Artificial Intelligence in Vehicular Wireless Networks: A Case Study Using ns-3

Matteo Drago
University of Padova
Padova, Italy
dragomat@dei.unipd.it

Tommaso Zugno
Huawei Technologies
Munich, Germany
tommaso.zugno@huawei.com

Federico Mason
University of Padova
Padova, Italy
masonfed@dei.unipd.it

Marco Giordani
University of Padova
Padova, Italy
giordani@dei.unipd.it

Mate Boban
Huawei Technologies
Munich, Germany
mate.boban@huawei.com

Michele Zorzi
University of Padova
Padova, Italy
zorzi@dei.unipd.it

ABSTRACT

Artificial Intelligence (AI) techniques have emerged as a powerful approach to make wireless networks more efficient and adaptable. In this paper we present an ns-3 simulation framework, able to implement AI algorithms for the optimization of wireless networks. Our pipeline consists of: (i) a new geometry-based mobility-dependent channel model for V2X; (ii) all the layers of a 5G-NR-compliant protocol stack, based on the `ns3-mmwave` module; (iii) a new application to simulate V2X data transmission, and (iv) a new intelligent entity for the control of the network via AI. Thanks to its flexible and modular design, researchers can use this tool to implement, train, and evaluate their own algorithms in a realistic and controlled environment. We test the behavior of our framework in a Predictive Quality of Service (PQoS) scenario, where AI functionalities are implemented using Reinforcement Learning (RL), and demonstrate that it promotes better network optimization compared to baseline solutions that do not implement AI.

CCS CONCEPTS

• **Networks** → **Network simulations**.

KEYWORDS

ns-3, RAN-AI, artificial intelligence (AI), Predictive Quality of Service (PQoS), reinforcement learning (RL), V2X.

ACM Reference Format:
Matteo Drago, Tommaso Zugno, Federico Mason, Marco Giordani, Mate Boban, and Michele Zorzi. 2022. Artificial Intelligence in Vehicular Wireless Networks: A Case Study Using ns-3. In *Proceedings of the WNS3 2022 (WNS3 2022), June 22–23, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3532577.3532605

WNS3 2022, June 22–23, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9651-6/22/06…$15.00
https://doi.org/10.1145/3532577.3532605

1 INTRODUCTION

Artificial Intelligence (AI) will be a key component of future 6th Generation (6G) wireless networks [10], as a means to achieve autonomous network optimization [17]. In particular, the co-design of communication systems and applications with AI in mind will allow 6G networks to learn, adapt, and support diverse services and requirements, without human intervention.

Among other areas, AI has been recognized as a promising technology in Vehicle-To-Everything (V2X) networks, to enable applications like traffic flow and congestion control, localization, platoon management, and autonomous driving [26]. For these systems to be truly autonomous, intelligent vehicles need to acquire, process, and eventually disseminate massive amounts of data generated by on-board sensors [32]. Notably, AI, in combination with machine learning (ML), can be designed to extract features from input data [11], and support complex V2X tasks like object detection and recognition [24], data compression [22], as well as tracking and trajectory prediction [3].

As far as AI/ML is concerned, the availability of data for training and optimization is essential. In this regard, experiments with real testbeds are impractical due to limitations in the scalability and flexibility of these platforms, as well as the high cost of hardware components. On the other hand, AI-based research should follow an iterative approach where modeling and validation will be cyclically performed. Therefore, computer simulations have emerged as important tools for testing the performance of AI solutions in different conditions and scenarios. Python-based simulators are among the most popular softwares for AI, thanks to desirable features like versatility and readability [21]. Most importantly, many open-source libraries providing base-level ready-to-use coding solutions to develop AI functionalities, like TensorFlow, PyTorch or Keras, are implemented in Python. When it comes to the simulation of wireless networks, however, Python simulators tend to rely on simplified assumptions on the system architecture, and have not proven particularly successful in modeling the protocol stack of complex networks. Instead, discrete-event network simulators, like ns-3 [13], are valid alternatives to analyze the performance of wireless networks in more realistic scenarios. How to integrate the two simulators and build an open software able to support end-to-end full-stack simulations, as well as network optimization via AI, is still an open challenge. An initial effort in this direction was made in [8]. However, while the authors focused on OpenAI Gym,

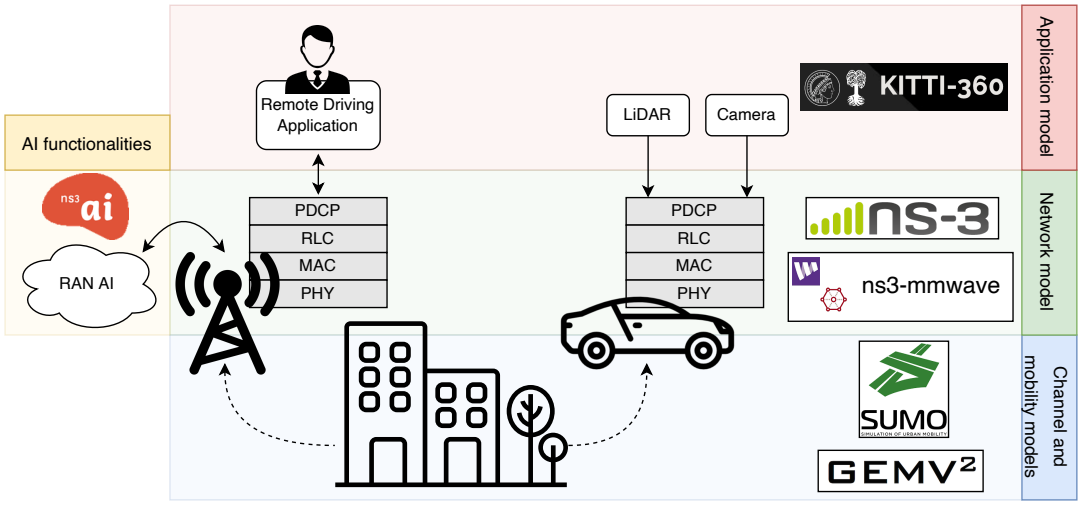


Figure 1: Overview of the Proposed ns-3 Simulation Framework/Pipeline to Incorporate AI Functionalities in Wireless Networks

