase of design and implementation involves the establishment of fundamental elements, including simulation settings, observations, actions, reward signals, and NN hyper-parameters. The subsequent training involves the execution of multiple simulation episodes, potentially incorporating adjustments such as modifications to the RF. Once the per-episode reward achieved by the agent reaches an appropriate threshold, the training process is terminated, and the acquired policy can be subsequently evaluated through pure inference. In the event of a successful test, the model may be deployed.

However, if the test is unsuccessful, the training process must be restarted using a model that has been updated by the designer based on the knowledge and experience acquired.

The per-episode workflow, which includes the configuration of randomized parameters to enhance the agent's generalization abilities, is outlined in Figure 37.

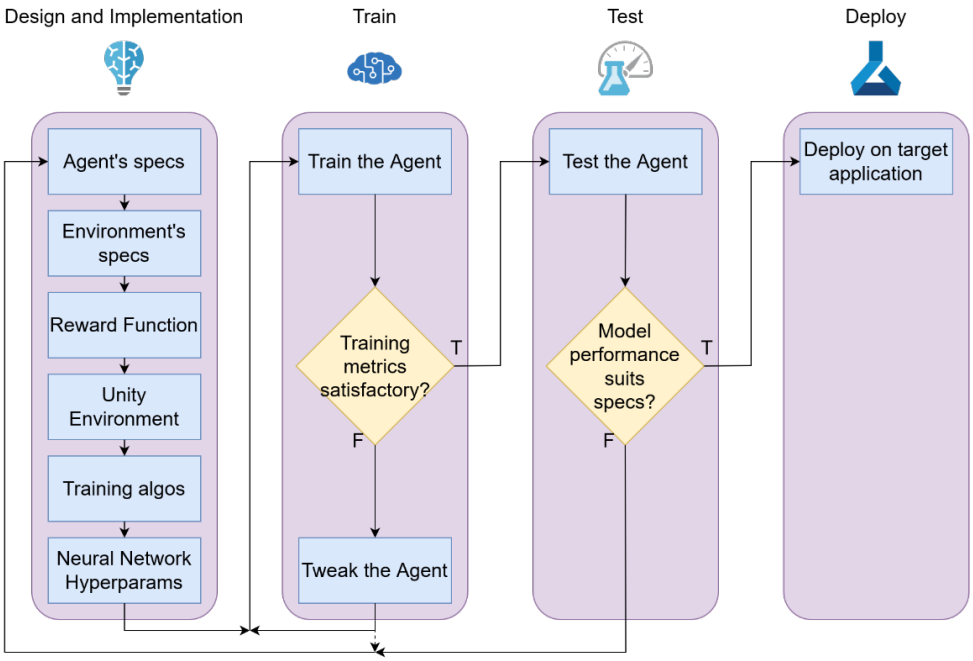


Figure 36: ML-Agents project development workflow

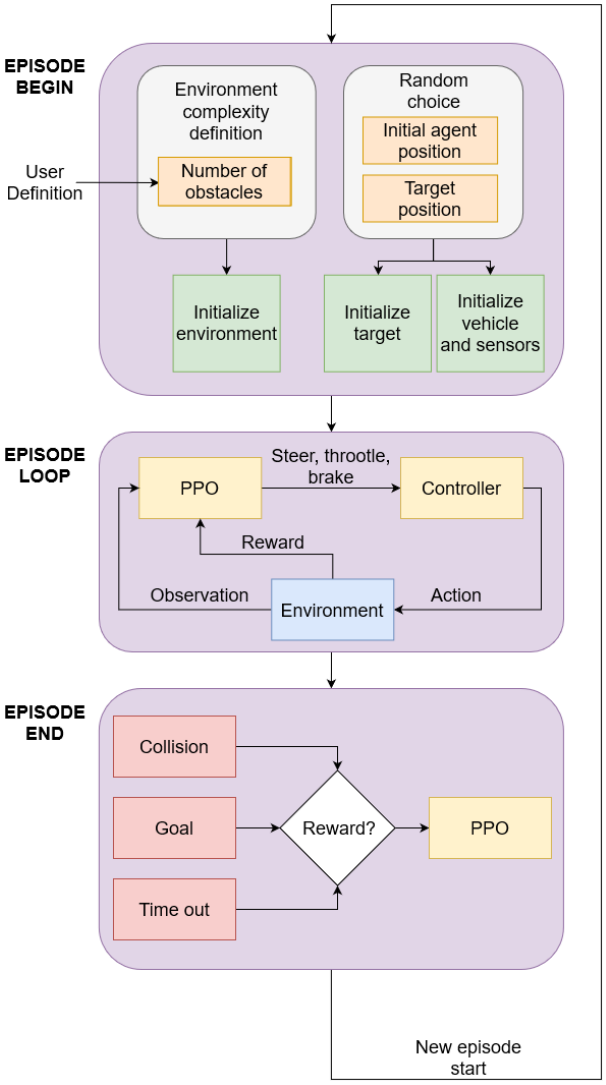


Figure 37: ML-Agents per-episode workflow

5.2.2 Experiment setup

To incorporate the DRL agent into the vehicle, a basic three-dimensional model that was accessible from [212] has been used. The collider component of the vehicle, which facilitates collision detection, is in the form of a parallelepiped with dimensions 3.90 × 1.70 m.

The utilization of a basic kinematic bicycle model for the vehicle is justified by the low-speed maneuver setup [213]. The equations pertaining to this model, along with a corresponding illustration in Figure 38, are as follows.

$$\begin{aligned} \dot{X} &= V\,cos\big(\psi + \beta(u_2)\big) \\ \dot{Y} &= V\,sin\big(\psi + \beta(u_2)\big) \\ \dot{V} &= u_1 \\ \dot{\psi} &= \frac{V}{l_r} sin\big(\beta(u_2)\big) \end{aligned} \tag{5}$$

where u_1 is the acceleration command and u_2 the front wheel angle used as a steering command. $\beta(u_2)$ is the slip angle at the center of gravity:

$$\beta(u_2) = arctan\left(tan(u_2)\frac{l_r}{l_f + l_r}\right) \tag{6}$$

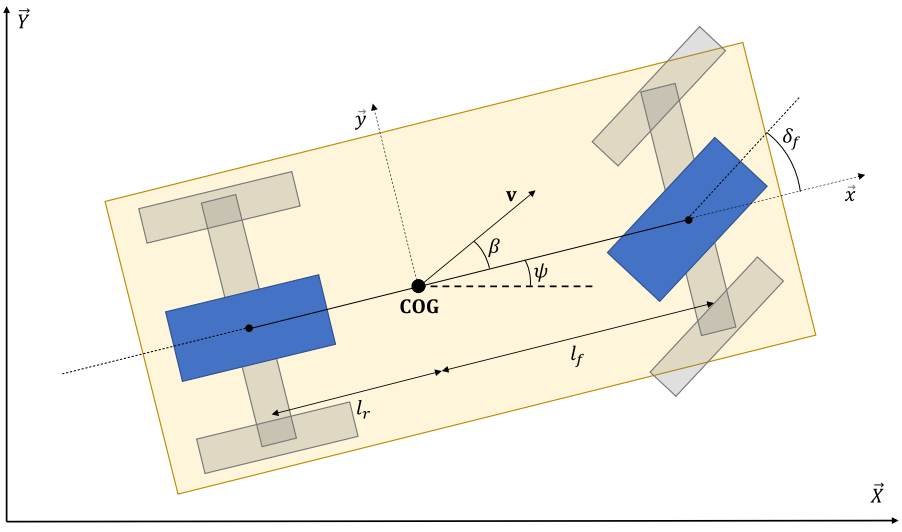


Figure 38: The kinematic bicycle model

The agent receives information regarding the vehicle's position, the target's position, and the speed vector from the environment. In addition, the system can receive a continuous flow of data from both a lidar sensor and a camera. The lidar sensor, which has a field of view spanning 360° and an angular resolution of 10° (see Figure 39), is positioned at the apex of the roof, precisely at its center. This sensor generates a one-dimensional array of data. The length of the radius is 5 meters. This deliberate choice of a short-range was made in an attempt to improve the agent's ability to generalize. The camera is positioned above the windshield and has a vertical field of view of 60° and a horizontal field of view of 120°. It has a resolution of 84 × 84 pixels. The lidar is utilized via the ML-Agents Raycast sensor component, while the camera is employed through a Camera component. Equation (7) presents the comprehensive observation vector.

$$\begin{gathered} S_{lidar} = (x_{vehicle}, y_k, w_{k-1}) \\ S_{camera} = (x_k, v_k) \\ S_{env} = \left(\text{agent}_{vel}, \text{agent}_{dir}, \text{target}_{dir}, \text{distance}_{agent \to target}\right) \\ S = (S_{env}, S_{camera}, S_{lidar}) \end{gathered} \tag{7}$$

The agent possesses the capability to manipulate the motor torque, the brake torque, and the orientation of its front wheels. The vehicle's steering angle range is in the range [-π/6, π/6]. The brake torque range spans from 0 to 300, while the motor torque range extends from -400

to 400. This enables the vehicle to execute both forward and backward maneuvers.

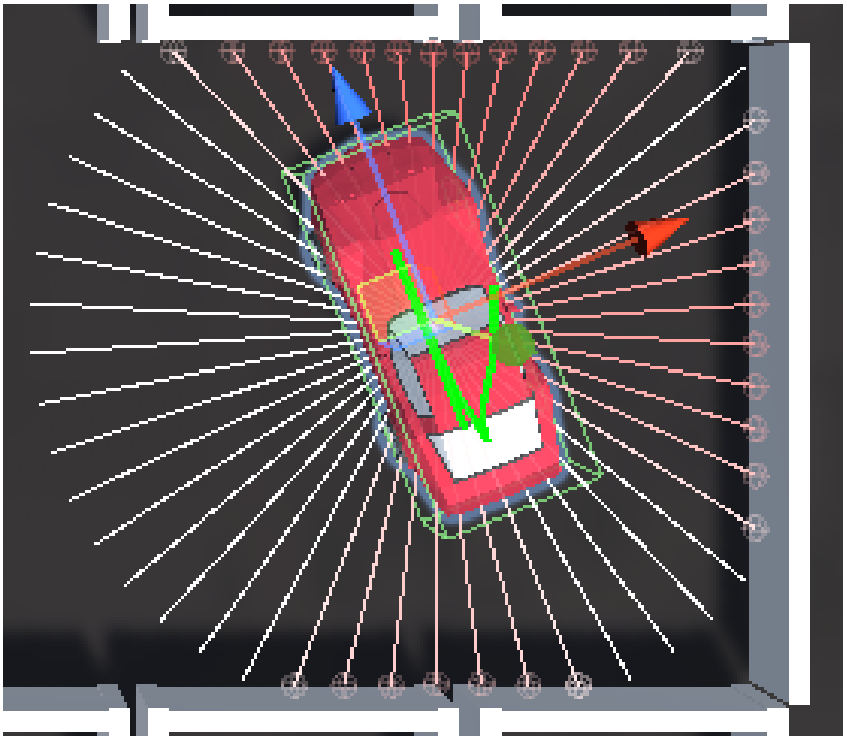


Figure 39: View of the Raycast lidar sensor

The selected neural network model is an MLP consisting of two hidden layers, each containing 256 neurons. In the presence of a camera sensor, the neural network incorporates two supplementary convolutional layers for the purpose of preprocessing the visual signal. Table 17 presents a synthesis of the values that characterize the network architecture.

Table 17: ML-Agents NN architecture

Layer	Sizes	Description
Input layer	(408 or 84×84×3)+8	408 lidar values or an 84×84 RGB image + 2D values of distance between agent and goal, agent's speed, agent's heading, and goal's heading
Convolutional layers	1st Kernel size = [8,8] 1st Stride size = [4,4] 2nd Kernel size = [4,4] 2nd Kernel size = [2,2]	2 convolutional layers to pre-process the camera input, when provided
Dense layers	256	2 layers
Output layer	3	Possible agent's actions: throttle, steering, brake

5.2.3 Results

This section provides a description of the two tests conducted to validate our approach, namely parking in a garage and navigating within an area with randomly placed obstacles.

The results presented in this section, along with the corresponding source code, can be accessed at the following URL: https://github.com/Elios-Lab/pathfollowing.

5.2.3.1 Garage environment

The first developed environment is visible in Figure 40. A garage-like setup is presented, with ten parking lots in a 400 m² area. Each lot, with dimensions of 5.3 × 3.5 meters, is enclosed by walls. The central corridor, which allows for bidirectional vehicular traffic, measures 20 meters in length and 8 meters in width. The objective of the DRL agent is to determine and implement a trajectory from an arbitrary starting location to a specified destination while ensuring avoidance of any potential wall collisions.

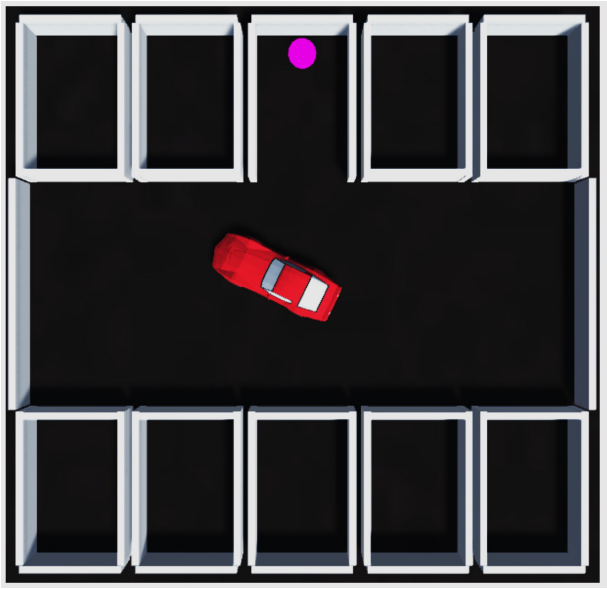


Figure 40: The Unity garage environment

The RF plays a crucial role in the design of autonomous agents as it serves as the mechanism by which the agent's behavior is influenced by the environment. The initial step involved replicating the RF (Equation (8)) from a well-known open-source project that utilizes

ML-Agents [214]. The proposed incentive system imposes penalties on the agent for engaging in collisions, while simultaneously providing rewards for its motion and successful attainment of the target. The proper alignment of the vehicle within the parking lot is also rewarded.

$$reward = \begin{cases} 0.2 * |alignment| & if\ goal\ and\ car\ parked\ facing\ the\ wall \\ 0.8 * |alignment| & if\ goal\ and\ car\ parked\ facing\ the\ road \\ -0.01 & if\ collision \\ 0.001 * target_{dir} * target_{velocity} \end{cases} \quad (8)$$

where $alignment = ||agent_{vel}|| * ||agent_{dir}|| * \cos(\vartheta)$ and ϑ is the angle between the EV heading vector and the EV-target conjunction vector.

By conducting multiple iterations of experiments, it was possible to enhance the RF, leading to improved outcomes in terms of both accuracy and convergence time. The ultimate equation is represented as Equation (9).

$$reward = \begin{cases} -1 & if\ collision \\ 0 & if\ goal \\ c_1 * ((c_2 * d^2) + \dfrac{c_3 * (1 - cos\,\theta)}{d + 1}) \end{cases} \quad (9)$$

where $c_1 = 0.01$, $c_2 = -0.01$, and $c_3 = -1.5$.

The function computes the sum of two types of rewards: sparse rewards, which are provided when a specific event takes place, and dense rewards, which are given at each step of the simulation. The first sparse reward imposes a penalty on the agent in the event of a

collision. The second component provides a reward for successfully attaining the objective. A supplementary objective incentive is incorporated in direct proportion to the ultimate alignment of the vehicle with respect to the parking lot.

The dense reward is designed to incentivize progress towards the goal and is comprised of two components, which align with the objectives of reaching the final position and achieving the final orientation. The first component imposes a penalty on the agent-goal distance, denoted as d. The subsequent component penalizes the deviation from alignment with the parking lot. The division of the second term by the distance d is justified, as the effective management of misalignment is relevant primarily in situations where the vehicle is in proximity to the target. The utilization of a Manhattan distance metric appears to be more suitable for the given environment, probably due to the nature of the maneuvering trajectories required to enter parking lots. The dense reward is normalized within the range of -2 to 0.

The allocation of sparse and dense rewards is calibrated using the $c1$, $c2$, and $c3$ coefficients, with the aim of attaining an optimal equilibrium based on empirical evidence.

The PPO algorithm was employed as the DRL backbone method, which is widely recognized as a benchmark for addressing continuous control problems. The optimal values for the training

hyperparameters were determined through empirical analysis or adapted to the machine in use and are presented in Table 18.

Table 18: Hyperparameters for the Unity garage parking experiment

Hyperparameter	Value
Batch size	512
Buffer size	51200
Learning rate schedule	Linear
Learning rate (initial)	1e-4
Time horizon	128

The scene undergoes re-initialization at the start of every training episode. A specific parking lot is chosen as the designated target, and its entrance is made accessible. The agent is then placed in a randomly determined position and orientation within the central corridor. An episode is considered complete under the following conditions: (i) the vehicle successfully reaches the designated goal; (ii) the vehicle encounters an obstacle, although this criterion is not enforced during the initial training phase to allow for continued exploration despite collisions; (iii) the episode reaches the maximum predetermined number of steps without the vehicle reaching the goal.

In evaluating the training phase, the following performance metrics have been considered:

- Cumulative reward: the total amount of reward accumulated over a given period of time or a series of events. The cumulative rewards obtained by the agent during an episode constitute its total reward. The temporal progression of this metric serves as a significant determinant of the efficacy of the training. Rewards are exclusively distributed during the training phase, rendering this particular quantity inapplicable during testing, in contrast to subsequent quantities;

- Goal rate: the ratio of episodes that achieved the desired outcome (i.e., goal reached) to the total number of episodes;

- Collision rate: the ratio of collisions to episodes. As it will be shown, in certain cases, collisions may result in terminal consequences, while in others, they may not;

- Timeout rate: the ratio between episodes ended with a timeout (i.e., doing the maximum number of steps, without reaching the goal) and the total number of episodes.

In the context of our analysis within garage settings, three sub-experiments have been undertaken, delineated as follows.

1. Utilization of a singular target lot across all training episodes, resulting in limited generalizability. A collision does not constitute a terminal event. The vehicle is equipped with a lidar;

2. Replacement of the lidar sensor with a camera and modifications to the NN architecture to accommodate the varying inputs. Specifically, the addition of two convolutional layers is employed to process the camera signal, and the resulting output is concatenated with the other inputs;
3. Utilization of a lidar-equipped vehicle that is trained to navigate towards one of ten parking lots. The specific parking lot is randomly chosen at the start of each episode. The occurrence of a collision is not considered a terminal event until the success rate surpasses a specific threshold, at which point the collision is designated as a terminal event.

In each of the sub-experiments, the testing phase comprises 100 episodes. During each episode, the vehicle is spawned with random position and orientation. The objective is to successfully navigate to one of the lots, randomly selected. Findings are presented in Table 19.

Table 19: Test results of the three garage experiments in Unity

Sub-exp. no.	Sensor	Number of target parking lots in training	Success rate in tests
1	Lidar	1	50%
2	Camera	1	80%
3	Lidar	10	94%

In the context of computational allocation during the training process, it was observed that the utilization of a dedicated GPU did not yield a substantial enhancement in performance, as evidenced by the results obtained in this experiment and corroborated by similar findings in other studies. This phenomenon may be attributed to the relatively shallow depth and limited number of neurons in the NN utilized. In addition, it should be noted that RL places a significant demand on the CPU as a result of the sequential nature of the agent-environment interaction, as depicted in Figure 33. Furthermore, it is worth mentioning that rendering is disabled when the camera is not in use. The recorded training speeds amount to approximately 1.4 million steps per hour. These measurements were obtained using a system comprising an Intel Xeon W2223 processor, 32 GB of RAM, and an NVIDIA Quadro RTX 4000 GPU.

In relation to sub-experiment #1, it has been observed that there exists a crucial phase within the initial 8 million steps. During this phase, the reward obtained is consistently low and unstable, while the lengths of episodes are primarily characterized by collisions initially, followed by timeouts. Specifically, it was possible to observe that in numerous instances, the vehicle consistently performed abrupt gear reversals, remaining predominantly stationary. This phenomenon is indicative of a local minimum, which hinders the agent's ability to enhance its performance. Upon completing 8 million steps, the agent's performance rapidly converges to a stable state characterized by

consistently high rewards and shorter episode lengths. Nevertheless, during the testing phase, wherein the target can be selected from any of the ten lots, it becomes evident that the agent, despite its training, is unable to effectively generalize its knowledge (Table 19).

In the subsequent sub-experiment (#2), the lidar was replaced with a camera. The efficacy of the training was demonstrated. However, it should be noted that the agent's ability to generalize to multiple target test cases is unsatisfactory, as indicated in Table 19. Therefore, it has been attempted to initiate the training process again by randomly simulating all potential target lot scenarios. However, this approach resulted in a deceleration of the learning process and did not yield substantial enhancements. Furthermore, the notion of imposing penalties for timeouts did not yield substantial benefits.

In sub-experiment #3, it has been reverted to employing a lidar sensor as opposed to a camera, resulting in a reduction in both input size and training duration. Additionally, the constants were tuned and the rewards normalized. To ensure the optimal tuning of these hyperparameters, it was imperative to generate and evaluate supplementary Tensorboard visualizations, as depicted in Figure 41. Specifically, the values of each component of the RF (namely, collision, goal, distance, and alignment) have been recorded in Tensorboard to conduct a quantitative analysis of the agent's behavior. Efficiently assessing various alternatives and fine-tuning the RF and simulation

settings is a crucial aspect that demands a significant amount of time and effort.

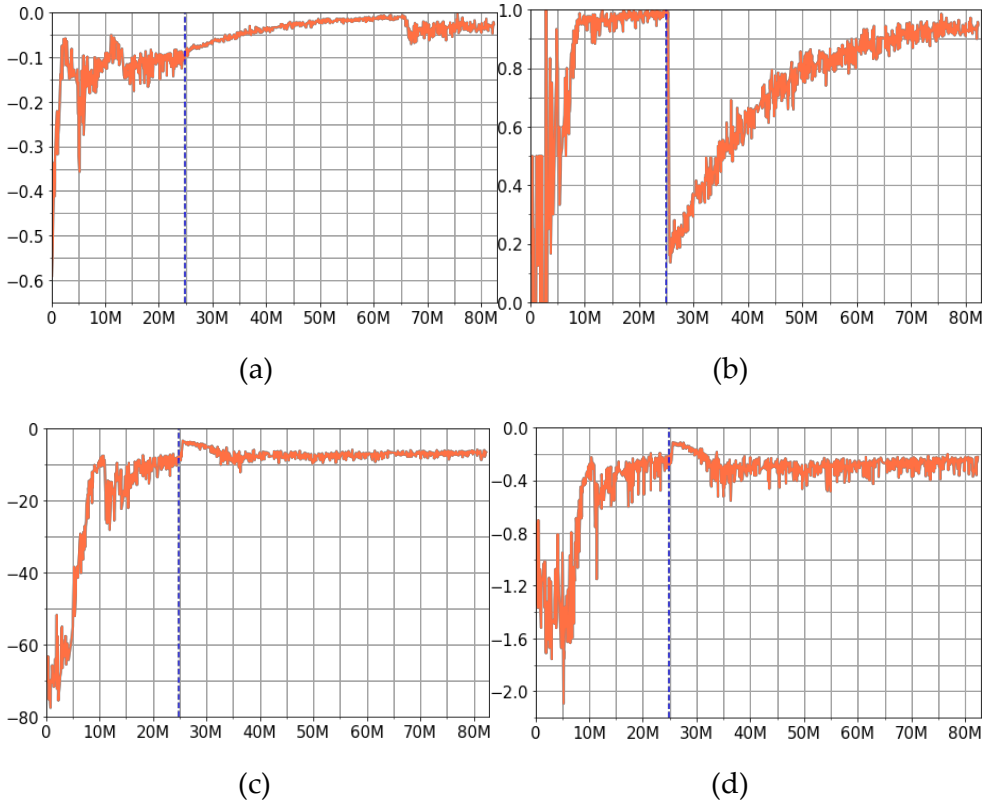


Figure 41: Tensorboard plots of each reward evolution during experiment #3. Rewards: (a) collision, (b) goal, (c) distance, (d) alignment. Dashed blue lines indicate the switch from the first to the second phase

As visible in Figure 41, it was necessary to split the training into two phases, therefore employing the Curriculum Learning (CL) paradigm.

CL in RL poses challenges due to the absence of a preexisting dataset, making it difficult to assess the difficulty of samples using a Difficulty Measurer [215]. In line with the study conducted in [216],

our approach involved the utilization of a one-pass algorithm [217], wherein the difficulty levels were incrementally adjusted to correspond to the progressively intricate iterations of the learning environment. It was also possible to incrementally enhance the agent's abilities in various training phases (e.g., as demonstrated by [218]). However, initial findings indicated that augmenting the training complexity did not yield any advantages. In contrast to the approach taken by [216], our study did not establish a predetermined quantity of steps for each stage. Instead, a performance criterion has been employed, as outlined by [215]. Furthermore, in each phase, not only the intricacy of the context has been heightened (such as the type of objective and the conditions for episode conclusion), but also adjustments to the RF of the agent have been made. These modifications were made based on a meticulous analysis of the Tensorboard charts from multiple training iterations, as previously stated. In the initial phase of this experiment, two distinct stages have been executed. In the initial scenario, the agent undergoes training with the objective of successfully navigating toward a specific designated location. Collisions are subject to penalties but do not result in the termination of the episode. This approach is implemented to facilitate the agent's exploration of the environment without frequent disruptions [219]. Once the objective is attained at a rate surpassing 99.5%, the commencement of the subsequent phase ensues. During this particular stage, the activation of the remaining nine targets occurs randomly. Also, collisions are considered as terminal

events for an episode. Moreover, the penalty associated with collisions is heightened, resulting in a decrease in the reward from -1 to -10. The introduction of these factors is responsible for the significant decline in performance of the goal reward, as evident from the stage transition depicted in Figure 41b. Nevertheless, the training process continues to be effective, albeit at a slower pace, due to the significantly higher intricacy of the learning environment. Subsequently, after a satisfactory number of iterations, the model achieves a desirable level of performance in the designated environment.

The initial phase is concluded in 25M steps, followed by an additional 57M steps required to complete the subsequent stage. Figure 42 illustrates the progression of the success rate, specifically the achievement of goals without any collisions, throughout the training process. Experimental findings indicate that for an agent to reach multiple targets, it is imperative to provide training on multiple targets (Table 19). Conversely, training the agent solely on a single target leads to overfitting, as it becomes overly specialized in locating the precise position of that target.

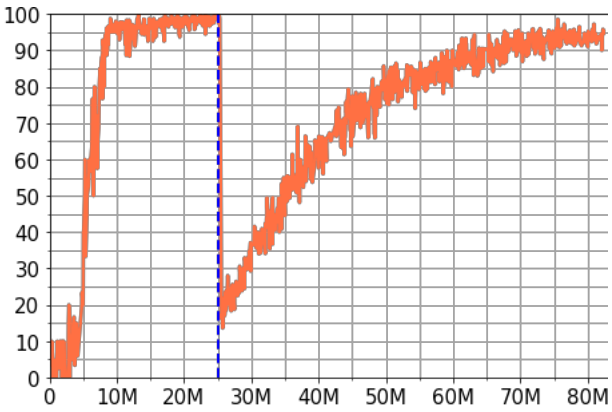


Figure 42: Progression of the success rate in the Unity experiment #3. Transition between phases indicated by a dashed blue line

The impact of CL was evaluated by conducting a comparative analysis of the outcomes achieved by an agent trained from the beginning under identical conditions and rewards as our stage 2 agent. Remarkably, despite undergoing 80 million iterations, the success rate consistently remains at 0. Our argument posits that a crucial determinant of CL lies in the potential occurrence of first episodes involving nonterminal collisions. This not only facilitated the investigation of the surroundings (including exploring the movement area and the consequences of the actions taken) but also exposed the agent to the rewards related to the achievement of the goal. Conversely, an agent which experiences significant penalties from the outset for collisions learns that remaining stationary is more advantageous than navigating through a hazardous environment. Consequently, CL facilitates a learning process that encourages exploration, discourages excessively cautious behaviors, and promotes a more adaptable approach to learning.

The test results obtained from sub-experiment #3 were compared with those of Hybrid A*. Hybrid A* is a modified version of the widely recognized A* search algorithm, specifically designed for the exploration of the three-dimensional kinematic state space of a vehicle. This algorithm ensures that the generated path remains kinematically feasible by incorporating a state-update rule that considers the continuous state of the vehicle within the A* nodes [220], [221].

An open-source Unity implementation of the Hybrid A* algorithm [222] has been used to conduct our experimentation within our designated test environment. The outcomes of our experimentation are outlined in Table 20. To generate the path, Hybrid A* employs a discretization technique that divides the map into square cells of 25 cm on each side. Each individual cell within the map is considered a node, and throughout the process of generating a path, each node is assigned a corresponding cost. The heuristic function takes into consideration the potential movements of the vehicle, taking into account a distinct set of actions defined by three steering angles: -30°, 0°, and 30°. In order to prevent collisions, a flow field is formed around the obstacles. The manipulation of the safety margin allows for regulation of the extent of its expansion. Findings indicate that Hybrid A* demonstrates a success rate of 100%, which subsequently decreases to 85% during the path following phase due to the discretization of the map. The computational time required to compute the path, which is

irrelevant to the DRL agent, exceeds 0.8 seconds, as observed in the performance evaluation conducted on an Intel Xeon W2223 machine. In order to enhance the real-time performance of Hybrid A* algorithm, it is necessary to augment the cell size to 1.2 meters. However, this adjustment resulted in a decrease in the target reach rate to 21%. Table 20 additionally demonstrates that the DRL agent has the capacity to decrease the average number of gear inversions required to attain the aim by 25%. Therefore, the DRL agent demonstrates superior performance in terms of target attainment rate, gear inversion rate, and latency, while also eliminating the requirement for a map.

Table 20: 100 episodes comparison between Hybrid A* and DRL PPO – Garage environment

Path planning algorithm	**Success rate**	**Gear inversions per episode**	**Time for path planning**
Proposed DRL	94%	3.36	-
Hybrid A*	85%	4.48	0.857 s

Our experience highlights the significance of certain aspects in establishing a suitable environment and mitigating the occurrence of systematic errors, which can be challenging to detect.

The time interval between consecutive updates of the environment, known as the simulation tick, must be sufficiently rapid to ensure an accurate physics simulation. Failure to meet this

requirement may result in delayed or missed detection of collisions and/or goal attainment. However, the specific tuning of the simulation tick should be based on the real-time speed of the simulated stuff and the computational capacity of the CPU.

Another parameter that requires meticulous adjustment is the decision period, which refers to the interval between each agent's action selection. A short decision period results in the vehicle remaining in a standstill state. The present analysis posits that, within this context, the agent lacks the temporal capacity to observe the consequences of its actions. When the decision period is very long, conversely, the agent's responsiveness diminishes, resulting in inadequate management of the vehicle's dynamics.

To enhance the efficiency of the training phase, the capability provided by ML-Agents to concurrently execute several training scenarios has been leveraged. This results in an elevated steps-per-second ratio, albeit with a concomitant rise in CPU workload. Due to this rationale, it is advisable to avoid excessively high numbers of parallel environments, as the step/second ratio experiences a decline beyond a particular threshold that is specific to the machine in use.

The selection of the sensor(s) significantly influences the learning and behavioral characteristics of the agent, as well as the duration of the training process. The choice between a camera sensor and lidar carries significant implications, as the camera necessitates scene

rendering at each timestep. This process reduces the number of instructions executed per second, hence prolonging the training time. In addition, the use of two convolutional layers in the neural network architecture for processing the camera information necessitates a higher network size, hence influencing the duration of the training process. Given the comparable outcomes obtained from both the camera and lidar in our studies, it has been made the decision to accord higher priority to the latter. In relation to the concept of generalization, it has been observed that there exist distinct limitations. Specifically, when alterations are made to the configuration, such as modifications in the width or length of the driving area or adjustments in the positioning of the parking lots, a notable decline in performance occurs. Consequently, it becomes necessary to fine-tune the agent in order to address this issue.

5.2.3.2 Random obstacles environment

The preceding experiment involved the training of an agent to successfully navigate to a designated parking lot, starting from a randomly selected location within a garage. However, it is important to consider the potential existence of unforeseen impediments that may be present prior to reaching the intended destination. In order to conduct an analysis, a separate Unity environment was established, and its examination is presented in this sub-section.

The input type, neural network setup, and RL training method used in this experiment are identical to those employed in Garage experiment #3. The objective of the agent is to navigate through an 80 × 80 m area, with the aim of reaching a position that is randomly determined. This task involves avoiding a variable number of static obstacles, which are randomly positioned and oriented between the starting point and the destination point, as depicted in Figure 43.

Figure 43: Sample random obstacles Unity environments, with random positions and orientations

The RF is built upon a methodology akin to that utilized for the garage parking task, with the inclusion of an additional penalty for low-speed behavior. The decision to implement this modification was made to address observed experimental scenarios wherein the agent would cease its movement in close proximity to the target without actually reaching it. This behavior appeared to indicate a sense of satisfaction with the achieved distance, maybe in comparison to the perceived risk of a collision.

During the training phase, a one-step CL process was employed, similar to the approach utilized in the initial experiment. The level of difficulty was systematically increased by introducing a greater number of obstacles inside the visual scene. After successfully achieving proficiency in a certain difficulty level, the agent's training progresses to more intricate environments. The levels of difficulty are documented in Table 21. Adjustments to hyperparameters, such as the RF, may also occur during these transitions between levels, guided by the agent's behavioral observations.

Table 21: Difficulty levels of the random obstacle environment

Difficulty level	Description
1	No obstacles
2	One obstacle midway between the spawning point and the target
3	Like level 2 but with two additional obstacles on the sides
4	Like level 3 but an additional obstacle is placed in a second series of obstacles
5	Like level 3 but two additional obstacles are placed in the second series of obstacles
6	Like level 3 but three additional obstacles are placed in the second series of obstacles

Figure 44 displays the progression of the training success rate across various difficulty levels. In relation to the initial three levels, it is seen that the initial level is attained after a total of 11 million steps, followed by the intermediate level after an additional 36.5 million steps, and ultimately, the third level is achieved after an additional 363.5 million steps. The observed decrease in performance with each level transition indicates a deficiency in the agent's capacity for generalization. Indeed, it is observed that the agent demonstrates proficiency in navigating toward a designated target while sticking to the existing obstacle framework at each phase. However, it does not exhibit the same level of competence in reaching a target without encountering any collisions. Nevertheless, it is seen that as the difficulty levels increase, there is a decrease in the drop, indicating that the agent finally acquires the ability to generalize. It is evident that each level transition necessitates specialized training, a process that the agent gradually and perceptibly accomplishes.