a toolkit specific for Reinforcement Learning (RL) research, in the long term we expect network prediction and optimization to be transparent to the underlying cellular infrastructure and the corresponding learning schemes, which calls for more general solutions to incorporate AI functionalities into the network.

In this paper, we fill this gap and propose a simulation pipeline to design and test AI in wireless networks. In particular, we focus on a V2X scenario, in which teleoperated vehicles use a cellular connection to exchange sensor data with a remote driver. Specifically, we extended the ns-3 `mmwave` module [19], one of the most 5G-oriented frameworks to simulate wireless networks, as follows:

- We implemented a geometry-based channel model for V2X, based on GEMV^2 and Simulation of Urban MObility (SUMO) traces, that is consistent with the actual deployment of buildings and vehicles in the scenario.
- We designed a new ns-3 application to simulate the traffic flow in V2X use cases. Specifically, the ns-3 application involves the exchange of sensor data, which may be preemptively compressed and/or segmented to reduce the file size before transmission, modeled based on the Kitti multi-modal dataset [9]. The application is characterized by (i) the size of the input data, and (ii) the time periodicity at which the information is generated and exchanged, and (iii) the level of compression/segmentation of the data.
- We introduced a new entity called "RAN-AI" (the core contribution of this paper) that, connected to the Radio Access Network (RAN) and with Python-based AI algorithms, optimizes network operations.

As a case study, we validate our AI pipeline for a Predictive Quality of Service (PQoS) application [7], defined as a mechanism to predict Quality of Service (QoS) changes and provide autonomous vehicles with advance notifications to react accordingly. In this scenario, the RAN-AI collects full-stack network metrics from different components of the RAN, and implements an RL framework (first introduced in [18]) able to identify the optimal network configuration. We demonstrate that our RAN-AI is able to improve the QoS of V2X applications, compared to other baseline solutions that do not implement AI techniques.

The rest of the paper is structured as follows. In Section 2 we discuss how we integrated AI operations in ns-3. In Section 3 we validate our ns-3 implementation for PQoS. In Section 4 we conclude the paper with suggestions for future research.

2 DESIGN OF INTELLIGENT WIRELESS NETWORKS USING NS-3

Figure 1 provides an overview of the ns-3 simulation framework that we developed to integrate AI functionalities in vehicular networks. We distinguish four main components, namely (i) the channel and mobility models (Section 2.1), (ii) the network model (Section 2.2), which simulates the communication network, (iii) the application model (Section 2.3), which mimics a real vehicular application, and (iv) the intelligent network controller (Section 2.4), which provides AI functionalities to optimize the network configuration. The source code is publicly available at [25].

2.1 Channel Model

A realistic characterization of the wireless channel is of paramount importance to obtain accurate simulation results [15]. The channel model should incorporate the impact of the radio environment on the propagation of wireless signals (e.g., the presence of buildings and other blockers) and user mobility.

The approach adopted in our framework relies on OpenStreetMap (OSM) to obtain a detailed representation of the area of interest. The generation of vehicles' mobility traces is handled by SUMO [14], a popular open-source tool for the simulation of vehicular traffic. In particular, the OSM representation of the scenario is converted into a SUMO network file using the `netconvert` utility, and the generation of random routes is carried out using the script `randomTrips.py`. Finally, the wireless channel is computed using GEMV^2, a geometry-based propagation model for V2X scenarios [6], which is open source and publicly available [5]. GEMV^2 calculates both geometry-based large- and small-scale fading components of

the channel from a map of the environment and the trajectories of the vehicles, taking into account the effect of mobility, buildings outlines and foliage, and outputs the propagation loss for all the possible device pairs at each time step.

To feed the channel traces into ns-3, we created a parser able to read the GEMV^2 output files, and eventually compute the received power between a pair of devices based on the current value of the transmit power. The parser is implemented by the class `GemvPropagationLossModel`, which exploits the modular design of the ns-3 `propagation` module by extending the `PropagationLossModel` interface. The new class provides the method `DoCalcRxPower()`, which reads the traces and retrieves the current simulation time to determine the received power for the desired link(s). For an efficient processing of the Comma Separated Values (CSV) files, we used the `CsvReader` class available in the ns-3 `core` module.

2.2 Network Model

For an accurate characterization of the network components, we chose to extend the `ns3-mmwave` module, an open source an publicly available ns-3 module for the simulation of 5G networks [19].

The `ns3-mmwave` module implements models for all the layers of the 5G NR protocol stack, for both Next Generation Node Bases (gNBs) and User Equipments (UEs). The custom Physical (PHY) and Medium Access Control (MAC) layers support different NR-compliant frame structures and numerologies, multiple beamforming algorithms and scheduling policies. The Radio Link Control (RLC) and Packet Data Convergence Protocol (PDCP) layers, as well as the core network models, are based on the ns-3 `lena` module for Long Term Evolution (LTE) networks [23]. It also supports dual connectivity with LTE base stations, which enables the simulation of non-standalone 5G deployments, and Carrier Aggregation (CA) at the MAC layer. Based on these features, `ns3-mmwave` supports end-to-end full-stack network simulations with a high level of detail, and has been taken as a reference benchmark simulator for 5G scenarios.

Although this module is meant for the simulation of 5G systems operating at millimeter wave (mmWave) frequencies, we introduced some modifications to make it work at lower frequencies too. In particular, we modified the class `MmWaveHelper` to accept the GEMV^2-based propagation model presented in Section 2.1, and create gNBs and UEs without beamforming capabilities (via the `InstallSub6GnbDevice()` and `InstallSub6UeDevice()` methods, respectively).

2.3 Application Model

We focused on a teleoperated driving scenario where a Host Vehicle (HV) is controlled by a remote driver through an ad hoc driving application installed on a remote or edge server. In order to be teleoperated, the HV must be able to disseminate perception data generated from on-board sensors like Light Detection and Ranging (LiDAR) to the remote driver, that will then detect/recognize sensitive entities in the environment (e.g., cars, pedestrians, cyclists, etc.). However, the transmission of sensor data requires a considerable amount of radio resources and could potentially congest the network, preventing a smooth driving experience. To tackle this issue, data compression and segmentation are often applied in order to reduce the size of raw data prior to transmission.

In ns-3, we designed a new application module to simulate this data exchange and generate sensor data as a function of:

(1) The size of the input sensor data (in bytes);
(2) The time periodicity at which the information is generated and exchanged (typically fixed to 100 ms for LiDAR sensors, according to the device's data sheets [29]);
(3) The level of compression/segmentation for the sensor data.

In this work we consider the sensor data from the Kitti multimodal dataset, collected using a Volkswagen Passat equipped with a Velodyne LiDAR [9]. In particular, we rely on the data compression pipeline proposed in [28]. First, we infer semantic segmentation of point clouds with RangeNet++ [20]. We consider 3 segmentation levels:

- *Raw (R)*: The raw LiDAR acquisition is considered.
- *Segmentation Conservative (SC)*: Data points associated to road elements are removed, thus reducing the file size.
- *Segmentation Aggressive (SA)*: Data points associated to buildings, vegetation, traffic signs, and the background are also removed, thus keeping only the most critical items in the scene (typically pedestrians and vehicles).

Second, we compress the resulting frame using Draco [12], a software whose flexibility allows the support of 15 quantization levels and 11 compression levels. Since there are 3 segmentation levels, 15 quantization levels and 11 compression ratios, overall there would be 495 distinct alternatives; for simplicity, our application implements the 7 most representative ones, in terms of the trade-off between compression accuracy and speed, referred to as "application modes" in the rest of the paper.

To implement the features described above, we started from the application presented in [16], able to generate large data frames and automatically fragment them into bursts of packets that are then re-aggregated at the receiver, whenever possible. Specifically, we extended the `TraceFileBurstGenerator`, that allows the user to reproduce real world traffic traces, into the `KittiTraceBurstGenerator`. Using the `CsvReader` utility, already available in ns-3, we created a method to import sensor traces from the Kitti dataset (after applying compression and segmentation to the data where applicable), and save each frame information into a data structure.

As in a specific time instant of the simulation the application can operate in one specific application mode, we provided `KittiTraceBurstGenerator` with ad hoc methods to change the mode in real time. In addition, considering that a single traffic trace could incorporate and/or last for multiple scenes, the user can choose what sequence the application is replicating and decide, in case the reader reaches the end of the scene of interest, whether to loop again from the beginning of the same scene or stop the application.

Considering that a LiDAR collects 3D points periodically, we also emulate this behavior by the design of the `FramePeriod` attribute, that indicates the time interval between a frame and the following one, and is used to subsequently schedule the sending of each burst. The `BurstyApplication` and `BurstSink` classes will then take care of burst fragmentation, transmission and reception of packets, and re-aggregation of the burst.

For easy collection of statistics at the application, the module was integrated with an additional `BurstyAppStatsCalculator` utility class, which creates an output file to report how many bytes and bursts were received in a specific window of time, and the delay the received bursts have accumulated, on average, for each node.

2.4 Intelligent Network Controller

As highlighted in Section 1, the main goal of this work is to develop a framework to integrate AI functionalities in ns-3. In this section, we present a new entity called RAN-AI, installed at the gNB, which interfaces with different components of the RAN, as well as the core network, and incorporates AI capabilities with the purpose of optimizing V2X network operations. In particular, the RAN-AI collects network metrics and takes actions (also referred to as "countermeasures") to control the connected vehicles accordingly.

In our framework, the RAN-AI is responsible for:

(1) Collecting metrics from the gNB (i.e., cell-related information) and the end users.
(2) Running AI algorithms using as inputs the collected metrics. We highlight that our implementation is AI-agnostic, in the sense that it supports different AI models able to solve heterogeneous problems.
(3) Determining the actions to take in order to maximize the network performance.
(4) Communicating the actions to the relevant entities so that they can tune their behavior accordingly. In particular, in this work we allow the RAN-AI to control the end users, even though our framework does not prevent other countermeasures from being considered.