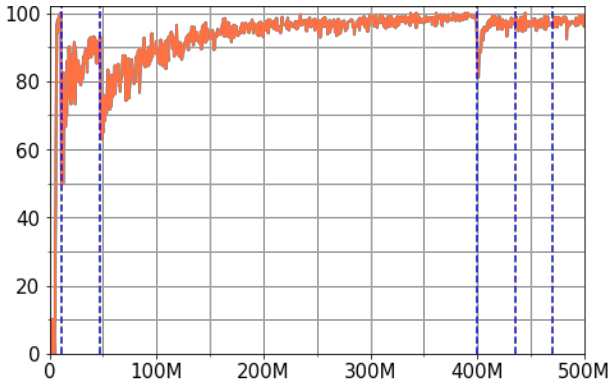


Figure 44: Progression of the success rate in the Unity random obstacles experiment. The transition between phases is indicated by dashed blue lines

Figure 45 illustrates the performance of a system that underwent direct training at the ultimate difficulty level, as described in Table 21. In this scenario, in contrast to garage parking, the agent that underwent training without CL demonstrates a noteworthy level of performance, but with a discernible disparity compared to CL (95% versus 98%). It is suggested that the dissimilarity shown in the garage case study can be ascribed to the increased spacing between obstacles inside this subsequent environment. While collisions are terminal events since the beginning of the training process, they are not overly common. Therefore, the agent categorized as a "novice" is still able to navigate and familiarize itself with the environment, ultimately acquiring knowledge of the task at hand. A comparison with Figure 44 reveals that the agent commences its learning process at a later stage due to the greater complexity of the environment. This delay in learning could be an issue in situations when prompt feedback is required, such as when exploring various hyperparameters on hardware that is shared or requires payment upon usage.

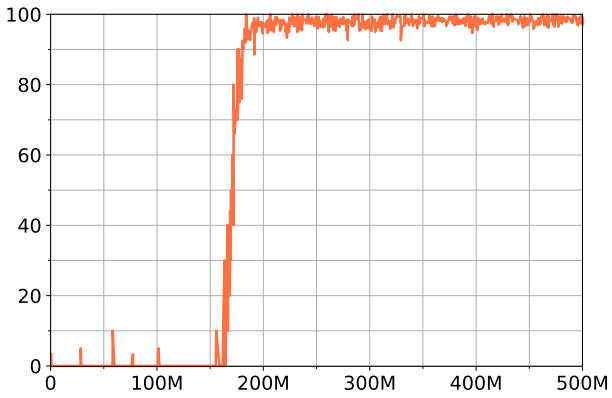


Figure 45: Success rate over training for an agent trained without CL

In addition, for the purposes of this experiment, a comparative analysis between our DRL agent and a Hybrid A* implementation is provided. The Hybrid A* algorithm demonstrates a slightly superior performance compared to our DRL agent in terms of goal reach rate, with a success rate of 99% as opposed to the DRL agent's 98%. Our contention is that this can be attributed to the reduced intricacy of the maneuvers required to attain the desired objective. Nevertheless, achieving a decrease in latency from 0.56 seconds to 0.17 seconds necessitates an increase in the size of the cell side from 0.5 meters to 1.2 meters, resulting in a decline in the success rate to 69%.

Table 22: 100 episodes comparison between Hybrid A* and DRL PPO – Random obstacles environment

Path planning algorithm	**Success rate**	**Time for path planning**
Proposed DRL	98%	-
Hybrid A*	99%	0.561 s

Also in this scenario, the DRL agent demonstrates superior performance in terms of latency and similar success rates compared to Hybrid A* while also not needing an a priori map.

5.3 CARLA experiment

The experiment involves a parking case-study in the CARLA simulator, which is becoming very popular in the automotive domain for the simulation of ADFs.

5.3.1 Experiment setup

Compared to the previous case in Unity, the main aim of this implementation is to enhance the level of realism, to obtain a model that is closer to real-world vehicles. In order to enhance the degree of realism in graphics and physics modeling, the choice for a driving simulator was CARLA [25], an open-source software designed specifically for the advancement of ADFs. The system encompasses a variety of town maps and diverse vehicle models that may accommodate various sensors, including radar, lidar, camera, and others. Furthermore, CARLA provides an API that grants users the ability to manipulate several elements pertaining to the simulation, such as traffic patterns, pedestrian actions, weather conditions, sensor functionality, and more. The official section pertaining to DRL lacks recent updates; however, there have been recent proposals [223], [224] that provide promising outcomes in the context of trajectory tracking applications.

Commencing with the pre-existing Town 5, which includes a parking space, extraneous items have been eliminated from the environment. This was done to create a streamlined scenario that

minimizes the computational burden of rendering and thereby reduces training time. The parking area, depicted in Figure 46, has dimensions of 50 x 50 m. It consists of a total of 60 comb parking places, each measuring 5 x 2.5 m, arranged in a 90-degree orientation. The designated training area is enclosed by brick walls, effectively creating a barrier that serves to protect the agent from experiencing falls. The weather encompasses a total of 15 distinct atmospheric situations, including variations of sunlight, rainfall, fog, and cloud cover. In addition, there exists a diverse range of 41 vehicle models, spanning from motorcycles to trucks, each offering the option for personalized color customization.

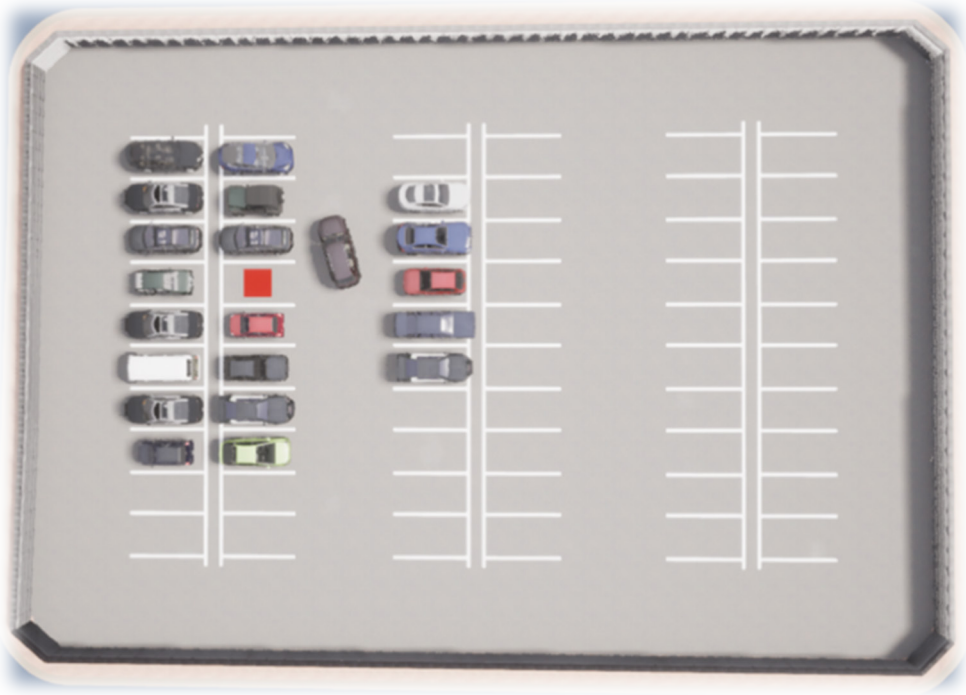


Figure 46: The CARLA parking environment. The targeted parking lot is indicated with a red square

In order to integrate our DRL agent, the Gymnasium toolkit, previously referred to as OpenAI Gym [225], has been employed. Gymnasium is a Python toolkit that provides an interface for creating and implementing DRL environments. It has gained widespread recognition as the preferred method for facilitating communication between agents and simulation environments. Furthermore, the Gymnasium framework offers the capability to construct customized settings. Consequently, the CARLA parking scenario has been encapsulated within a Gymnasium environment to obtain a DRL-compatible interface. In addition, it should be noted that Gymnasium is fully compatible with Stable-Baselines3 (SB3) [226], which is a Python library that is openly available and provides a diverse range of DRL algorithms. This compatibility allows for the seamless integration of Gymnasium with SB3, enabling users to efficiently establish the training method as well as the NN architecture and configuration in a straightforward and simple manner. Figure 47 displays the diagram illustrating the tools employed and their corresponding interfaces.

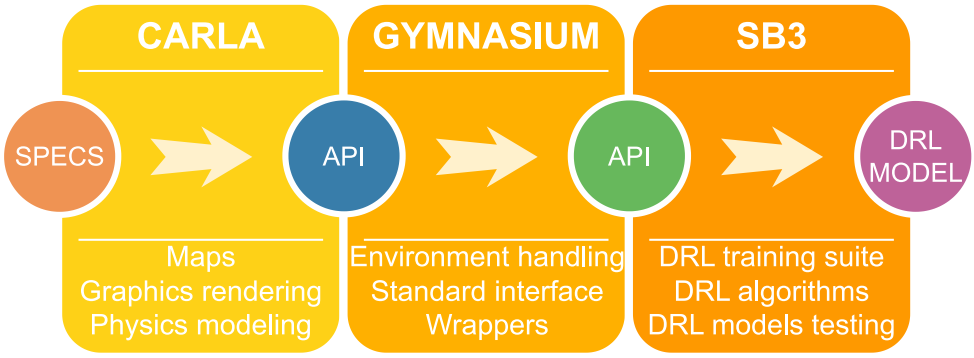


Figure 47: The toolchain implemented in CARLA: from behavior specifications to DRL model

In each episode, the initial position and orientation of the car are randomly determined within the drivable region. The objective to be achieved is reaching one of the 60 lots, which are picked randomly in each episode. In order to provide training for our agent, the Jeep Wrangler vehicle model has been used, which was accessible within the CARLA simulation environment. The dimensions of the car are 4.80 × 1.90 × 1.90 m, and it possesses a total mass of 2206 kg. CARLA models the vehicle dynamics using NVIDIA PhysX [227], a widely used physics simulation engine. The EV (i.e., the agent) is equipped with a lidar sensor positioned at the vehicle's midpoint, offering a panoramic horizontal field of view spanning 360 degrees.

Table 23 provides a comprehensive summary of the constituent elements of the RF. The two primary factors that influence the EV's approach towards the goal are the distance and heading rewards. Collision is sparse, but it is essential to instruct the agent on how to avoid other Non-Player Vehicles (NPVs) and the boundaries of the

designated area. There are two key factors that are associated with the successful attainment of the designated parking lot: goal and alignment. The goal reward is granted upon the arrival of the EV to the parking lot, whereas alignment compensates the agent for successfully parking in a well-aligned manner. This component is maximum if the car is perfectly aligned (0 degrees difference from the parking orientation) and decreases linearly to 0 depending on how crookedly the agent is parked (where 0 indicates parking with 90-degree difference from the parking orientation).

Table 23: CARLA experiment RF components

Name	Description	Range	Type
Distance	Euclidean distance from EV to target	[-0.1, 0]	dense
Heading	Angle between EV forward vector and EV-target vector	[-0.1, 0]	dense
Collision	Collision with walls or NPVs event	[-1, 0]	sparse
Goal	Target achieved event	[0, 15]	sparse
Alignment	Angle between EV forward vector and parking lot orientation	[0, 10]	sparse

In order to be compliant with the previous Unity experiment, the PPO algorithm [208] was employed for training the model. The chosen architecture for the primary NN is an MLP configuration, consisting of two hidden layers each containing 512 neurons. The input and output values of this configuration are presented in Table 24. In order to enhance the agent's perception of its motion, at each network update the most recent four input values have been stacked on top of the current input value. This enables the agent to effectively capture the evolving dynamics and fluctuations in the environment over time.

Table 24: PPO network setup for the CARLA experiment

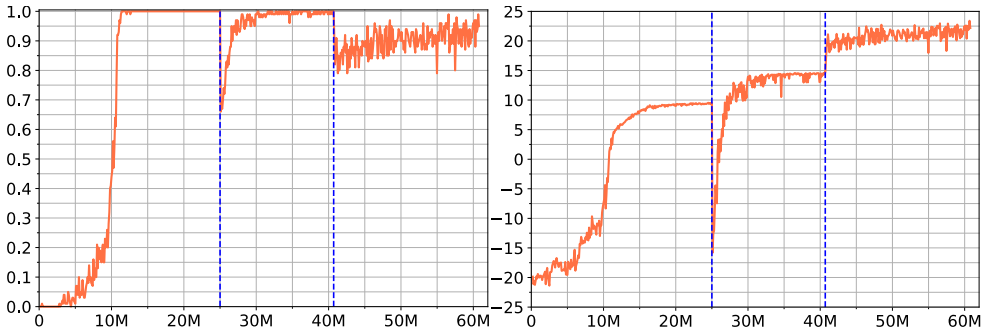
Type	Quantity	Dimension	Resulting size
Input	Lidar data	61	340 (5 stacked inputs)
	Distance from target (x, y)	2	
	Speed (x, y)	2	
	Acceleration (x, y)	2	
	Angular velocity (yaw)	1	
Output	Throttle	1	4
	Steering	1	
	Brake	1	
	Reverse	1	

Based on the above findings in the Unity environment, albeit of less complexity, a similar CL technique has been implemented for the purpose of training the agent. In this scenario, the training process is divided into three stages. In the initial stage, the analysis just focuses on the EV, no NPVs are present in the scene. An episode concludes either when the goal is achieved or when it exceeds a maximum of 240 steps (corresponding to 4 minutes in the real world). The absence of penalties for collisions serves as an incentive for the agent to engage in exploratory behavior inside the environment. During the second phase, the selection of the number of NPVs is performed randomly within the range of 0 to 20 for each episode. Similar to the previous phase, the episode can only terminate upon meeting the target or when a timeout occurs. Collisions are now subject to a minor penalty, quantified by a weight of -0.1. In the third phase, the conditions are comparable to those in phase #2, with the exception that an episode is now terminated by a collision, which incurs a penalty weight of -1.

5.3.2 Results

The PPO agent underwent training for approximately 60 million steps, as depicted in Figure 48. In the initial CL phase, the agent successfully achieved the goal in all instances after a total of 25 million steps, as depicted in Figure 48a. Once this is attained in the absence of NPVs within the given context, the subsequent phase begins, spanning approximately 15 million steps. During this phase, the agent acquires the skill of parking while NPVs are positioned near the designated

parking area. The last stage, when collisions denote a terminal state, persisted for a duration of 20 million steps, resulting in a concluding success rate of 97%.



(a) (b)

Figure 48: The training of the PPO agent in CARLA over around 60M steps. (a) is the success rate, whereas (b) represents the cumulative reward. The three CL phases are distinguished by dashed blue lines

While the overall success rate is deemed satisfactory, certain issues pertaining to the utilization of CARLA emerged during the process of code development. Specifically, the absence of explicit assistance for DRL posed challenges in terms of implementing the environment and ensuring compatibility with Gymnasium and SB3. Overkill values of the CARLA environment update frequency, exceeding 20 Hz, significantly prolonged the training duration due to increased computational demands. Conversely, frequencies below 10 Hz failed to ensure adequate observation of the environment by the agent, resulting in instances where the agent inadvertently reached the goal without being detected. Furthermore, it was again crucial to implement a decision period, as excessively high network update

frequencies (exceeding 10 Hz) hindered the agent's ability to acquire a knowledge of locomotion. This limitation arose from the fact that a brief accelerator pedal pressure did not provide adequate duration for initiating movement. An optimal compromise that was identified involves utilizing a CARLA environment update rate of 20 Hz in conjunction with a network update rate of 5 Hz, resulting in a decision period of 4. However, it is important to note that this determination was made through extensive experimentation and iterative refinement.

It is worth remarking that, for this experiment, test results have been obtained using Edgine. First, the obtained NN is uploaded to Measurify through its dedicated GUI Graphical User Interface [228]. Successively, an Edgine instance is executed on a Ubuntu PC (since the model has been trained using Ubuntu) to automatically download the NN weights from the cloud. Once the model is obtained, a dedicated script allows to perform inference. Since the model described has been trained using the Gymnasium toolkit, a specific Edgine operation has been defined which enables it to execute the Python snippet dedicated to launch the CARLA server, load the model, and run inference in the parking environment. Once results are obtained, they are returned to the Edgine instance which automatically ships them to Measurify using the *send()* operation.

At the moment of writing, this Edgine feature results still working progress, but the results achieved so far are encouraging and reveal other potential that Edgine can offer in the field of edge computing.

5.4 Unity or CARLA?

In this part, we undertake a comparative analysis of our development experience in both Unity and CARLA simulators.

Unity is a versatile graphic engine utilized across multiple industries. It has the capability to adjust simulation environments and settings and provides a range of readily integrable vehicle sensors like lidars, radars, cameras, and semantic cameras. Additionally, the process of generating other observations is uncomplicated. The utilization of a GUI enhances the user experience by facilitating a pleasant and efficient interaction, eliminating the necessity to delve into the underlying source code. A Unity simulation necessitates the computation of physics and graphics, particularly in cases when the agent's input incorporates a camera, hence resulting in a significantly elevated processing need. The aforementioned concern, which is applicable to all environments, can be effectively addressed through the utilization of Unity's capability to instantiate multiple environments within parallel processes and/or execute multiple scenes concurrently. This approach optimizes training time by fully leveraging the CPU's cores. One significant constraint is the limited range of selectable NN topologies and tunable parameters. For

instance, only dense and convolutional neural networks are available, without including advanced mechanisms like attention [13] that are currently considered state-of-the-art. Unity may be accessed through the ML-Agents Python library, which provides also an interface for Unity as a Gymnasium environment. This enables the utilization of many neural network architectures, including bespoke ones [229]. Nevertheless, the available material is scarce, and there is a lack of scholarly publications on this topic in the existing literature.