With respect to the code structure, RAN-AI functionalities are implemented by the `RanAI` class, that is in charge of integrating the features of `ns3-ai` [31], an ad hoc module to provide efficient and high-speed data exchange between Python-based AI algorithms and ns-3. Specifically, this module uses a shared memory implementation for interprocess communications, and provides a high-level interface in both Python and C++ for different algorithms, which is the reason why we integrated it in our framework.

To enable this new entity, an instance of the `RanAI` has to be installed on each gNB in the simulation environment. First, we included in `MmWaveEnbNetDevice` the attribute `m_ranAI` representing the RAN-AI instance, which is initialized by calling the class method `InstallRanAI()`. Specifically, `InstallRanAI()` (i) initializes instances that gather full-stack network statistics through the classes `MmWaveBearerStatsCalculator` and `BurstyAppStatsCalculator`, and (ii) schedules `SendStatusUpdate()`, a routine that is executed every `m_statusUpdate` seconds. In particular, `SendStatusUpdate()` allows:

(1) The gNB to collect measurements at the PHY, MAC, RLC, PDCP, and application layers, pertaining to the end users.
(2) The gNB to organize this information to be compatible with `ReportMeasures()` input requirements and, through this method, provide it to the RAN-AI.
(3) The RAN-AI to process the dataset via the AI framework.

When it comes to delivering the agent's decision (i.e., the optimal action) to an end user, the RAN-AI checks whether that user is already configured to operate as specified by the action. If not, the action must be communicated to the end user, and in our ns-3 framework we devise two possibilities:

- *Ideal notification*: We directly trigger a callback function to apply the agent's decision. No packet is transmitted, thus we do not model the impact of the transmission delays and/or communication errors/failure when changing the action.
- *Real notification*: The notification involves the transmission of a real packet towards the intended end user. Specifically, each notification packet contains (i) the agent's decision, (ii) the International Mobile Subscriber Identity (IMSI) of the user, and (iii) the Radio Network Temporary Identifier (RNTI) of the user in the cell. As such, both transmission delays and communication errors are involved in the process (i.e., the packet could be lost, leading the end user to operate sub-optimally).[1]

With respect to the AI framework running on top of the RAN-AI, we deployed multiple Markov Decision Processes (MDPs), each of which represents the behavior of a distinct end user. Specifically, we developed a novel Python module, named `CentralizedAgent`, which allows an agent to interact with multiple learning environments, and to test multiple learning configurations, thereby evaluating the performance of different AI algorithms applied to the same scenario. The `CentralizedAgent` module implements two main methods, namely `get_action()` and `update()`.

- `get_action()` takes as an input the states of the N_u end users in the target scenario, and computes N_u actions. Notably, the state of an end user is a vector that includes all the input parameters that the RAN-AI entity can collect for the given vehicle. Depending on how state and action spaces are explored, the method will return different actions.
- `update()` takes as an input a list of N_u *learning transitions*, i.e., sequences $[s_t, a_t, s_{t+1}, r_t]$ associated with each of the N_u end users in the target scenario during slot t. Specifically, a learning transition consists of (i) the state s_t of the end user at the beginning of time slot t, (ii) the action a_t performed by the end user during slot t, (iii) the state s_{t+1} at the beginning of the next slot $t+1$ and, (iv) the reward r_t that the end user receives at the end of slot t. In general, the quality of the agent's decisions increases as more data are gathered by the `update()` method.

The reward function is specific to the target application. As a case study, in this paper we will validate our AI framework for a PQoS application, as described in Section 3.1. Nevertheless, our RAN-AI implementation is transparent to the ns-3 scenario, the only dependence being how the input data is structured.

It should be mentioned that training an AI algorithm may require a huge computational effort. In our case, this challenge is exacerbated by the fact that the algorithm's input is directly taken from the ns-3 simulation running under the hood, to provide information on how the system evolves and reacts to the decisions of the agent. At every step, in fact, the agent needs to wait for ns-3 to compute and collect a set of metrics, that are then used for training. In this sense, the proposed framework supports *transfer learning*, where initial learning can be performed in a simplified (and faster)

[1] We recall that our framework is based on the `ns3-mmwave` module, that we extended to support communication at both sub-6 GHz and mmWave frequencies.

environment, with further training in a more realistic environment (in our case, in ns-3).

3 A CASE STUDY

The framework we devised can be used to study different scenarios where vehicular networks are orchestrated by AI algorithms. With respect to the components described in Section 2, `ReportMeasures()` is the most important function as it allows to collect and forward data of all users from the gNB to the ML agent. How this information is collected is completely transparent to the ML architecture that will use it as an input. The most trivial example corresponds to the design of an algorithm to estimate a QoS index (in terms of, for example, the delay, the Packet Reception Ratio (PRR), or the application throughput) in each position of a given map, and notify such prediction to the end users according to one of the methods described in Section 2.4. Then, it is up to the developer to decide what each vehicle should do with the information received from the learning framework and, considering how the module has been structured, such changes could be straightforward.

For this paper, we decided to demonstrate how our tool can be used to evaluate and optimize wireless networks by focusing on a PQoS use case. Our solution for this specific use case implements RL, but it has to be highlighted that our framework can natively support other AI approaches. In Section 3.1 we present our AI learning setup, in Section 3.2 we describe our simulation parameters, and in Section 3.3 we show some numerical results.

3.1 AI Algorithm

Overview. As a case study, we focus on PQoS, a paradigm to provide autonomous systems with advanced notifications in case of upcoming QoS changes [7]. To address this problem, the RAN-AI introduced in Section 2.4 collects network statistics at the RAN level, including the average Signal to Interference plus Noise Ratio (SINR), average Modulation and Coding Scheme (MCS) index, Physical Resource Block (PRB) utilization, mean/max/min/std of the delay and PRR experienced at the PDCP layer. Based on the collected metrics, it defines and applies network countermeasures in case QoS requirements are not satisfied. Specifically, we designed the RAN-AI to implement an AI agent, based on RL, able to identify the optimal application mode (see Section 2.3) for the end users when transmitting sensor data. The rationale behind this choice is that end users will be encouraged to select more aggressive application modes (e.g., those that apply compression/segmentation) to reduce the size of the packets to send and promote faster transmissions.

Learning agent. Based on our previous work [18], our AI/RL agent is trained according to the Double Q-learning (DQL) algorithm described in [27], which is an extended version of the classical *Q-learning* [30], even though our framework is flexible enough to support more general, i.e., non-RL, learning schemes, for example based on federated and/or distributed learning in which network optimization is transparent to the underlying architecture. Hence, whenever the `update()` function is called, `CentralizedAgent` follows the DQL procedure to perform a new training step. Our framework approximates the agent's policy by means of a Neural Network (NN), which makes it possible to handle continuous state spaces and overcome the *curse of dimensionality* phenomenon [4]. We consider a Feed-Forward NN, with S inputs and A output neurons, and implement the Rectified Linear Unit (ReLU) activation function across the different layers [2].

We highlight that the input size of the NN coincides with the dimension of the system's state, i.e., the number of input parameters of the RAN-AI entity. Instead, the output size of the NN corresponds to the number of possible actions for the agent, i.e., in our case the different application modes.

Reward function. In our target scenario, the performance of the system depends on two different aspects:

- The Quality of Service (QoS): The vehicles should satisfy QoS requirements, especially in terms of maximum end-to-end delay δ_M and minimum PRR PRR_m.
- The Quality of Experience (QoE): The transmitted data should be accurate enough to perform driving operations. For our case study, the Quality of Experience (QoE) depends on the symmetric point-to-point Chamfer Distance CD_{sym}, which is inversely proportional to the quality of the received data [28].

To incorporate both these factors, the agent reward is designed to balance between QoS and QoE via a tuning parameter $\alpha \in [0, 1]$.

Let $\hat{\text{PRR}}_t$ and $\hat{\delta}_t$ be the PRR and average delay of the vehicle at time t, respectively. If the QoS requirements are not met, i.e., $\hat{\delta}_t \geq \delta_M$ and $\hat{\text{PRR}}_t < \text{PRR}_m$, the agent reward $R(t)$ is 0, otherwise it is given by

$$R(t) = (1-\alpha)\frac{\delta_M - \hat{\delta}_t}{\delta_M} + \alpha\frac{\text{CD}_{\text{sym},m} - \hat{\text{CD}}_{\text{sym},t}}{\text{CD}_{\text{sym},m}}, \qquad (1)$$

where $\hat{\text{CD}}_{\text{sym},t}$ and $\text{CD}_{\text{sym},m}$ are the Chamfer Distance at time t and the maximum Chamfer Distance that can be tolerated, respectively.

Table 1: Simulation Parameters

Parameter	Description	Value
f_c	Carrier frequency	3.5 GHz
B	Total bandwidth	50 MHz
P_{TX}	Transmission power	23 dBm
T	RAN-AI update periodicity	100 ms
τ_s	Simulation time	80 s
N_u	Number of vehicles	$\{1,\ 5\}$
λ	Discount factor	0.95
ζ	Learning rate	10^{-4}
ϵ	Weight decay	10^{-3}
α	QoS/QoE weight	1
δ_M	Max. tolerated delay	50 ms
PRR_m	Min. tolerated PRR	1
$\text{CD}_{\text{sym},m}$	Max. tolerated Chamfer Distance	45
Layer size (inputs × outputs)		$8\times12 \rightarrow 12\times6 \rightarrow 6\times3$

3.2 Simulation Parameters

The simulation parameters are shown in Table 1.

Scenario. We consider a scenario with N_u teleoperated vehicles traveling on a real road topology, which corresponds to the portion of the city of Bologna (Italy) depicted in Figure 2. There are two main streets connected by a circular intersection, and several urban

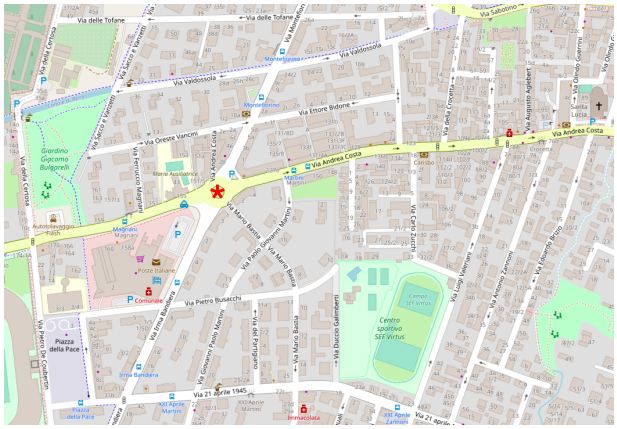


Figure 2: Our Simulation Scenario, Corresponding to a Portion of the City of Bologna, Italy (the Red Star Represents the Position of the gNB)

and sub-urban streets. This area includes both commercial and residential buildings with different heights and sizes. We consider a single gNB, operating at 3.5 GHz with a bandwidth of 50 MHz, at the center of the circular intersection at a height of 6.5 m (represented with a red star in the figure).