The CARLA platform is an open-source tool designed for the simulation and evaluation of ADFs. The platform offers a considerable level of adaptability in relation to the customization and management of the physical aspects of the environment, facilitating the creation of intricate situations. This feature proves to be highly advantageous for conducting experiments in the field of DRL. CARLA offers pre-established maps and tools, such as the Scenario Runner [230] and Map Editor [231], to facilitate the creation of simulations in settings that are either extremely realistic or tailored to specific requirements. Additionally, the software showcases a noteworthy assortment of meticulously replicated vehicle models, encompassing various metrics such as maximum RPM, the moment of inertia pertaining to the engine, the duration required for gear shifting, and the drag coefficient associated with the vehicle's chassis, among others. This achievement is facilitated by the utilization of the Unreal Engine game engine [232], which offers advanced capabilities in terms of generating

three-dimensional graphics and conducting physics simulations. CARLA utilizes a customized version of Unreal Engine 4.26, incorporating tailored modifications exclusive to CARLA's functionalities, necessitating robust computational resources. The dynamic models of vehicles are derived from NVIDIA PhysX, the established Unreal Engine 4 vehicle model [227]. The learning curve of the tool is rather steep. The training process is influenced by the overall sophistication and complexity of the environment. This influence is observed through various aspects, including the type of vehicle, the observations made, and most notably, the timing of the simulation world's progress and the agent's decision.

In summary, the selection of the simulation framework is contingent upon the requirements and objectives of the research endeavor. If the primary focus is on realism and achieving a high level of physical accuracy, CARLA would be the optimal selection. However, the Unity ML-Agents toolbox presents a potential compromise between the accuracy of virtual reality depiction and the effectiveness of modeling, training, and simulation.

6

DRL models explainability

The primary issue with employing ML algorithms is that they operate as black-box methods, which makes it challenging to identify patterns attributed to the decisions made by the agent. This problem becomes more critical when considering the automotive industry, where enforcing very high safety standards is paramount to safeguard human lives. The chapter presents an examination of SHAP values and a comparison with attention layer outputs from a DRL neural network for a highway use case.

First, the development environment and DRL model are introduced. Next, experimental results are presented and analyzed in three modes: (i) episode timeline analysis, (ii) frame-by-frame analysis, and (iii) aggregate statistical analysis.

6.1 Environment

Being the scope of this research investigating the explainability of DRL models, it has been decided to adopt a simple DRL algorithm and a basic learning environment, so to avoid any possible interference caused by the complexity of the DRL algorithm or the development environment. To accomplish the outlined research objectives, a DRL agent was trained using a basic DQN algorithm within the highway-env environment [21]. Highway-env is a collection of simple 2D bird-eye view environments for training RL agents in AD tasks, particularly in highway settings. Highway-env is one of the environments provided within Gymnasium. The open-source code base was altered to extract the necessary data for analysis from each episode, including the status log, action log, attention matrix values, and Q matrix values.

The architecture of the agent model is centered on a single EV-attention layer, which is designed to capture the interdependence between the EV and the other vehicles. The layer receives a single vector input for each vehicle. This input is generated by embedding the feature list of each vehicle using a linear dense layer encoder with 64 units. The final output is produced by a decoder consisting of two dense layers (Figure 49, [149]). A single EV-attention head is employed in order to maintain simplicity and accommodate the unidirectional nature of the highway environment.

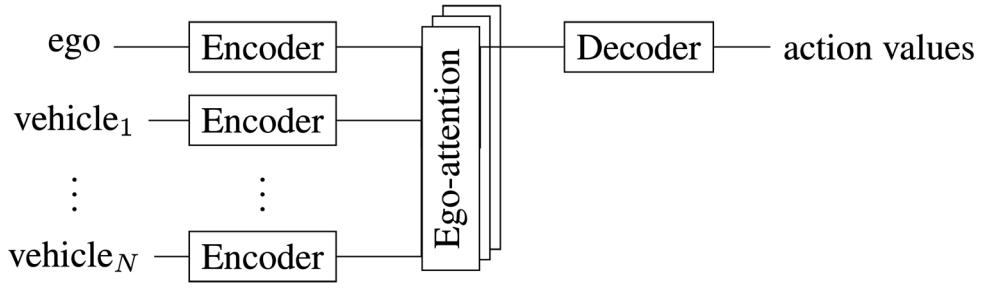


Figure 49: Highway DRL model architecture. Picture courtesy of [149]

The analysis was conducted on a standard 3-lane highway-env environment, utilizing its default values with the exception of certain parameters. These parameters include a traffic density of 0.6 (medium), an observation of 8 vehicles to mitigate computational complexity and reduce training times, a policy frequency of 1 Hz to allow the agent sufficient time to observe the consequences of its decisions, and an episode duration of 80 frames (equivalent to 80 seconds) to adequately assess agent behavior. In the highway-env framework, the vehicles under observation are arranged in a sequential manner based on their distance from the EV. Specifically, the EV is denoted as *v0*, while *v1* represents the vehicle closest to the EV, *v2* refers to the second closest vehicle, and so on. Hence, it can be observed that there is no distinct identity assigned to each vehicle for every episode, apart from the EV. However, cars are identifiable on a per-frame basis. Each episode encompassed a collective sum of 15 vehicles, constituting the predetermined standard value. Every observation sent to the DRL model has seven distinct features for each vehicle. These features include the presence of a vehicle in the current

frame, as the observation region around the EV is constrained. Additionally, the features consist of the vehicle's x and y coordinates, longitudinal and lateral speed, as well as the two trigonometric directions. The observations are normalized with a longitudinal distance of 100 meters and a speed of 80 meters per second. Each lane has a width of 3.5 meters and serves as a reference point for both the EV, where its x-coordinate is always 0, and the other cars, where their positions are measured relative to the EV.

The velocity of the EV can be categorized into three distinct levels: 20, 25, and 30 m/s. These levels are slightly more than the average velocity of other vehicles, which typically travel at approximately 22 m/s. The agent can select one of five actions, namely *right*, *left*, *idle*, *faster,* or *slower*, at each frame. The system proceeds to process the determined activities while considering the relevant context. As an example, the deceleration at a velocity of 20 m/s leads to an *idle* action, and the same principle applies to the left action when the EV is already positioned in the third lane. To elucidate this point, we shall henceforth make a distinction between action, which refers to the decision made by the model, and real action, which pertains to the action executed by the EV while considering the constraints imposed by the context.

The agent underwent training using a dense reward of 0-1 for speed and a sparse reward of -2 for collisions, which served as the factor for the premature conclusion of an episode. To enhance

simplicity, we have excluded the right lane reward from the analysis, as the initial model does not impose any penalties for overtaking on the right side.

Following the completion of the training process, the ultimate success rate stands at 89%. Additionally, an average distance of 2.3 km was driven every episode. In our empirical observations, it was generally observed that training effectiveness is limited to the initial phase, specifically between 400 and 1,200 episodes. Subsequently, performance deterioration occurs without any notable recovery, even after approximately 35,000 episodes, which corresponds to a training duration of 48 hours on an NVIDIA DGX system. The phenomenon described in the literature is sometimes referred to as catastrophic forgetting [233]. Extended training durations have been observed to result in more caution exhibited by the agent, potentially attributable to the occurrence of accidents arising from unfavorable behaviors exhibited by other vehicles.

Once an agent has undergone adequate training, it becomes suitable for analysis. SHAP values are derived by executing a specific quantity of episodes and training the model using the SHAP Python module. Based on the SHAP documentation and SHAP library [234], a DeepExplainer has been used and provided with the agent's DQN model (the value network) and the observations from a set of 20 training episodes (equivalent to 1,600 samples). This approach aims to generate accurate estimations of the SHAP values for each input

feature. After being trained, the SHAP model can be provided with test values in order to generate estimations. The computation of SHAP values occurs at each frame in the context of a DRL system. These values are calculated based on the value network of the DRL, which provides the Q value for every available action. The Q value represents the anticipated cumulative reward when the agent is in the observed state and executes the action, subsequently continuing to play until the episode concludes according to a specific policy π. Hence, SHAP values are available for every potential action, while this study primarily concentrates on values pertaining to the chosen action only. Despite the algorithm's exponential complexity [14], the execution times of SHAP were rather short, typically on the scale of a few seconds.

The attention values are derived from the output of the attention layer, which represents a probability distribution over cars. Therefore, the aggregate always equals 1, and higher maximum attention values signify a concentration of attention on a particular vehicle, whereas lower maximum attention values suggest a dispersion of attention across multiple vehicles.

A necessary modification is required when comparing attention and SHAP, as attention values are assigned on a per-vehicle basis, but SHAP values are assigned on a per-feature basis. Upon reviewing less complex issues, such as [17], it has been deemed a suitable

approximation to establish the SHAP value of a vehicle as the SHAP value of its most significant attribute.

$$SHAP(V_i) = \max_{f_i \in \{features\ of\ V_i\}} \{SHAP(f_i)\} \quad (10)$$

The per-vehicle SHAP values are subsequently transformed into probabilities using the softmax function. Max SHAP Vehicle (MSV) refers to the vehicle within a certain frame that possesses the greatest per-vehicle SHAP value, while Max Attention Vehicle (MAV) refers to the vehicle that possesses the highest attention value.

6.2 Experimental results

The analysis was structured on three primary perspectives. One aspect to be considered is the timeline of each individual episode. Subsequently, an examination is conducted on the SHAP and attention values pertaining to each decision frame. The final step involves calculating statistical measures by aggregating many episodes. Consequently, a Python Jupyter notebook has been constructed for the purpose of conducting the analysis. To enhance the clarity of the presentation, the three aspects are designated as episode view, frame view, and aggregated view, respectively. The analysis will be presented in a sequential manner, with the aim of simplifying the presentation. However, it is crucial to note that many of the considerations presented are the result of an iterative synergy between the three views.

The complete analysis, along with the corresponding source code, can be accessed at the following URL: https://github.com/Elios-Lab/explain-drl-highway.

6.2.1 Episode view

The initial stage of the analysis involves examining the progression of one or more episodes on a chronological scale, referred to as the episode view. As an illustrative example, Figure 50 presents a subset of the most pertinent variables observed in episode 40, which has been carefully selected to serve as a notable instance. The perspective presented exhibits certain attributes that are considered significant for enabling the analysis, such as action, actual action, and ego lane. These attributes are then graphically represented by plotting their respective values along the timeline. The display also includes the MAV and MSV vehicles. Vehicles are identified in a sequential manner, frame by frame, based on their relative distances. In this context, *v1* represents the vehicle in closest proximity to the EV, *v2* denotes the second closest vehicle, and so forth. The term *v0* represents the EV. The classification of the most significant feature (referred to as the feature with the highest SHAP value) is also provided. Based on the established highway-env convention, the lanes are assigned numerical values ranging from 0 to 2, with the numbering progressing from left to right. The features are assigned numerical values in the following manner: 0 - presence, 1 - longitudinal distance (x) from the EV, 2 - lateral

distance (y), 3 - relative longitudinal velocity, 4 - relative lateral velocity, and 5 and 6 - trigonometric headings.

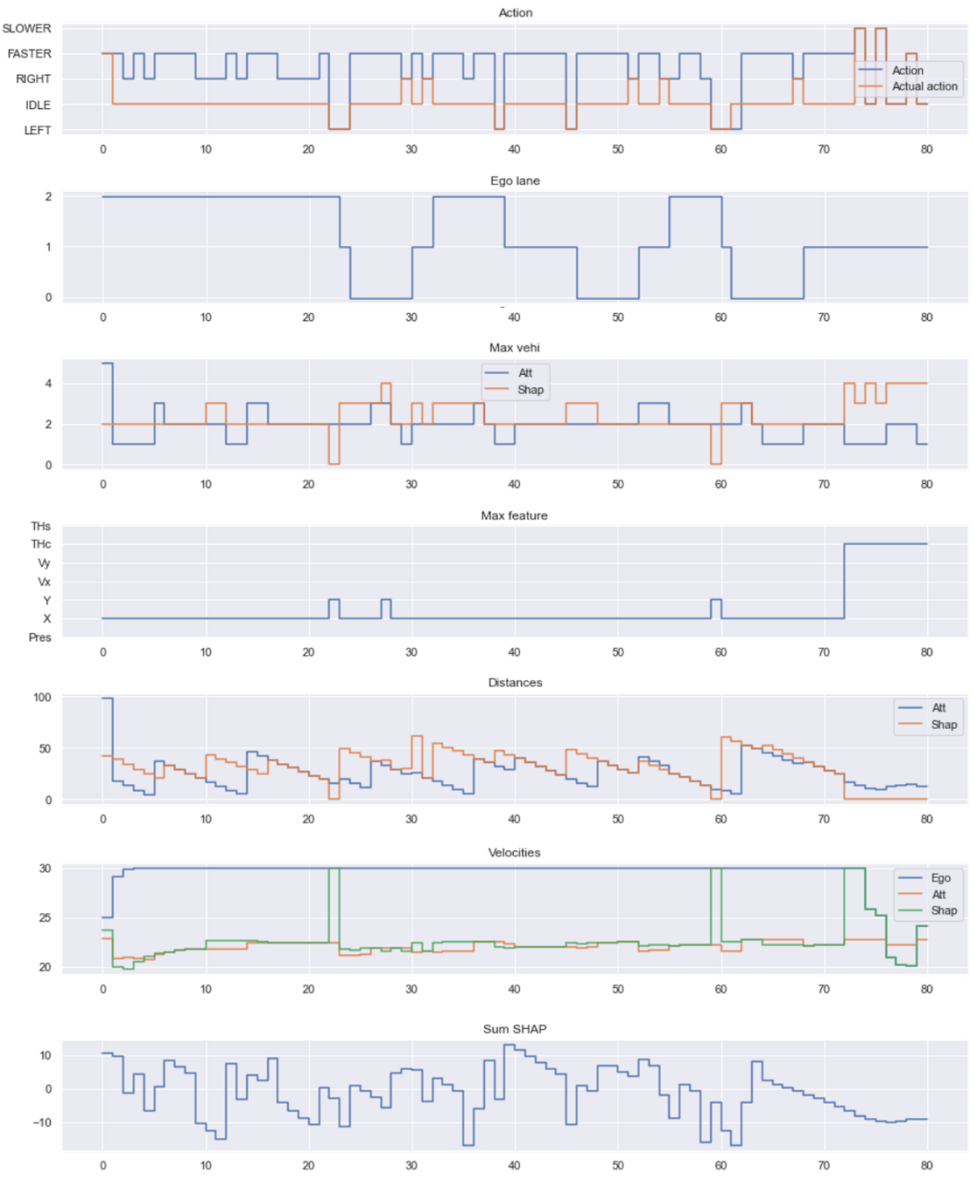


Figure 50: Episode View of episode nr. 40. The "Max vehi" chart represents the MAV and MSV timelines [235]

Upon observing the lane plot, it is evident that the EV initiates its trajectory in the rightmost lane, subsequently transitions to the leftmost lane at approximately frame 24, and subsequently reverts to the rightmost lane shortly after frame 30. Two additional instances of this pattern, resulting from a double simultaneous overtaking, are observed. Subsequently, the EV concludes the episode by proceeding along the central lane. In the present episode, it is evident that the highest attention is never on the EV, in contrast to the maximum SHAP. Furthermore, a distinct disparity can be observed between MAV and MSV, as the MAV tends to be in closer proximity to the EV compared to the MSV. It may therefore be inferred that attention is of a broader nature, whereas SHAP is more focused on the decision. There exist only three instances in which the MAV is $v3$ and the MSV is $v2$, indicating an MAV farther away and an MSV in closer proximity to the EV.

Considering the max (i.e., most important) feature diagram, it appears that it is almost always the longitudinal distance from the EV, the lane position (in these frames the MSV is the EV), and, in the episode's last frames (72-79), the trigonometric heading (in one case, lane position), when the MSV is v4 or v3. Upon careful examination of the timelines depicted in Figure 50, it becomes evident that the concluding frames exhibit a distinctive pattern characterized by a series of alternating deceleration and acceleration events. Notably, the max feature is the trigonometric heading of the lowest order absent

vehicle (specifically, *v3* or *v4*). This topic warrants further investigation in the following. It is also observed that the agent consistently maintains the highest speed throughout its trajectory, with the exception of these final frames.

MSV is the EV exclusively in frames 22 and 59. This phenomenon is characterized by the occurrence of two overtaking maneuvers in the left lane, with the most prevalent behavior being *idle*.

Upon examining the chart depicting the distance between the distance from EV of the MAV and MSV, it becomes evident that attention is predominantly paid to the EV. Additionally, there exists a "step & staircase" pattern, characterized by attention steps occurring near EV overtakes, which can be identified by the longitudinal distance of *v1* approaching zero. Conversely, MSV tends to transition abruptly, or "jump," to a vehicle located further ahead prior to the overtaking event. In certain instances, the execution of the "step" is postponed until the overtaking maneuver is initiated. The observed pattern indicates that the decision to change lanes, whether it involves overtaking or returning to the right lane, is typically influenced by a reference vehicle that is positioned at a greater distance from the EV. This suggests that the decision-making process is not significantly influenced by the nearest vehicles, as they are already in the process of being overtaken, i.e., the agent has completed the processing of said vehicles and has subsequently shifted its attention towards a broader perspective.

Except for the aforementioned cases, the MAV is on the closest or second closest vehicle. Conversely, the MSV exhibits a greater range of possibilities, encompassing the EV as well as possessing a longer line of sight, but it is never on the closest vehicle. The concept of SHAP pertains to the ultimate determination made by the NN, whereas attention operates at a shallower layer (Figure 49). This suggests that SHAP exhibits a form of predictive capability in decision-making, a trait commonly associated with skilled drivers [236]. Conversely, attention is primarily focused on nearby vehicles in order to prevent unexpected incidents, such as infrequent spontaneous lane changes by certain vehicles in the highway-env model, which have the potential to cause accidents. In certain instances, there is an occurrence of convergence between maximum attention and SHAP within the frames. This phenomenon occurs as a result of the MAV "approaching" the MSV, i.e., the EV gets closer to the MSV and causes it to also become the MAV. However, the MSV undergoes a rapid advancement to a vehicle that is more distant, thus representing a "delayed step" in the aforementioned pattern.