V2X application. Each vehicle runs an instance of the `KittiApplication` presented in Section 2.3 for streaming LiDAR data to a remote driver, and receives downlink commands for teleoperated driving operations, which is modeled as a UDP source with constant rate 0.32 Mbps.

AI/RL algorithm. The RAN-AI collects network metrics every 100 ms, and implements the RL algorithm described in Section 3.1 to optimize network operations. In our implementation, the agent's action space $\mathcal{A}$ is limited to four most representative actions, corresponding to the set of application modes $\mathcal{A} \in \{\text{C-R, C-SC, C-SA}\}$, where C-R means that data are compressed (with compression level 14) but not segmented, C-SC that data is compressed and conservative segmentation is applied, and C-SA that data is compressed and aggressive segmentation is applied. In terms of the reward function in Eq. (1), we set the tuning parameter to $\alpha = 1$ (i.e., the agent tries to maximize the QoE, as long as QoS demands are satisfied), while communication requirements for teleoperated driving are based on 5GAA specifications [1], so we have $\delta_M = 50$ ms and $\text{PRR}_m = 1$. Finally, $\text{CD}_{\text{sym},m}$ is set to 45, while $\hat{\text{CD}}_{\text{sym},t}$ depends on the application mode and increases when considering more aggressive compression/segmentation. Simulation results as a function of α can be found in [18], where we proved that decreasing α has the benefit to further improve the QoS, even beyond the QoS requirements under consideration, at the expense of some QoE degradation.

3.3 Results

To validate our framework, we compared the following policies:

- *DQL* (proposed), where at each step the agent implements our proposed RAN-AI framework.

Table 2: Average QoE $\in [0, 1]$ Performance of Different PQoS Policies, as a Function of the RAN-AI Implementation for Notifications (Real or Ideal) and N_u

PQoS policy	RAN-AI notification ($N_u = 1$)		Number of vehicles N_u (Real Notification)	
	Real	Ideal	1	5
C-R	1	1	1	1
C-SC	0.88	0.88	0.88	0.88
C-SA	0.22	0.22	0.22	0.22
DQL	0.98	0.94	0.98	0.78

- *Constant* (benchmark), where at the beginning of the simulation the end user maintains one application mode $\in \mathcal{A}$ for the whole simulation.

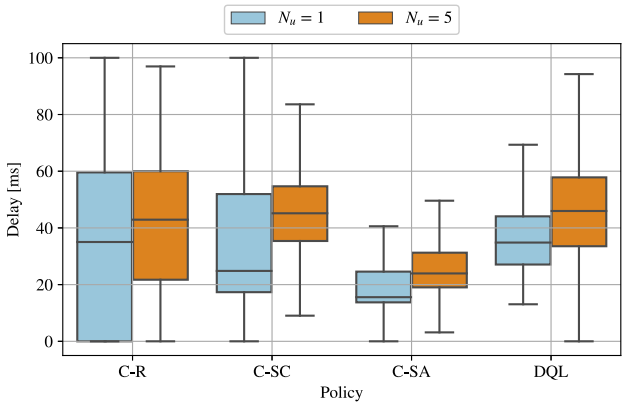
The two policies have been tested separately, and will be compared in terms of the user's (i) QoS, expressed in terms of delay and PRR at the application layer, and (ii) QoE, which depends on the Chamfer Distance. We investigate the impact of the RAN-AI implementation for delivering the agent's decisions to the end users (real or ideal) and the number of vehicles N_u.

In Table 2 we report the QoE for different PQoS policies. We observe that all the baseline solutions show constant values of the QoE, regardless of the RAN-AI implementation and N_u, as the compression level remains always fixed, while DQL shows different QoE performance depending on the adopted policy. The best QoE is achieved when transmitting raw sensor data (C-R), given that data segmentation privileges efficiency over accuracy and eventually distorts the LiDAR data before transmission. Moreover, from Table 2 we can see that introducing ideal notifications leads to QoE degradation, on average from 0.98 to 0.94. Although this behavior may sound counterintuitive, we found that, when using ideal notifications, DQL was encouraged to adopt a more aggressive behavior, i.e., the agent was encouraged to change the application mode more frequently throughout the learning process. This has the benefit to improve the QoS (as we will show in Figure 4), at the cost of a lower QoE. This is an indication that QoS and QoE should be studied together to really understand the performance of the system, as we will discuss in the following results.

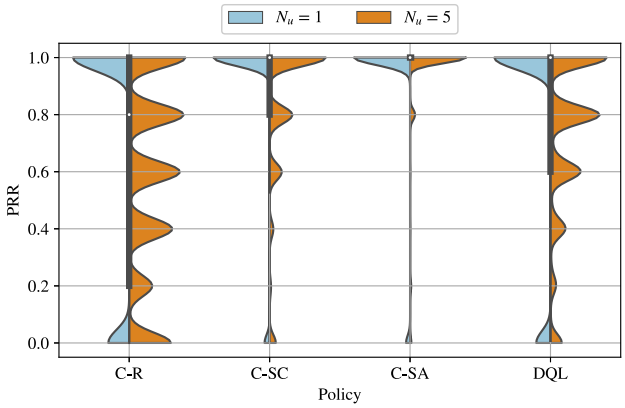
In Figures 3a and 3b we illustrate the distributions of the delay and PRR, respectively, experienced at the application layer vs. N_u. Precisely, Figure 3b represents a *split violin plot*, which depicts the estimated probability density function of the PRR for different values of N_u, for each application mode.