Finally, the cumulative sum of SHAP values within a given frame serves as a noteworthy indicator of the anticipated advantages resulting from the executed action. Jumps in this value follow a successful lane change decision, such as initiating or completing an overtaking maneuver. Additionally, a relatively minor decrease in the value occurs when new vehicles approach or remain near the EV.

The interpretation of the above-described final frames, which tend to occur in other episodes as well, required a more in-depth investigation, which spurred us to implement a second display modality, namely the frame view, that we describe in the following.

6.2.2 Frame view

The frame view of an episode refers to a quantitative perspective that presents a comprehensive observation for each frame. This information is structured in the form of a table, where vehicles are listed in rows and their corresponding features are displayed in columns. Additionally, the action undertaken during the specific frames is also included. For instance, Figure 51 illustrates the frame view of frames 39 and 40 in episode 40. The perspective additionally incorporates the SHAP values, represented by a color code superimposed on the tabulated values. The values of attention are also color-encoded in the row header, specific to each vehicle.

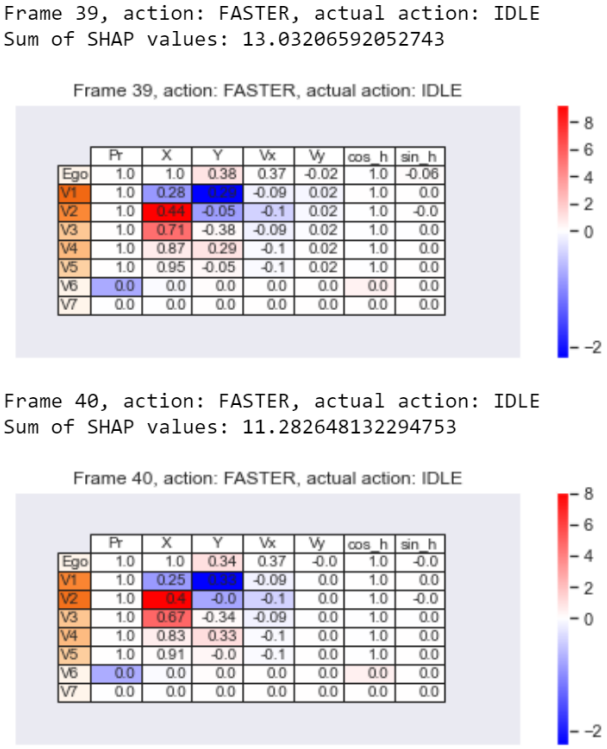


Figure 51: Frame view of frames 39 and 40, episode 40. Features values normalized [235]

The agent's decision in frames 39 and 40 is *faster*. Given that the vehicle is currently operating at its maximum velocity, the ensuing action is *idle*. In frame 39, the maximum attention is observed on *v1*, while the maximum SHAP value is observed on *v2*. However, in the subsequent frame, the attention and SHAP values converge, both indicating vehicle *v2* as the focal point. Upon initial observation, it is evident that attention values are evenly distributed among vehicles, with a proportional relationship to longitudinal distance, particularly in the same lane as the vehicle (y=0), which corresponds to the central lane. In contrast, SHAP values are specifically focused on the

longitudinal distance of *v2* and *v3*. The ticks located on the color scale positioned at the right side of each frame provide an indication of the precise SHAP values. Consequently, it can be observed that both frames exhibit a predominant presence of positive values. Specifically, the cumulative SHAP values for all features amount to 13.0 and 11.3 in the respective frames. This suggests that the Q value surpasses the mean value, potentially due to the presence of ample space in the vicinity of the vehicle's front region. Positive rewards are exclusively granted for achieving high speed, while negative rewards are assigned for instances of collisions. In frame 39, the vehicle with the highest attention would impede the model's inclination towards selecting a faster action, while in the subsequent frame, the nearest vehicle (*v1*) would similarly deter such action. In the current context, *v1* does not receive the highest level of attention, as it is positioned in a different lane, albeit by a small margin. This observation appears to validate the prior perception that the model's decision-making process exhibits greater foresight compared to the MAV, as it relies more heavily on information pertaining to vehicles situated at greater distances.

As expected, the concluding frames (73-75) of the episode present a considerable level of difficulty. In Figure 52, the EV consistently occupies the central lane, while no other vehicles are present within its designated lane. Therefore, it would be a straightforward task for it to maintain a constant velocity without any deviations. However, the progress of the vehicle is impeded by the proximity of two adjacent

vehicles in the adjacent lanes, which captures all the driver's focus (as they are the sole entities within the visual field). The MSV is assigned to the trigonometric orientations of three vehicles that are not present in the current scene. As anticipated, one could argue that the model intends this as an indication of presence. However, it is worth noting that the tangible presence aspect of these vehicles serves as a significant deterrent to the intended course of action. This discrepancy underscores the challenge that the agent encounters when confronted with this situation. Following the occurrence of two *slower* actions at frames 73 and 75, the agent undergoes *faster* in frame 78, subsequently assuming an *idle* state in frame 79, without reaching its maximum velocity. Considering the aforementioned factors, it becomes evident that the decisions made by agents are typically influenced by a reference vehicle that is positioned at a considerable distance from the EV, specifically in the lane that is the target of the decision-making process. Within the final frames of each simulation, which are infrequent during the training process, the absence of a vehicle is observed, thereby posing a challenge for the model to effectively address this scenario. In actuality, the agent's ability to overtake all 15 vehicles and successfully complete the episode is a rare occurrence. When considering the frame view and examining the non-chosen actions, it becomes evident that the trigonometric heading of *v3*, *v4*, and *v5* serves as a motivating factor for all of them, thus confirming the presence of uncertainty. However, if we narrow our analysis to the present vehicles, it becomes apparent that the EV y-coordinate holds

primary significance, as previously observed in other scenarios involving non-*idle* actions.

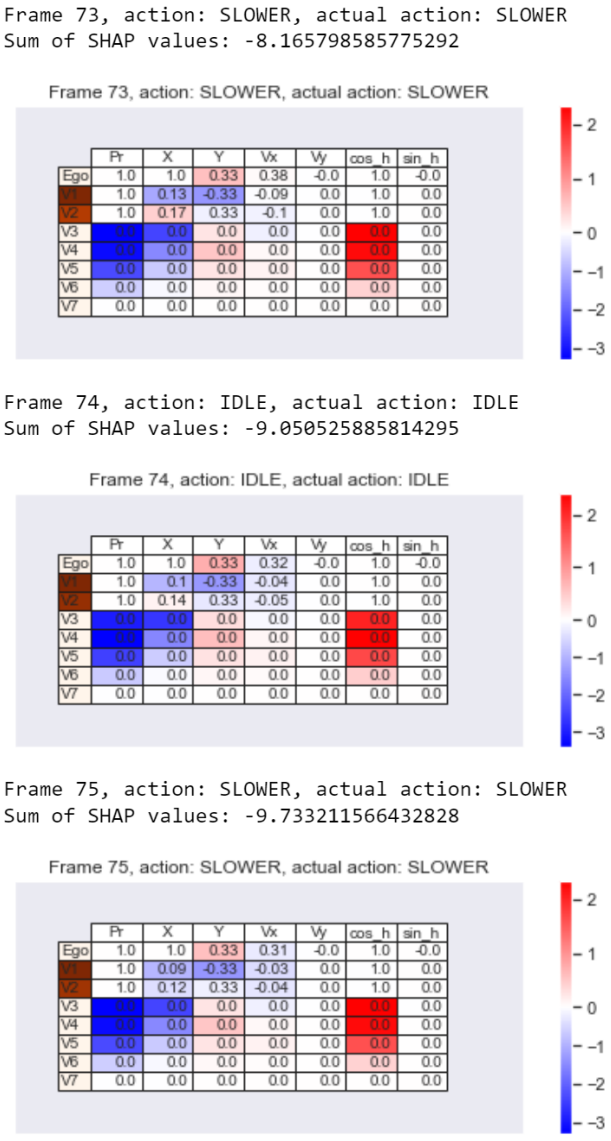


Figure 52: Frame view of episode 40, frames 73-75 [235]

6.2.3 Aggregated view

The episode analysis provides valuable insights for further examination. Nevertheless, conducting a comprehensive statistical analysis across a series of episodes would prove beneficial, potentially even subdividing specific driving situations. Furthermore, the preceding analysis largely overlooks the significance of 2D spatial factors. The rationale behind our third approach, referred to as the aggregated view, is to provide an analysis based on a collection of 150 test episodes. In conducting the analysis, unsuccessful episodes are excluded to prevent the inclusion of unfavorable behaviors that may compromise the accuracy of the interpretation.

The distribution of the determined actions is as follows: The distribution of percentages in the given data is as follows: 2% for *idle*, 21% for *right*, 10% for *left*, 60% for *faster*, and 7% for *slower*. In relation to actions, the allocation is as follows: 70% of the time is spent in an idle state, 9% of the time is dedicated to moving to the right, 9% of the time is allocated to moving to the left, 6% of the time is spent in accelerated motion, and 6% of the time is devoted to deceleration. The EV exhibits its maximum velocity in 85% of the frames, while it demonstrates its minimum velocity in 7% of the frames. The agent's average speed is 104 km/h, resulting in a distance traveled of 2.3 km. The agent exhibits a propensity for occupying the right lane, with a frequency of 45%, while the center and left lanes are chosen 28% of the time each. This preference is observed even in the absence of explicit

rules or rewards pertaining to lane selection. We contend that this phenomenon can be attributed to the increased potential for danger in the central lane, which arises from the possibility of vehicles merging from both the left and right sides. The acquisition of a preference for either the right or left lane may occur in a random manner during the training process. This can be attributed to the variability of traffic conditions, as evidenced by instances where a preference for the left lane was observed instead of the right. The inclination towards driving in a lateral lane can account for the human-like overtaking behaviors evident in the "Ego lane" chart depicted in Figure 50.

The highest level of attention is on *v2* and *v1*. Conversely, the maximum SHAP values are observed in *v2*, occasionally extending to *v0*, *v3*, and *v4*, with minimal presence in other areas. Additionally, the analysis reveals that the maximum attention and SHAP values coincide for the same vehicle in only 27% of instances. Also, only 31% of the two maximum values are within the second maximum value of the other, providing further evidence of a distinct separation between the attention layer and the network's decision-making output.

The max SHAP features are the longitudinal distance, which accounts for 76% of the observed occurrences, followed by the lateral position (i.e., lane) at 15%, and trigonometric heading at 9%. It is worth noting that the trigonometric heading is considered an indicator of the absence of the vehicle in the scene, as previously mentioned.

The mean distance of the MAV is 31.8 m, with a standard deviation of 15.2 m. This value is significantly lower compared to the MSV, which has an average distance of 47.9 m and a standard deviation of 25.0 m. In both instances, the EV is omitted.

Except for the EV cases, the vehicle in the same lane as the EV receives greater attention in the majority of instances (61%), compared to the maximum attention received by the max SHAP vehicle (46%). This implies that the distinction between attention and SHAP also encompasses the latitudinal dimension. The distinction between MAV and MSV is even more evident through a spatial analysis conducted on heatmaps (Figure 53). The EV is in the (2, 1) cell of the grid view, corresponding to the third row and second column. The rows in the grid represent the relative lanes in relation to the EV. For instance, the fifth row represents the second lane to the right of the EV, which may or may not exist in a particular frame. The width of each cell measures 5 m. The diagram illustrates the spatial arrangement of the MAV and MSV, as depicted in Figure 53a and Figure 53c, respectively. Figure 53b and Figure 53d report the same values normalized by the traffic within the cell. The data presented in the figures indicate that the primary focus of attention is predominantly towards the front of the vehicle, with minimal attention allocated laterally. The concept of normalization emphasizes the fact that the agent consistently allocates the highest level of attention to any vehicle located within the three to four cells ahead. The maximum values of SHAP, when not on the EV,

are observed at a greater distance. When examining normalized values, it becomes evident that the maximum SHAP values exhibit a greater dispersion across the various lanes. The findings of this study validate the previous analyses, which suggest that the network decision-making process is primarily influenced by the EV state (specifically, the lane) or, more commonly, by the presence of a vehicle in the target lane, rather than the nearby vehicle in the same lane (which receives most of the attention). This vehicle appears to serve as a point of reference for the decision-making process. It is crucial to emphasize that SHAP, attention mechanisms, and the DRL NN primarily pertain to vehicles rather than sparsely populated road segments.

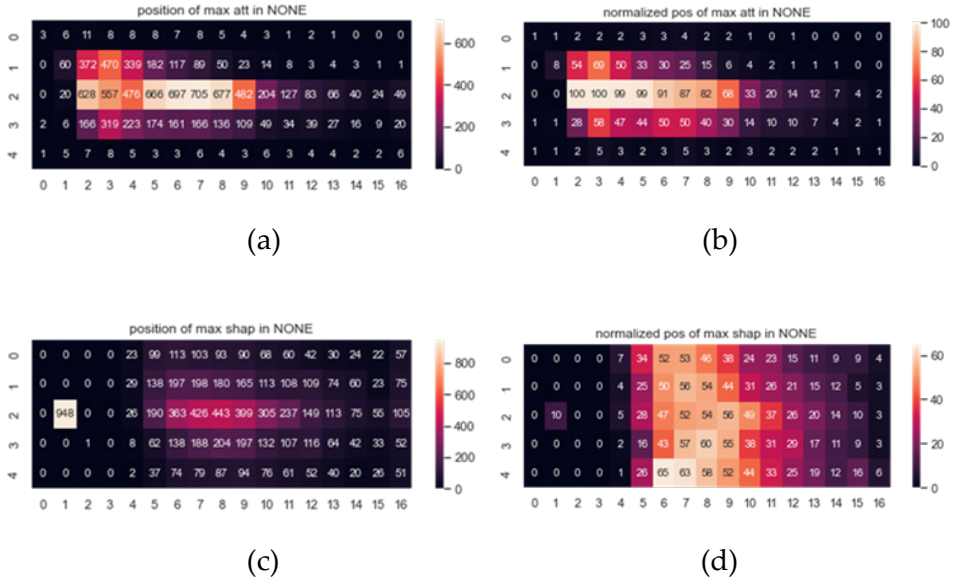


Figure 53: Heatmaps representing, on the road grid (relative to the EV), the number of times in which a vehicle in the cell gets max attention (a) and max SHAP (c). On the right (b and d), values are normalized by the traffic in the cell, thus numbers represent percentages [235]

In summary, by computing the most significant SHAP feature for each position in the grid (refer to Figure 54), it is confirmed that lane information is the most crucial aspect to consider when analyzing the EV. In close proximity to the EV, there is a lack of noteworthy maximum SHAP cases. However, as the longitudinal distance increases, particularly beyond 10 meters, it emerges as the most significant feature. It is observed that speed does not emerge as a predominant feature in the SHAP model. This can be attributed to the consistently high speed of the EV, typically reaching 30 m/s, while NPVs tend to operate at relatively slower speeds, averaging around 22 m/s, with no substantial variations.

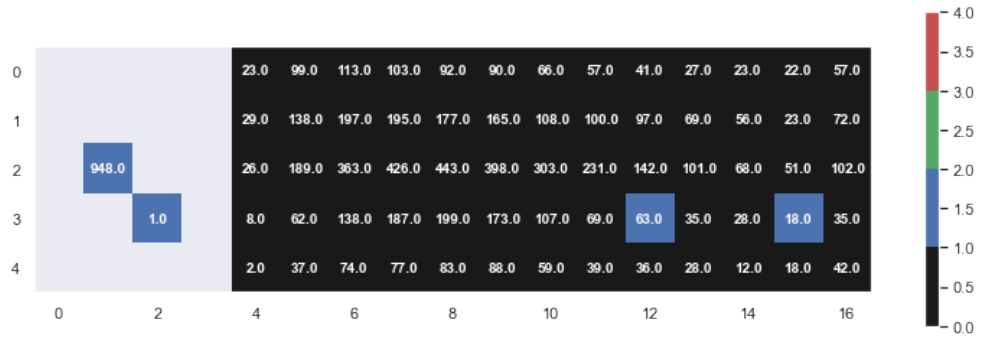


Figure 54: Absolute number of times of presence of max SHAP feature in the grid. Grey cells indicate no presence. Color code: black, x; blue, y; green velocity; red: trigonometric heading. The total number of samples is 9,040 [235]

7

Summary of contributions

The contributions made by the author in relation to this thesis are listed in Table 25, along with resulting publications in scientific journals or international conferences.

Table 25: Author's contributions and publications related to the thesis

Topic	Contribution	Publications
Edgine	Development of the Edgine system	[158]
	Application of the system to IoT and virtual contexts	[9], [237]
Embedded Voice Assistant	Development of the VA	[238]
DRL for low-speed maneuvering	Unity experiment	[12], [239]
	CARLA experiment	[240]
DRL models explainability	Study of the highway-env	[241]
	Explainability study	[235], [242]

8

Conclusions

This thesis proposed an edge AI approach in the field of automotive applications. The research has yielded significant advancements, also highlighting the importance of the interconnections between the different sub-areas of research touched upon.

We have successfully developed and implemented Edgine, a versatile tool independent of specific edge/cloud platforms and open source. Quantitatively, Edgine has demonstrated substantial improvements in data handling and development efficiency, reducing IoT applications implementation times according to users with a basic knowledge of the subject. Qualitatively, its adaptability was evident through successful deployment in both professional and educational settings. Future work will focus on enhancing Edgine's capabilities to process more complex data types and support advanced ML and DL models directly at the edge.

Building upon the foundational advancements in IoT, the exploration extended into the realm of embedded VAs. The development, deployment, and test of an end-to-end VA system, capable of functioning entirely offline and optimized for low-resource environments, has been illustrated. This system's support for the Italian language and its performance, comparable to cloud-connected solutions in terms of WER, CER, ICER, SER, and IRER, demonstrates the potential of integrating advanced AI methods in embedded systems, which also results in reduced latency and higher privacy standards. The application of transfer learning has been pivotal in this achievement, and future research aims to broaden the system's capabilities, including reducing initialization latency and extending language and domain support. Also, the Edgine integration allowed us to easily compute evaluation metrics with a streamlined process.

The journey then led us into the intricate world of DRL, particularly in the context of AD applied to a low-speed maneuvering automotive context. We successfully trained DRL agents for tasks such as map-less path planning and parking maneuvers, within both Unity and CARLA-simulated environments, proving the effectiveness of DRL in complex, real-time scenarios. Key factors identified for successful training, such as curriculum learning and the fine-tuning of simulation parameters, underscored the nuanced balance required in DRL applications. Future research in this area is directed towards application in dynamic scenarios and real-world vehicle

implementation, starting from a scale model. For this application as well, the use of Edgine was essential to easily test the model and collect results.

The exploration concluded with an investigation in the realm of DRL model explainability. As DL models become increasingly complex, understanding their decision-making processes becomes crucial, especially in applications like AD. The novel approach for interpretability analysis implemented, utilizing a combination of episode timelines, frame-by-frame analysis, and aggregated statistical analysis, has provided profound insights into the behavior and decision-making factors of DRL models. This analysis not only revealed the complex decision-making processes of the models, but also highlighted areas for further research, such as exploring temporal correlations, more complex vehicular models, and refining training methods through incident analysis.

In summary, this collective exploration weaved together the advancements in IoT technologies with Edgine, the evolution of VAs in low-resource automotive environments and in a non-mainstream language such as Italian, the practical application of DRL in AD, and the critical aspect of DRL explainability. Each element of this exploration builds upon the other, showcasing the ever-evolving landscape in IoT, AI, and AD technology and paving the way for future innovations and interconnections in these fields.

References

[1] L. Lin, X. Liao, H. Jin, and P. Li, "Computation Offloading Toward Edge Computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, Aug. 2019, doi: 10.1109/JPROC.2019.2922285.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013, doi: 10.1016/j.future.2013.01.010.

[3] M. Hoy, "Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants," *Medical Reference Services Quarterly*, vol. 37, pp. 81–88, Jan. 2018, doi: 10.1080/02763869.2018.1404391.

[4] L. Hernández Acosta and D. Reinhardt, "A survey on privacy issues and solutions for Voice-controlled Digital Assistants," *Pervasive and Mobile Computing*, vol. 80, p. 101523, Feb. 2022, doi: 10.1016/j.pmcj.2021.101523.

[5] J. C. S. Dos Anjos *et al.*, "Data Processing Model to Perform Big Data Analytics in Hybrid Infrastructures," *IEEE Access*, vol. 8, pp. 170281–170294, 2020, doi: 10.1109/ACCESS.2020.3023344.

[6] *WWDC 2021 — June 7 | Apple*, (Jun. 07, 2021). Accessed: Jun. 21, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=0TD96VTf0Xs

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[8] X. Wang, T. Wei, L. Kong, L. He, F. Wu, and G. Chen, "ECASS: Edge computing based auxiliary sensing system for self-driving vehicles," *Journal of Systems Architecture*, vol. 97, pp. 258–268, Aug. 2019, doi: 10.1016/j.sysarc.2019.02.014.

[9] R. Berta, F. Bellotti, A. De Gloria, and L. Lazzaroni, "Assessing Versatility of a Generic End-to-End Platform for IoT Ecosystem Applications," *Sensors*, vol. 22, no. 3, Art. no. 3, Jan. 2022, doi: 10.3390/s22030713.

[10] "NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics," NVIDIA Technical Blog. Accessed: Feb. 28, 2022. [Online]. Available: https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/

[11] S. Mittal, "A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform," *Journal of Systems Architecture*, vol. 97, pp. 428–442, Aug. 2019, doi: 10.1016/j.sysarc.2019.01.011.

[12] L. Lazzaroni, F. Bellotti, A. Capello, M. Cossu, A. De Gloria, and R. Berta, "Deep Reinforcement Learning for Automated Car Parking," in *Applications in Electronics Pervading Industry, Environment and Society*, R. Berta and A. De Gloria, Eds., in Lecture Notes in Electrical Engineering. Cham: Springer Nature Switzerland, 2023, pp. 125–130. doi: 10.1007/978-3-031-30333-3_16.

[13] A. Vaswani *et al.*, "Attention Is All You Need," *arXiv:1706.03762 [cs]*, Dec. 2017, doi: 10.48550/arXiv.1706.03762.

[14] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: Jul. 11, 2022. [Online]. Available: https://papers.nips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[15] C. Molnar, *Interpretable Machine Learning*. Accessed: Aug. 08, 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book/

[16] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable AI: A Review of Machine Learning Interpretability Methods," *Entropy*, vol. 23, no. 1, Art. no. 1, Jan. 2021, doi: 10.3390/e23010018.

[17] R. Liessner, J. Dohmen, and M. Wiering, "Explainable Reinforcement Learning for Longitudinal Control: 13th International Conference on Agents and Artificial Intelligence, ICAART 2021," *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, pp. 874–881, 2021.

[18] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, in KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 1135–1144. doi: 10.1145/2939672.2939778.

[19] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning Important Features Through Propagating Activation Differences." arXiv, Oct. 12, 2019. doi: 10.48550/arXiv.1704.02685.

[20] A. Binder, G. Montavon, S. Bach, K.-R. Müller, and W. Samek, "Layer-wise Relevance Propagation for Neural Networks with Local Renormalization Layers." arXiv, Apr. 04, 2016. doi: 10.48550/arXiv.1604.00825.

[21] "GitHub - eleurent/highway-env: A minimalist environment for decision-making in autonomous driving." Accessed: Jul. 11, 2022. [Online]. Available: https://github.com/eleurent/highway-env

[22] P. Polack, F. Altché, B. d'Andréa-Novel, and A. de La Fortelle, "The Kinematic Bicycle Model: a Consistent Model for Planning Feasible Trajectories for Autonomous Vehicles?," in *IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, United States, Jun. 2017. Accessed: Jul. 11,

2022. [Online]. Available: https://hal-polytechnique.archives-ouvertes.fr/hal-01520869

[23] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, no. 2, pp. 1805–1824, Aug. 2000, doi: 10.1103/PhysRevE.62.1805.

[24] "General Lane-Changing Model MOBIL for Car-Following Models - Arne Kesting, Martin Treiber, Dirk Helbing, 2007." Accessed: Jul. 12, 2022. [Online]. Available: https://journals.sagepub.com/doi/10.3141/1999-10

[25] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," no. arXiv:1711.03938. arXiv, Nov. 10, 2017. doi: 10.48550/arXiv.1711.03938.

[26] P. Alvarez Lopez *et al.*, "Microscopic Traffic Simulation using SUMO," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Maui, USA: IEEE, Nov. 2018, pp. 2575–2582. Accessed: Jul. 11, 2022. [Online]. Available: https://www.itsc2019.org/

[27] J. Liao, T. Liu, X. Tang, X. Mu, B. Huang, and D. Cao, "Decision-Making Strategy on Highway for Autonomous Vehicles Using Deep Reinforcement Learning," *IEEE Access*, vol. 8, pp. 177804–177814, 2020, doi: 10.1109/ACCESS.2020.3022755.

[28] S. Zhang, Y. Wu, H. Ogai, H. Inujima, and S. Tateno, "Tactical Decision-Making for Autonomous Driving Using Dueling Double Deep Q Network With Double Attention," *IEEE Access*, vol. 9, pp. 151983–151992, 2021, doi: 10.1109/ACCESS.2021.3127105.

[29] R. Chandra, A. Bera, and D. Manocha, "Using Graph-Theoretic Machine Learning to Predict Human Driver Behavior," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 2572–2585, Mar. 2022, doi: 10.1109/TITS.2021.3130218.

[30] "AWS IoT Greengrass Documentation." Accessed: Dec. 19, 2021. [Online]. Available: https://docs.aws.amazon.com/greengrass/

[31] "AWS Lambda Documentation." Accessed: Dec. 19, 2021. [Online]. Available: https://docs.aws.amazon.com/lambda/?id=docs_gateway

[32] "Edge TPU - Run Inference at the Edge," Google Cloud. Accessed: Dec. 18, 2021. [Online]. Available: https://cloud.google.com/edge-tpu

[33] "Bringing intelligence to the edge with Cloud IoT," Google Cloud Blog. Accessed: Dec. 18, 2021. [Online]. Available: https://cloud.google.com/blog/products/gcp/bringing-intelligence-edge-cloud-iot/

[34] "TensorFlow Lite | ML for Mobile and Edge Devices," TensorFlow. Accessed: Dec. 18, 2021. [Online]. Available: https://www.tensorflow.org/lite

[35] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," no. arXiv:1603.04467. arXiv, Mar. 16, 2016. doi: 10.48550/arXiv.1603.04467.

[36] R. David *et al.*, "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," *arXiv:2010.08678 [cs]*, Mar. 2021, Accessed: Dec. 18, 2021. [Online]. Available: http://arxiv.org/abs/2010.08678

[37] C. N. Coelho *et al.*, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nat Mach Intell*, vol. 3, no. 8, Art. no. 8, Aug. 2021, doi: 10.1038/s42256-021-00356-5.

[38] "Keras 3: A new multi-backend Keras." Keras, Oct. 03, 2023. Accessed: Oct. 03, 2023. [Online]. Available: https://github.com/keras-team/keras

[39] "IBM Edge Application Manager - Overview." Accessed: Dec. 19, 2021. [Online]. Available: https://www.ibm.com/cloud/edge-application-manager

[40] "Red Hat OpenShift makes container orchestration easier." Accessed: Dec. 19, 2021. [Online]. Available: https://www.redhat.com/en/technologies/cloud-computing/openshift

[41] D. Jensen, *Beginning Azure IoT Edge Computing: Extending the Cloud to the Intelligent Edge*. Apress, 2019.

[42] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," presented at the Proceedings of the 45th International Symposium on Computer Architecture, 2018, Jun. 2018. Accessed: Dec. 19, 2021. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-configurable-cloud-scale-dnn-processor-for-real-time-ai/

[43] D. K. Dennis *et al.*, "EdgeML: Machine Learning for resource-constrained edge devices." [Online]. Available: https://github.com/Microsoft/EdgeML

[44] C. Gupta *et al.*, "ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices," p. 16.

[45] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, "Compiling KB-sized machine learning models to tiny IoT devices," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Phoenix AZ USA: ACM, Jun. 2019, pp. 79–95. doi: 10.1145/3314221.3314597.

[46] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things," in *Proceedings of the 34th International Conference on Machine Learning*, PMLR, Jul. 2017, pp. 1935–1944. Accessed:

Jan. 17, 2022. [Online]. Available: https://proceedings.mlr.press/v70/kumar17a.html

[47] S. G. Patil *et al.*, "GesturePod: Enabling On-device Gesture-based Interaction for White Cane Users," in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, New Orleans LA USA: ACM, Oct. 2019, pp. 403–415. doi: 10.1145/3332165.3347881.

[48] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network," p. 23.

[49] D. Dennis, C. Pabbaraju, H. V. Simhadri, and P. Jain, "Multiple Instance Learning for Efficient Sequential Data Classification on Resource-constrained Devices," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2018. Accessed: Jan. 17, 2022. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/d9fbed9da256e344c1fa46bb46c34c5f-Abstract.html

[50] Y. Cui, Y. Liang, and R. Wang, "Resource Allocation Algorithm With Multi-Platform Intelligent Offloading in D2D-Enabled Vehicular Networks," *IEEE Access*, vol. 7, pp. 21246–21253, 2019, doi: 10.1109/ACCESS.2018.2882000.

[51] O. Väänänen and T. Hämäläinen, "Requirements for Energy Efficient Edge Computing: A Survey," in *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, O. Galinina, S. Andreev, S. Balandin, and Y. Koucheryavy, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 3–15. doi: 10.1007/978-3-030-01168-0_1.

[52] R. Morabito, "Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017, doi: 10.1109/ACCESS.2017.2704444.

[53] O. Debauche, S. Mahmoudi, S. A. Mahmoudi, P. Manneback, and F. Lebeau, "Edge Computing and Artificial Intelligence Semantically Driven. Application to a Climatic Enclosure," *Procedia Computer Science*, vol. 175, pp. 542–547, Jan. 2020, doi: 10.1016/j.procs.2020.07.077.

[54] X. Wu, R. Dunne, Q. Zhang, and W. Shi, "Edge computing enabled smart firefighting: opportunities and challenges," in *Proceedings of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies - HotWeb '17*, San Jose, California: ACM Press, 2017, pp. 1–6. doi: 10.1145/3132465.3132475.

[55] Z. Idrees, Z. Zou, and L. Zheng, "Edge Computing Based IoT Architecture for Low Cost Air Pollution Monitoring Systems: A Comprehensive System Analysis, Design Considerations & Development," *Sensors*, vol. 18, no. 9, Art. no. 9, Sep. 2018, doi: 10.3390/s18093021.

[56] H. Sun, X. Liang, and W. Shi, "*VU*: video usefulness and its application in large-scale video surveillance systems: an early experience," in *Proceedings of the Workshop on Smart Internet of Things*, in SmartIoT '17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1–6. doi: 10.1145/3132479.3132485.

[57] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, and Q. Zhang, "Edge Computing in IoT-Based Manufacturing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 103–109, Sep. 2018, doi: 10.1109/MCOM.2018.1701231.

[58] R. Ardila *et al.*, "Common Voice: A Massively-Multilingual Speech Corpus," *arXiv:1912.06670 [cs]*, Mar. 2020, doi: 10.48550/arXiv.1912.06670.

[59] "Common Voice by Mozilla." Accessed: Dec. 14, 2021. [Online]. Available: https://commonvoice.mozilla.org/

[60] "The M-AILABS Speech Dataset – caito." Accessed: Dec. 14, 2021. [Online]. Available: https://www.caito.de/2019/01/the-m-ailabs-speech-dataset/

[61] "LibriVox: Free Public Domain Audiobooks," *Reference Reviews*, vol. 28, no. 1, pp. 7–8, Jan. 2014, doi: 10.1108/RR-08-2013-0197.

[62] "Project Gutenberg," Project Gutenberg. Accessed: Dec. 14, 2021. [Online]. Available: https://www.gutenberg.org/

[63] V. Pratap, Q. Xu, A. Sriram, G. Synnaeve, and R. Collobert, "MLS: A Large-Scale Multilingual Dataset for Speech Research," *Interspeech 2020*, pp. 2757–2761, Oct. 2020, doi: 10.21437/Interspeech.2020-2826.

[64] "VOSK Offline Speech Recognition API," VOSK Offline Speech Recognition API. Accessed: Dec. 14, 2021. [Online]. Available: https://alphacephei.com/vosk/

[65] "Project DeepSpeech." Mozilla, Feb. 02, 2022. Accessed: Feb. 02, 2022. [Online]. Available: https://github.com/mozilla/DeepSpeech

[66] R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2Letter: an End-to-End ConvNet-based Speech Recognition System," *arXiv:1609.03193 [cs]*, Sep. 2016, doi: 10.48550/arXiv.1609.03193.

[67] O. Kuchaiev *et al.*, "NeMo: a toolkit for building AI applications using Neural Modules," *arXiv:1909.09577 [cs, eess]*, Sep. 2019, doi: 10.48550/arXiv.1909.09577.

[68] N. Nagari, "Comparing 4 Popular Open Source Speech To Text Neural Network Models," Medium. Accessed: Feb. 02, 2022. [Online]. Available: https://medium.com/@nick.nagari/comparing-4-popular-open-source-speech-to-text-neural-network-models-92676a9f9265

[69] Z. Weng, Z. Qin, X. Tao, C. Pan, G. Liu, and G. Y. Li, "Deep Learning Enabled Semantic Communications with Speech Recognition and Synthesis." arXiv, Mar. 31, 2023. doi: 10.48550/arXiv.2205.04603.

[70] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations." arXiv, Oct. 22, 2020. doi: 10.48550/arXiv.2006.11477.

[71] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2015, pp. 5206–5210. doi: 10.1109/ICASSP.2015.7178964.

[72] L. Torrey and J. Shavlik, *Transfer Learning*. IGI Global, 2010, pp. 242–264. doi: 10.4018/978-1-60566-766-9.ch011.

[73] D. Wang and T. F. Zheng, "Transfer learning for speech and language processing," in *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, Dec. 2015, pp. 1225–1237. doi: 10.1109/APSIPA.2015.7415532.

[74] J. Huang *et al.*, "Cross-Language Transfer Learning, Continuous Learning, and Domain Adaptation for End-to-End Automatic Speech Recognition," *arXiv:2005.04290 [eess]*, May 2020, doi: 10.48550/arXiv.2005.04290.

[75] "Develop Smaller Speech Recognition Models with NVIDIA's NeMo Framework," NVIDIA Developer Blog. Accessed: Feb. 02, 2022. [Online]. Available: https://developer.nvidia.com/blog/develop-smaller-speech-recognition-models-with-nvidias-nemo-framework/

[76] "Build chatbots with Dialogflow," Google Developers. Accessed: Feb. 02, 2022. [Online]. Available: https://developers.google.com/learn/pathways/chatbots-dialogflow

[77] "Conversational AI and Chatbots - Amazon Lex - Amazon Web Services," Amazon Web Services, Inc. Accessed: Jun. 21, 2023. [Online]. Available: https://aws.amazon.com/lex/

[78] "Wit.ai." Accessed: Jun. 21, 2023. [Online]. Available: https://wit.ai/docs/quickstart

[79] "Microsoft Bot Framework." Accessed: Feb. 02, 2022. [Online]. Available: https://dev.botframework.com/

[80] "Open source conversational AI," Rasa. Accessed: Feb. 02, 2022. [Online]. Available: https://rasa.com/

[81] "spaCy · Industrial-strength Natural Language Processing in Python." Accessed: Feb. 02, 2022. [Online]. Available: https://spacy.io/

[82] M. Burtsev *et al.*, "DeepPavlov: Open-Source Library for Dialogue Systems," in *Proceedings of ACL 2018, System Demonstrations*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 122–127. doi: 10.18653/v1/P18-4021.