First, as expected, the median and the percentiles of the delay (PRR) are always higher (lower) when $N_u = 5$, due to the fact that increasing the number of vehicles may congest the communication channel, thus degrading the overall system performance. This result validates the accuracy and realism of our ns-3 framework.

In terms of QoS, C-SA outperforms any other solution, since in this configuration the data are extremely compressed and segmented before transmission, thus reducing the size of the packets to send, at the expense of a very low QoE (0.22). In turn, C-R maximizes the QoE, but results in a QoS degradation (up to 2.3× higher latency compared to C-SA). It appears clear that, in our DQL implementation, the RAN-AI tries to adapt the compression level to the conditions of the scenario via AI/RL, and achieves the best trade-off between QoE and QoS. In particular, with $N_u = 1$, DQL is able to

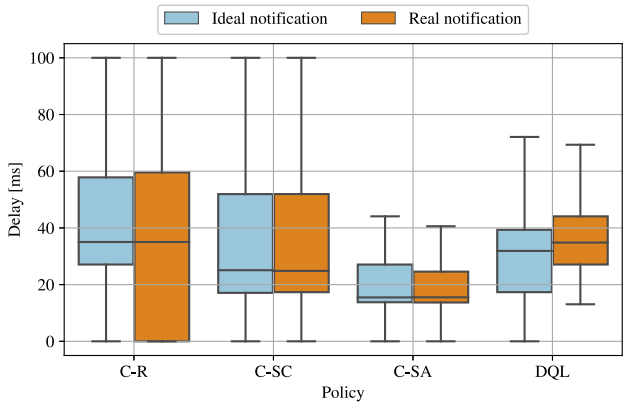


(a) Delay (Application Layer)

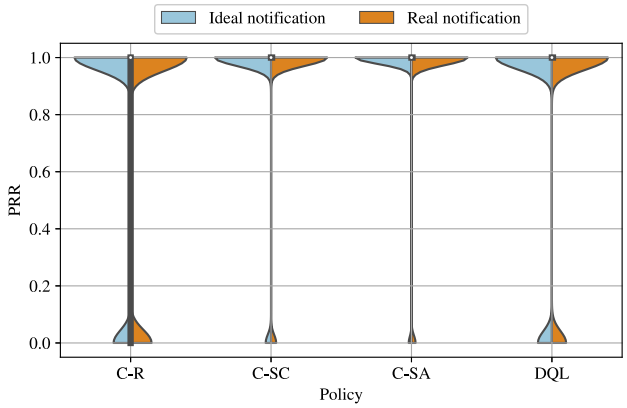


(b) PRR (Application Layer)

Figure 3: Performance of Different PQoS Policies, Considering *Real Notifications* at the RAN-AI and the Impact of the Number of Users



(a) Delay (Application Layer)



(b) PRR (Application Layer)

Figure 4: Performance of Different PQoS Policies with $N_u = 1$, Considering the Impact of the RAN-AI Overhead for Notifications

guarantee an average QoE of 0.98, while ensuring an average end-to-end delay lower than 40 ms and a reliable data delivery. With $N_u = 5$, DQL sacrifices the performance in terms of QoE (0.78) in order to ensure an average delay lower than the maximum tolerated value for teleoperated applications (i.e., 50 ms). Furthermore, DQL reduces the variability of both the delay and the PRR compared to its competitors, a critical requirement in V2X.

In Figure 4 we demonstrate that the impact of the additional overhead introduced when control notifications are transmitted from the RAN-AI to the end users is not negligible. Specifically, with *ideal notification* settings, both the delay and the PRR improve: the median delay and percentiles take lower values, while the probability of packet loss decreases, given that the transmission of real notification packets may incur additional delays and communication errors.

4 CONCLUSIONS

In recent years, V2X networks are incorporating AI as a method to analyze large volumes of data and self-optimize. While simulators like ns-3 have been popular tools to analyze the performance of wireless networks, how to simulate AI techniques and their impact on the communication stack is still an open question. In this paper we addressed this challenge, and proposed a novel framework in ns-3 able to simulate AI algorithms. To this aim, we implemented a new geometry-based channel model and application for V2X, and a new intelligent entity (called RAN-AI) for optimizing wireless networks. We demonstrated the accuracy and technical soundness of our framework in a test scenario where the network performance is controlled via PQoS. We showed from ns-3 simulations that V2X performance requirements in terms of QoS and QoE can be satisfied when the RAN-AI implements an RL algorithm for optimization. We provide the source code of the simulator at [25], in the hope that it will be useful to the broader community when implementing and evaluating AI techniques to improve wireless networks performance.

Future developments of our module will include a more extensive simulation campaign to consider a higher number of vehicles orchestrated by the same RAN-AI. From a learning perspective we will study the performance when using different values of parameter α, but also test different learning tools such as federated

learning for vehicular networks or QoS prediction based on the vehicles' positions on the map.

REFERENCES

[1] 5GAA. 2020. C-V2X Use Cases Volume II: Examples and Service Level Requirements. *White Paper* (Oct. 2020).

[2] Abien Fred Agarap. 2018. Deep Learning using Rectified Linear Units (ReLu). *arXiv preprint arXiv:1803.08375* (2018).

[3] Minjin Baek, Donggi Jeong, Dongho Choi, and Sangsun Lee. 2020. Vehicle Trajectory Prediction and Collision Warning via Fusion of Multisensors and Wireless Vehicular Communications. *Sensors* 20, 1 (Jan. 2020), 288.