[83] M. Saxon, S. Choudhary, J. P. McKenna, and A. Mouchtaris, "End-to-End Spoken Language Understanding for Generalized Voice Assistants," *Interspeech 2021*, pp. 4738–4742, Aug. 2021, doi: 10.21437/Interspeech.2021-1826.

[84] L. Lugosch, M. Ravanelli, P. Ignoto, V. S. Tomar, and Y. Bengio, "Speech Model Pre-training for End-to-End Spoken Language Understanding," *arXiv:1904.03670 [cs, eess]*, Jul. 2019, doi: 10.48550/arXiv.1904.03670.

[85] A. Coucke *et al.*, "Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces." arXiv, Dec. 06, 2018. doi: 10.48550/arXiv.1805.10190.

[86] L. Massai, P. Nesi, and G. Pantaleo, "PAVAL: A location-aware virtual personal assistant for retrieving geolocated points of interest and location-based services," *Engineering Applications of Artificial Intelligence*, vol. 77, pp. 70–85, Jan. 2019, doi: 10.1016/j.engappai.2018.09.013.

[87] J. Shen *et al.*, "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions," *arXiv:1712.05884 [cs]*, Feb. 2018, doi: 10.48550/arXiv.1712.05884.

[88] T. Nekvinda and O. Dušek, "One Model, Many Languages: Meta-learning for Multilingual Text-to-Speech," *arXiv:2008.00768 [cs, eess]*, Aug. 2020, doi: 10.48550/arXiv.2008.00768.

[89] "Tacotron 2 (without wavenet)." NVIDIA Corporation, Feb. 02, 2022. Accessed: Feb. 02, 2022. [Online]. Available: https://github.com/NVIDIA/tacotron2

[90] R. Prenger, R. Valle, and B. Catanzaro, "Waveglow: A Flow-based Generative Network for Speech Synthesis," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 3617–3621. doi: 10.1109/ICASSP.2019.8683143.

[91] B. Zhai *et al.*, "SqueezeWave: Extremely Lightweight Vocoders for On-device Speech Synthesis," *arXiv:2001.05685 [cs, eess]*, Jan. 2020, doi: 10.48550/arXiv.2001.05685.

[92] "TTS Vocoder Uniglow | NVIDIA NGC," NGC Catalog. Accessed: Feb. 02, 2022. [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/nemo/models/tts_uniglow

[93] K. Kumar *et al.*, "MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis," *arXiv:1910.06711 [cs, eess]*, Dec. 2019, doi: 10.48550/arXiv.1910.06711.

[94] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis," *arXiv:2010.05646 [cs, eess]*, Oct. 2020, doi: 10.48550/arXiv.2010.05646.

[95] "TTS En E2E FastPitch Hifigan | NVIDIA NGC," NGC Catalog. Accessed: Feb. 02, 2022. [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/nemo/models/tts_en_e2e_fastpitchhifigan

[96] A. Łańcucki, "Fastpitch: Parallel Text-to-Speech with Pitch Prediction," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Jun. 2021, pp. 6588–6592. doi: 10.1109/ICASSP39728.2021.9413889.

[97] "TTS En E2E Fastspeech2 Hifigan | NVIDIA NGC," NGC Catalog. Accessed: Feb. 02, 2022. [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/nemo/models/tts_en_e2e_fastspeech2hifigan

[98] Y. Ren *et al.*, "FastSpeech 2: Fast and High-Quality End-to-End Text to Speech," *arXiv:2006.04558 [cs, eess]*, Mar. 2021, doi: 10.48550/arXiv.2006.04558.

[99] X. Tan *et al.*, "NaturalSpeech: End-to-End Text to Speech Synthesis with Human-Level Quality." arXiv, May 10, 2022. doi: 10.48550/arXiv.2205.04421.

[100] R. Huang *et al.*, "FastDiff: A Fast Conditional Diffusion Model for High-Quality Speech Synthesis." arXiv, Apr. 21, 2022. doi: 10.48550/arXiv.2204.09934.

[101] "The LJ Speech Dataset." Accessed: Jun. 21, 2023. [Online]. Available: https://keithito.com/LJ-Speech-Dataset

[102] M. Luo, X. Hou, and J. Yang, "Surface Optimal Path Planning Using an Extended Dijkstra Algorithm," *IEEE Access*, vol. 8, pp. 147827–147838, 2020, doi: 10.1109/ACCESS.2020.3015976.

[103] L. Liu *et al.*, "Path Planning for Smart Car Based on Dijkstra Algorithm and Dynamic Window Approach," *Wireless Communications and Mobile Computing*, vol. 2021, p. e8881684, Feb. 2021, doi: 10.1155/2021/8881684.

[104] Z. Boroujeni, D. Goehring, F. Ulbrich, D. Neumann, and R. Rojas, "Flexible unit A-star trajectory planning for autonomous vehicles on structured road maps," in *2017 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Vienna, Austria: IEEE, Jun. 2017, pp. 7–12. doi: 10.1109/ICVES.2017.7991893.

[105] S. Sedighi, D.-V. Nguyen, and K.-D. Kuhnert, "Guided Hybrid A-star Path Planning Algorithm for Valet Parking Applications," in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, Apr. 2019, pp. 570–575. doi: 10.1109/ICCAR.2019.8813752.

[106] S. Pothan, J. L. Nandagopal, and G. Selvaraj, "Path planning using state lattice for autonomous vehicle," in *2017 International Conference on*

Technological Advancements in Power and Energy (TAP Energy), Dec. 2017, pp. 1–5. doi: 10.1109/TAPENERGY.2017.8397363.

[107] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug. 1996, doi: 10.1109/70.508439.

[108] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001, doi: 10.1177/02783640122067453.

[109] D. Gonzalez Bautista, J. Pérez, V. Milanes, and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 1–11, Nov. 2015, doi: 10.1109/TITS.2015.2498841.

[110] M. A. R. Pohan, B. R. Trilaksono, S. P. Santosa, and A. S. Rohman, "Path Planning Algorithm Using the Hybridization of the Rapidly-Exploring Random Tree and Ant Colony Systems," *IEEE Access*, vol. 9, pp. 153599–153615, 2021, doi: 10.1109/ACCESS.2021.3127635.

[111] "Bézier curve-based trajectory planning for autonomous vehicles with collision avoidance - Zheng - 2020 - IET Intelligent Transport Systems - Wiley Online Library." Accessed: Feb. 07, 2023. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-its.2020.0355

[112] P. Wang, J. Yang, Y. Zhang, Q. Wang, B. Sun, and D. Guo, "Obstacle-Avoidance Path-Planning Algorithm for Autonomous Vehicles Based on B-Spline Algorithm," *World Electric Vehicle Journal*, vol. 13, no. 12, Art. no. 12, Dec. 2022, doi: 10.3390/wevj13120233.

[113] K. Hao, J. Zhao, K. Yu, C. Li, and C. Wang, "Path Planning of Mobile Robots Based on a Multi-Population Migration Genetic Algorithm," *Sensors*, vol. 20, no. 20, Art. no. 20, Jan. 2020, doi: 10.3390/s20205873.

[114] H. Nobahari and S. Nasrollahi, "A terminal guidance algorithm based on ant colony optimization," *Computers & Electrical Engineering*, vol. 77, pp. 128–146, Jul. 2019, doi: 10.1016/j.compeleceng.2019.05.012.

[115] Y. Ling, N. Yang, H. Yu, and Y. Zhu, "Novel Bayesian Network Incremental Learning Method Based on Particle Swarm Optimization Algorithm," in *Emerging Trends in Intelligent and Interactive Systems and Applications*, M. Tavana, N. Nedjah, and R. Alhajj, Eds., in Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2021, pp. 941–947. doi: 10.1007/978-3-030-63784-2_114.

[116] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," in *Autonomous Robot Vehicles*, I. J. Cox and G. T. Wilfong, Eds., New York, NY: Springer, 1990, pp. 396–404. doi: 10.1007/978-1-4613-8997-2_29.

[117] R. Szczepanski, T. Tarczewski, and K. Erwinski, "Energy Efficient Local Path Planning Algorithm Based on Predictive Artificial Potential Field," *IEEE Access*, vol. 10, pp. 39729–39742, 2022, doi: 10.1109/ACCESS.2022.3166632.

[118] P. Grabusts, J. Musatovs, and V. Golenkov, "The application of simulated annealing method for optimal route detection between objects," *Procedia Computer Science*, vol. 149, pp. 95–101, Jan. 2019, doi: 10.1016/j.procs.2019.01.112.

[119] F. L. Silva, S. Filgueira da Silva, F. Mazzariol Santiciolli, J. J. Eckert, L. C. A. Silva, and F. G. Dedini, "Multi-objective Optimization of the Steering System and Fuzzy Logic Control Applied to a Car-Like Robot," in *Multibody Mechatronic Systems*, M. Pucheta, A. Cardona, S. Preidikman, and R. Hecker, Eds., in Mechanisms and Machine Science. Cham: Springer International Publishing, 2021, pp. 195–202. doi: 10.1007/978-3-030-60372-4_22.

[120] M. Kobayashi and N. Motoi, "Local Path Planning: Dynamic Window Approach With Virtual Manipulators Considering Dynamic Obstacles," *IEEE Access*, vol. 10, pp. 17018–17029, 2022, doi: 10.1109/ACCESS.2022.3150036.

[121] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning," *FNT in Machine Learning*, vol. 11, no. 3–4, pp. 219–354, 2018, doi: 10.1561/2200000071.

[122] R. Bellman, "DYNAMIC PROGRAMMING AND LAGRANGE MULTIPLIERS," *Proc Natl Acad Sci U S A*, vol. 42, no. 10, pp. 767–769, Oct. 1956.

[123] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach Learn*, vol. 8, no. 3, pp. 279–292, May 1992, doi: 10.1007/BF00992698.

[124] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[125] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning." arXiv, Dec. 08, 2015. doi: 10.48550/arXiv.1509.06461.

[126] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning." arXiv, Apr. 05, 2016. doi: 10.48550/arXiv.1511.06581.

[127] M. Sewak, "Deep Q Network (DQN), Double DQN, and Dueling DQN," *Deep Reinforcement Learning*, 2019, doi: 10.1007/978-981-13-8285-7_8.

[128] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, Art. no. 7540, Feb. 2015, doi: 10.1038/nature14236.

[129] X. Lei, Z. Zhang, and P. Dong, "Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning," *Journal of Robotics*, vol. 2018, p. e5781591, Sep. 2018, doi: 10.1155/2018/5781591.

[130] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," no. arXiv:1509.06461. arXiv, Dec. 08, 2015. doi: 10.48550/arXiv.1509.06461.

[131] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, "Robot Navigation of Environments with Unknown Rough Terrain Using deep Reinforcement Learning," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Aug. 2018, pp. 1–7. doi: 10.1109/SSRR.2018.8468643.

[132] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning." arXiv, Jun. 16, 2016. doi: 10.48550/arXiv.1602.01783.

[133] A. Faust *et al.*, "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning." arXiv, May 16, 2018. doi: 10.48550/arXiv.1710.03937.

[134] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators from RL Policies." arXiv, Jul. 12, 2019. doi: 10.48550/arXiv.1907.04799.

[135] J. Gao, W. Ye, J. Guo, and Z. Li, "Deep Reinforcement Learning for Indoor Mobile Robot Path Planning," *Sensors*, vol. 20, no. 19, Art. no. 19, Jan. 2020, doi: 10.3390/s20195493.

[136] Z. Chu, F. Wang, T. Lei, and C. Luo, "Path Planning Based on Deep Reinforcement Learning for Autonomous Underwater Vehicles Under Ocean Current Disturbance," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 108–120, Jan. 2023, doi: 10.1109/TIV.2022.3153352.

[137] Y. Peng, Y. Liu, and H. Zhang, "Deep Reinforcement Learning based Path Planning for UAV-assisted Edge Computing Networks," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2021, pp. 1–6. doi: 10.1109/WCNC49053.2021.9417292.

[138] E. Leurent, "An Environment for Autonomous Driving Decision-Making." May 2018. Accessed: Jun. 10, 2022. [Online]. Available: https://github.com/eleurent/highway-env

[139] G. Brockman *et al.*, "OpenAI Gym," no. arXiv:1606.01540. arXiv, Jun. 05, 2016. doi: 10.48550/arXiv.1606.01540.

[140] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, Art. no. 268, 2021.

[141] "Stable Baselines3." DLR-RM, Feb. 03, 2023. Accessed: Feb. 03, 2023. [Online]. Available: https://github.com/DLR-RM/stable-baselines3

[142] U. Technologies, "Real-time 3D Technology for Automotive and Transportation | Unity." Accessed: Feb. 03, 2023. [Online]. Available: https://unity.com/solutions/automotive-and-transportation

[143] A. Juliani *et al.*, "Unity: A General Platform for Intelligent Agents." arXiv, May 06, 2020. Accessed: May 27, 2022. [Online]. Available: http://arxiv.org/abs/1809.02627

[144] C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller, "Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2017, pp. 1671–1678. doi: 10.1109/IVS.2017.7995949.

[145] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2014, pp. 392–399. doi: 10.1109/ITSC.2014.6957722.

[146] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, no. 5, pp. 4282–4286, May 1995, doi: 10.1103/PhysRevE.51.4282.

[147] K. Messaoud, I. Yahiaoui, A. Verroust-Blondet, and F. Nashashibi, "Attention Based Vehicle Trajectory Prediction," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 1, p. 175, 2021, doi: 10.1109/TIV.2020.2991952.

[148] J. Wang, Q. Zhang, and D. Zhao, "Highway Lane Change Decision-Making via Attention-Based Deep Reinforcement Learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 3, pp. 567–569, Mar. 2022, doi: 10.1109/JAS.2021.1004395.

[149] E. Leurent and J. Mercat, "Social Attention for Autonomous Decision-Making in Dense Traffic." arXiv, Nov. 27, 2019. doi: 10.48550/arXiv.1911.12250.

[150] Y. Wu *et al.*, "HSTA: A Hierarchical Spatio-Temporal Attention Model for Trajectory Prediction," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 11, pp. 11295–11307, Nov. 2021, doi: 10.1109/TVT.2021.3115018.

[151] K. Coussement and D. F. Benoit, "Interpretable data science for decision making," *Decision Support Systems*, vol. 150, p. 113664, Nov. 2021, doi: 10.1016/j.dss.2021.113664.

[152] J. H. Friedman and J. J. Meulman, "Multiple additive regression trees with application in epidemiology," *Stat Med*, vol. 22, no. 9, pp. 1365–1381, May 2003, doi: 10.1002/sim.1501.

[153] M. Gashi *et al.*, "State-of-the-Art Explainability Methods with Focus on Visual Analytics Showcased by Glioma Classification," *BioMedInformatics*, vol. 2, no. 1, Art. no. 1, Mar. 2022, doi: 10.3390/biomedinformatics2010009.

[154] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A Survey of Methods for Explaining Black Box Models," *ACM Comput. Surv.*, vol. 51, no. 5, p. 93:1-93:42, Aug. 2018, doi: 10.1145/3236009.

[155] C. Molnar *et al.*, "General Pitfalls of Model-Agnostic Interpretation Methods for Machine Learning Models," in *xxAI - Beyond Explainable AI: International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers*, A. Holzinger, R. Goebel, R. Fong, T. Moon, K.-R. Müller, and W. Samek, Eds., in Lecture Notes in Computer Science. , Cham: Springer International Publishing, 2022, pp. 39–68. doi: 10.1007/978-3-031-04083-2_4.

[156] L. S. Shapley, "A Value for N-Person Games," RAND Corporation, Mar. 1952. Accessed: Jul. 11, 2022. [Online]. Available: https://www.rand.org/pubs/papers/P295.html

[157] R. Berta, A. Kobeissi, F. Bellotti, and A. De Gloria, "Atmosphere, an Open Source Measurement-Oriented Data Framework for IoT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 1927–1936, Mar. 2021, doi: 10.1109/TII.2020.2994414.

[158] R. Berta, A. Mazzara, F. Bellotti, A. De Gloria, and L. Lazzaroni, "Edgine, A Runtime System for IoT Edge Applications," in *Applications in Electronics Pervading Industry, Environment and Society*, S. Saponara and A. De Gloria, Eds., in Lecture Notes in Electrical Engineering. Cham: Springer International Publishing, 2021, pp. 261–266. doi: 10.1007/978-3-030-66729-0_31.

[159] W. Dai, H. Nishi, V. Vyatkin, V. Huang, Y. Shi, and X. Guan, "Industrial Edge Computing: Enabling Embedded Intelligence," *IEEE Industrial Electronics Magazine*, vol. 13, no. 4, pp. 48–56, Dec. 2019, doi: 10.1109/MIE.2019.2943283.

[160] "arduino - Understanding how to use an accelerometer to detect vehicle collisions," Electrical Engineering Stack Exchange. Accessed: Dec. 28, 2021. [Online]. Available: https://electronics.stackexchange.com/questions/156352/understanding-how-to-use-an-accelerometer-to-detect-vehicle-collisions

[161] "Flutter - Build apps for any screen." Accessed: Dec. 28, 2021. [Online]. Available: //flutter.dev/

[162] W. Hou, Z. Ning, and L. Guo, "Green Survivable Collaborative Edge Computing in Smart Cities," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1594–1605, Apr. 2018, doi: 10.1109/TII.2018.2797922.

[163] X. Lyu *et al.*, "Selective Offloading in Mobile Edge Computing for the Green Internet of Things," *IEEE Network*, vol. 32, no. 1, pp. 54–60, Jan. 2018, doi: 10.1109/MNET.2018.1700101.

[164] Md. S. Munir, S. F. Abedin, D. H. Kim, N. H. Tran, Z. Han, and C. S. Hong, "A Multi-Agent System toward the Green Edge Computing with Microgrid," in *2019 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2019, pp. 1–7. doi: 10.1109/GLOBECOM38437.2019.9013574.

[165] D. Zhang *et al.*, "Near-Optimal and Truthful Online Auction for Computation Offloading in Green Edge-Computing Systems," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 880–893, Apr. 2020, doi: 10.1109/TMC.2019.2901474.

[166] Z. Wu and C. Zhou, "Equestrian Sports Posture Information Detection and Information Service Resource Aggregation System Based on Mobile Edge Computing," *Mobile Information Systems*, vol. 2021, p. e4741912, Jul. 2021, doi: 10.1155/2021/4741912.

[167] Z. Chen *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, San Jose California: ACM, Oct. 2017, pp. 1–14. doi: 10.1145/3132211.3134458.

[168] S. Salkic, B. C. Ustundag, T. Uzunovic, and E. Golubovic, "Edge Computing Framework for Wearable Sensor-Based Human Activity Recognition," in *Advanced Technologies, Systems, and Applications IV -Proceedings of the International Symposium on Innovative and Interdisciplinary Applications of Advanced Technologies (IAT 2019)*, S. Avdaković, A. Mujčić, A. Mujezinović, T. Uzunović, and I. Volić, Eds., in Lecture Notes in Networks and Systems. Cham: Springer International Publishing, 2020, pp. 376–387. doi: 10.1007/978-3-030-24986-1_30.

[169] Z. Han, "Research on Sports Balanced Development Evaluation System Based on Edge Computing and Balanced Game," *Security and Communication Networks*, vol. 2021, p. e5557138, Apr. 2021, doi: 10.1155/2021/5557138.

[170] R. Massoud, R. Berta, S. Poslad, A. De Gloria, and F. Bellotti, "IoT Sensing for Reality-Enhanced Serious Games, a Fuel-Efficient Drive Use Case," *Sensors*, vol. 21, no. 10, Art. no. 10, Jan. 2021, doi: 10.3390/s21103559.

[171] "Grove_BMP280." Accessed: Dec. 29, 2021. [Online]. Available: https://github.com/Seeed-Studio/Grove_BMP280

[172] F. Sakr, R. Berta, J. Doyle, A. De Gloria, and F. Bellotti, "Self-Learning Pipeline for Low-Energy Resource-Constrained Devices," *Energies*, vol. 14, no. 20, Art. no. 20, Jan. 2021, doi: 10.3390/en14206636.

[173] H. Zhao, S. Wang, G. Zhou, and W. Jung, "TennisEye: Tennis Ball Speed Estimation using a Racket-mounted Motion Sensor," in *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2019, pp. 241–252. doi: 10.1145/3302506.3310404.

[174] "Amazon EC2 Inf1 Instances," Amazon Web Services, Inc. Accessed: Apr. 11, 2022. [Online]. Available: https://aws.amazon.com/ec2/instance-types/inf1/

[175] L. Prechelt, "Early Stopping - But When?," in *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Müller, Eds., in Lecture Notes in Computer

Science. , Berlin, Heidelberg: Springer, 1998, pp. 55–69. doi: 10.1007/3-540-49430-8_3.

[176] F. Jia, S. Majumdar, and B. Ginsburg, "MarbleNet: Deep 1D Time-Channel Separable Convolutional Neural Network for Voice Activity Detection," *arXiv:2010.13886 [cs, eess]*, Feb. 2021, doi: 10.48550/arXiv.2010.13886.

[177] R. Hebbar, K. Somandepalli, and S. Narayanan, "Robust Speech Activity Detection in Movie Audio: Data Resources and Experimental Evaluation," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 4105–4109. doi: 10.1109/ICASSP.2019.8682532.

[178] "LifeTouch - automotive UX design and engineering studio." Accessed: Dec. 13, 2021. [Online]. Available: http://lifetouch.it/

[179] "Freesound - Freesound." Accessed: Dec. 13, 2021. [Online]. Available: https://freesound.org/

[180] S. Kriman *et al.*, "Quartznet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2020, pp. 6124–6128. doi: 10.1109/ICASSP40776.2020.9053889.

[181] D. B. Paul and J. M. Baker, "The Design for the Wall Street Journal-based CSR Corpus," in *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*, 1992. Accessed: Dec. 14, 2021. [Online]. Available: https://aclanthology.org/H92-1073

[182] C. Cieri, D. Miller, and K. Walker, "The Fisher Corpus: a Resource for the Next Generations of Speech-to-Text," in *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal: European Language Resources Association (ELRA), May 2004. Accessed: Jun. 18, 2023. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2004/pdf/767.pdf

[183] J. J. Godfrey, E. C. Holliman, and J. McDaniel, "SWITCHBOARD: telephone speech corpus for research and development," presented at the Acoustics, Speech, and Signal Processing, IEEE International Conference on, IEEE Computer Society, Mar. 1992, pp. 517–520. doi: 10.1109/ICASSP.1992.225858.

[184] J. Koh *et al.*, *Building the Singapore English National Speech Corpus*. 2019, p. 325. doi: 10.21437/Interspeech.2019-1525.

[185] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, in ICML '06. New York, NY, USA: Association for Computing Machinery, Jun. 2006, pp. 369–376. doi: 10.1145/1143844.1143891.

[186] W. Xiong *et al.*, "Toward Human Parity in Conversational Speech Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 12, pp. 2410–2423, Dec. 2017, doi: 10.1109/TASLP.2017.2756440.

[187] "Multidataset-QuartzNet15x5 | NVIDIA NGC." Accessed: Dec. 14, 2021. [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/models/multidataset_quartznet15x5

[188] C. Tekur and F. Gardiner, "Nvidia {'NGC'} Deep Learning Containers," 2019, Accessed: Dec. 14, 2021. [Online]. Available: https://www.usenix.org/conference/opml19/presentation/tekur

[189] B. Ginsburg *et al.*, "Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks," *arXiv:1905.11286 [cs, stat]*, Feb. 2020, doi: 10.48550/arXiv.1905.11286.

[190] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," *arXiv:1608.03983 [cs, math]*, May 2017, doi: 10.48550/arXiv.1608.03983.

[191] "PyTorch Lightning." Accessed: Dec. 14, 2021. [Online]. Available: https://pytorchlightning.ai/

[192] "apex.amp — Apex 0.1.0 documentation." Accessed: Dec. 14, 2021. [Online]. Available: https://nvidia.github.io/apex/amp.html

[193] eric-urban, "Test accuracy of a Custom Speech model - Speech service - Azure Cognitive Services." Accessed: Jun. 18, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-custom-speech-evaluate-data

[194] C. Munteanu, G. Penn, R. Baecker, E. Toms, and D. James, "Measuring the acceptable word error rate of machine-generated webcast transcripts," in *Ninth International Conference on Spoken Language Processing*, Citeseer, 2006.

[195] "JiWER: Similarity measures for automatic speech recognition evaluation." Jitsi, Feb. 08, 2022. Accessed: Feb. 16, 2022. [Online]. Available: https://github.com/jitsi/jiwer

[196] W. Lund, D. Kennard, and E. Ringger, *Combining Multiple Thresholding Binarization Values to Improve OCR Output*. 2013. doi: 10.1117/12.2006228.

[197] "Speech-to-Text Benchmark." Picovoice, Apr. 27, 2022. Accessed: Apr. 27, 2022. [Online]. Available: https://github.com/Picovoice/speech-to-text-benchmark

[198] T. Bunk, D. Varshneya, V. Vlasov, and A. Nichol, "DIET: Lightweight Language Understanding for Dialogue Systems," *arXiv:2004.09936 [cs]*, May 2020, doi: 10.48550/arXiv.2004.09936.

[199] "Tuning Your NLU Model." Accessed: Feb. 25, 2022. [Online]. Available: https://rasa.com/docs/rasa/tuning-your-model/

[200] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-Based Models for Speech Recognition." arXiv, Jun. 24, 2015. doi: 10.48550/arXiv.1506.07503.

[201] "PyAudio Documentation — PyAudio 0.2.11 documentation." Accessed: Feb. 28, 2022. [Online]. Available: https://people.csail.mit.edu/hubert/pyaudio/docs/

[202] "PortAudio - an Open-Source Cross-Platform Audio API." Accessed: Mar. 22, 2022. [Online]. Available: http://www.portaudio.com/

[203] "num2words library - Convert numbers to words in multiple languages." Savoir-faire Linux, Feb. 23, 2022. Accessed: Feb. 28, 2022. [Online]. Available: https://github.com/savoirfairelinux/num2words

[204] V. Perera, T. Chung, T. Kollar, and E. Strubell, "Multi-task learning for parsing the alexa meaning representation language," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, in AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press, Feb. 2018, pp. 5390–5397. doi: 10.1609/aaai.v32i1.12019.

[205] "Sutton & Barto Book: Reinforcement Learning: An Introduction." Accessed: Jan. 26, 2023. [Online]. Available: http://www.incompleteideas.net/book/the-book-2nd.html

[206] Y. Gu, Y. Cheng, C. L. P. Chen, and X. Wang, "Proximal Policy Optimization With Policy Feedback," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 7, pp. 4600–4610, Jul. 2022, doi: 10.1109/TSMC.2021.3098451.

[207] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms".

[208] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms." arXiv, Aug. 28, 2017. Accessed: Jan. 28, 2023. [Online]. Available: http://arxiv.org/abs/1707.06347

[209] D. Wang, H. He, and D. Liu, "Adaptive Critic Nonlinear Robust Control: A Survey," *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3429–3451, Oct. 2017, doi: 10.1109/TCYB.2017.2712188.

[210] D. Wang, M. Ha, and J. Qiao, "Self-Learning Optimal Regulation for Discrete-Time Nonlinear Systems Under Event-Driven Formulation," *IEEE Transactions on Automatic Control*, vol. 65, no. 3, pp. 1272–1279, Mar. 2020, doi: 10.1109/TAC.2019.2926167.

[211] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization." arXiv, Apr. 20, 2017. doi: 10.48550/arXiv.1502.05477.

[212] Y. Dev, "Chevrolet Corvette 1980 Different colours." Accessed: May 30, 2022. [Online]. Available: https://sketchfab.com/3d-models/chevrolet-corvette-1980-different-colours-7e428bdb3ab54b4e9ac610e545fd9d03

[213] P. Polack, F. Altché, B. d'Andréa-Novel, and A. de La Fortelle, "The Kinematic Bicycle Model: a Consistent Model for Planning Feasible Trajectories for Autonomous Vehicles?," in *IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, United States, Jun. 2017. Accessed: Jul. 11, 2022. [Online]. Available: https://hal-polytechnique.archives-ouvertes.fr/hal-01520869

[214] R. K, "Autonomous Car Parking using ML-Agents," XRPractices. Accessed: Feb. 03, 2023. [Online]. Available: https://medium.com/xrpractices/autonomous-car-parking-using-ml-agents-d780a366fe46

[215] X. Wang, Y. Chen, and W. Zhu, "A Survey on Curriculum Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 4555–4576, Sep. 2022, doi: 10.1109/TPAMI.2021.3069908.

[216] L. Anzalone, P. Barra, S. Barra, A. Castiglione, and M. Nappi, "An End-to-End Curriculum Learning Approach for Autonomous Driving Scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 19817–19826, Oct. 2022, doi: 10.1109/TITS.2022.3160673.

[217] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, in ICML '09. New York, NY, USA: Association for Computing Machinery, Jun. 2009, pp. 41–48. doi: 10.1145/1553374.1553380.

[218] Y. Wei, H. Zhang, Y. Wang, and C. Huang, "Autonomous Maneuver Decision-Making Through Curriculum Learning and Reinforcement Learning With Sparse Rewards," *IEEE Access*, vol. 11, pp. 73543–73555, 2023, doi: 10.1109/ACCESS.2023.3297095.

[219] L. Lazzaroni, F. Bellotti, A. Capello, M. Cossu, A. De Gloria, and R. Berta, "Deep Reinforcement Learning for Automated Car Parking," in *Applications in Electronics Pervading Industry, Environment and Society*, in Lecture Notes in Electrical Engineering. Cham: Springer International Publishing, 2022. doi: 10.1007/978-3-031-30333-3_16.

[220] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical Search Techniques in Path Planning for Autonomous Driving," *AAAI Workshop - Technical Report*, Jan. 2008.

[221] J. Petereit, T. Emter, C. W. Frey, T. Kopfstedt, and A. Beutel, "Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in

Unstructured Outdoor Environments," in *ROBOTIK 2012; 7th German Conference on Robotics*, May 2012, pp. 1–6.

[222] E. Nordeus, "Self Driving Vehicle." Jan. 27, 2023. Accessed: Feb. 05, 2023. [Online]. Available: https://github.com/Habrador/Self-driving-vehicle

[223] Ó. Pérez-Gil *et al.*, "Deep reinforcement learning based control for Autonomous Vehicles in CARLA," *Multimed Tools Appl*, vol. 81, no. 3, pp. 3553–3576, Jan. 2022, doi: 10.1007/s11042-021-11437-3.

[224] R. Gutiérrez-Moreno, R. Barea, E. López-Guillén, J. Araluce, and L. M. Bergasa, "Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator," *Sensors*, vol. 22, no. 21, Art. no. 21, Jan. 2022, doi: 10.3390/s22218373.

[225] G. Brockman *et al.*, "OpenAI Gym," no. arXiv:1606.01540. arXiv, Jun. 05, 2016. doi: 10.48550/arXiv.1606.01540.

[226] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[227] NVIDIA-Omniverse, "NVIDIA PhysX." Accessed: Mar. 14, 2023. [Online]. Available: https://github.com/NVIDIA-Omniverse/PhysX

[228] A. Capello *et al.*, "Exploiting Big Data for Experiment Reporting: The Hi-Drive Collaborative Research Project Case," *Sensors*, vol. 23, no. 18, Art. no. 18, Jan. 2023, doi: 10.3390/s23187866.

[229] "ml-agents/Python-Gym-API.md at develop · Unity-Technologies/ml-agents · GitHub." Accessed: Mar. 09, 2023. [Online]. Available: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Python-Gym-API.md

[230] "CARLA ScenarioRunner." Accessed: Mar. 15, 2023. [Online]. Available: https://carla-scenariorunner.readthedocs.io/en/latest/

[231] "CARLA Map Editor." Accessed: Mar. 15, 2023. [Online]. Available: https://github.com/carla-simulator/carla-map-editor

[232] "Unreal Engine | The most powerful real-time 3D creation tool," Unreal Engine. Accessed: Feb. 07, 2023. [Online]. Available: https://www.unrealengine.com/en-US

[233] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, Apr. 1999, doi: 10.1016/S1364-6613(99)01294-2.

[234] "shap.DeepExplainer — SHAP latest documentation." Accessed: Jul. 11, 2022. [Online]. Available: https://shap-lrjball.readthedocs.io/en/latest/generated/shap.DeepExplainer.html

[235] F. Bellotti, L. Lazzaroni, A. Capello, M. Cossu, A. De Gloria, and R. Berta, "Explaining a Deep Reinforcement Learning (DRL)-Based Automated Driving Agent in Highway Simulations," *IEEE Access*, vol. 11, pp. 28522–28550, 2023, doi: 10.1109/ACCESS.2023.3259544.

[236] P. Stahl, B. Donmez, and G. A. Jamieson, "Anticipatory driving competence: motivation, definition & modeling," in *Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, in AutomotiveUI '13. New York, NY, USA: Association for Computing Machinery, Oct. 2013, pp. 286–291. doi: 10.1145/2516540.2516579.

[237] L. Lazzaroni, A. Mazzara, F. Bellotti, A. De Gloria, and R. Berta, "Employing an IoT Framework as a Generic Serious Games Analytics Engine," in *Games and Learning Alliance*, I. Marfisi-Schottman, F. Bellotti, L. Hamon, and R. Klemke, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 79–88. doi: 10.1007/978-3-030-63464-3_8.

[238] L. Lazzaroni, F. Bellotti, and R. Berta, "An Embedded End-to-End Voice Assistant." submitted for publication.

[239] R. Berta *et al.*, "Developing Deep-Learning-Based Autonomous Agents for Low-Speed Maneuvering in Unity 3D." submitted for publication.

[240] L. Lazzaroni, A. Pighetti, F. Bellotti, A. Capello, M. Cossu, and R. Berta, "Automated Parking in CARLA: A Deep Reinforcement Learning-Based Approach," in *Applications in Electronics Pervading Industry, Environment and Society*, F. Bellotti, M. D. Grammatikakis, A. Mansour, M. Ruo Roch, R. Seepold, A. Solanas, and R. Berta, Eds., in Lecture Notes in Electrical Engineering. Cham: Springer Nature Switzerland, 2024, pp. 352–357. doi: 10.1007/978-3-031-48121-5_50.

[241] G. Campodonico *et al.*, "Adapting Autonomous Agents for Automotive Driving Games," in *Games and Learning Alliance*, F. de Rosa, I. Marfisi Schottman, J. Baalsrud Hauge, F. Bellotti, P. Dondio, and M. Romero, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 101–110. doi: 10.1007/978-3-030-92182-8_10.

[242] F. Bellotti, L. Lazzaroni, A. Capello, M. Cossu, A. De Gloria, and R. Berta, "Designing an Interpretability Analysis Framework for Deep Reinforcement Learning (DRL) Agents in Highway Automated Driving Simulation," in *Proceedings of SIE 2022*, G. Cocorullo, F. Crupi, and E. Limiti, Eds., in Lecture Notes in Electrical Engineering. Cham: Springer Nature Switzerland, 2023, pp. 239–244. doi: 10.1007/978-3-031-26066-7_37.