[4] Richard Bellman. 1966. Dynamic Programming. *Science* 153, 3731 (Jul. 1966), 34–37.

[5] Mate Boban. 2014. *GEMV2: Geometry Based Efficient Propagation Model for V2V Communication.* http://vehicle2x.net/

[6] Mate Boban, Joao Barros, and Ozan K. Tonguz. 2014. Geometry-Based Vehicle-to-Vehicle Channel Modeling for Large-Scale Simulation. *IEEE Transactions on Vehicular Technology* 63, 9 (November 2014).

[7] Mate Boban, Marco Giordani, and Michele Zorzi. 2021. Predictive Quality of Service: The Next Frontier for Fully Autonomous Systems. *IEEE Network* 35, 6 (November/December 2021), 104–110.

[8] Piotr Gawłowicz and Anatolij Zubow. 2019. ns-3 Meets OpenAI Gym: The Playground for Machine Learning in Networking Research *(MSWIM '19)*. Association for Computing Machinery, New York, NY, USA, 113–120.

[9] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[10] Marco Giordani, Michele Polese, Marco Mezzavilla, Sundeep Rangan, and Michele Zorzi. 2020. Toward 6G Networks: Use Cases and Technologies. *IEEE Communications Magazine* 58, 3 (Mar. 2020), 55–61.

[11] Marco Giordani, Andrea Zanella, Takamasa Higuchi, Onur Altintas, and Michele Zorzi. 2019. Investigating Value of Information in Future Vehicular Communications. In *IEEE 2nd Connected and Automated Vehicles Symposium (CAVS)*.

[12] Google. 2017. *Draco 3D Data Compression.* https://github.com/google/draco

[13] Thomas R. Henderson, Mathieu Lacage, George F. Riley, Craig Dowell, and Joseph Kopena. 2008. Network Simulations with the ns-3 Simulator. *SIGCOMM demonstration* 14, 14 (2008), 527.

[14] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. 2012. Recent Development and Applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements* 5, 3&4 (Dec. 2012), 128–138.

[15] Mattia Lecci, Paolo Testolina, Michele Polese, Marco Giordani, and Michele Zorzi. 2021. Accuracy Versus Complexity for mmWave Ray-Tracing: A Full Stack Perspective. *IEEE Transactions on Wireless Communications* 20, 12 (Dec. 2021), 7826–7841.

[16] Mattia Lecci, Andrea Zanella, and Michele Zorzi. 2021. An ns-3 Implementation of a Bursty Traffic Framework for Virtual Reality Sources. In *ACM Workshop on ns-3 (WNS3)*. Online Event, US.

[17] Khaled B. Letaief, Wei Chen, Yuanming Shi, Jun Zhang, and Ying-Jun Angela Zhang. 2019. The Roadmap to 6G: AI Empowered Wireless Networks. *IEEE Communications Magazine* 57, 8 (August 2019), 84–90.

[18] Federico Mason, Matteo Drago, Tommaso Zugno, Marco Giordani, Mate Boban, and Michele Zorzi. 2022. A Reinforcement Learning Framework for PQoS in a Teleoperated Driving Scenario. *IEEE Wireless Communications and Networking Conference Workshops (WCNC WKSHPS)* (2022).

[19] Marco Mezzavilla, Menglei Zhang, Michele Polese, Russel Ford, Sourjya Dutta, Sundeep Rangan, and Michele Zorzi. 2018. End-to-End Simulation of 5G mmWave Networks. *IEEE Communications Surveys and Tutorials* 20, 3 (Apr. 2018), 2237–2263.

[20] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. 2019. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[21] Abhinav Nagpal and Goldie Gabrani. 2019. Python for Data Analytics, Scientific and Technical Applications. In *Amity international conference on artificial intelligence (AICAI)*. IEEE.

[22] Francesco Nardo, Davide Peressoni, Paolo Testolina, Marco Giordani, and Andrea Zanella. 2022. Point Cloud Compression for Autonomous Driving: A Performance Comparison. In *IEEE Wireless Communications and Networking Conference (WCNC)*.

[23] Giuseppe Piro, Nicola Baldo, and Marco Miozzo. 2011. An LTE Module for the ns-3 Network Simulator. In *4th International ICST Conference on Simulation Tools and Techniques*.

[24] Valentina Rossi, Paolo Testolina, Marco Giordani, and Michele Zorzi. 2021. On the Role of Sensor Fusion for Object Detection in Future Vehicular Networks. In *Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit)*.

[25] SIGNET University of Padova. 2022. *RAN-AI ns-3 Source Code.* https://github.com/signetlabdei/ns3-ran-ai

[26] Wang Tong, Azhar Hussain, Wang Xi Bo, and Sabita Maharjan. 2019. Artificial Intelligence for Vehicle-To-Everything: A Survey. *IEEE Access* 7 (January 2019), 10823–10843.

[27] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[28] Andrea Varischio, Francesco Mandruzzato, Marcello Bullo, Marco Giordani, Paolo Testolina, and Michele Zorzi. 2021. Hybrid Point Cloud Semantic Compression for Automotive Sensors: A Performance Evaluation. *IEEE International Conference on Communications (ICC)* (2021).

[29] VeloView. 2022. *Velodyne LiDAR Dataset.* https://data.kitware.com/#collection/5b7f46f98d777f06857cb206

[30] Christopher J.C.H. Watkins and Peter Dayan. 1992. Q-Learning. *Machine learning* 8, 3-4 (May 1992), 279–292.

[31] Hao Yin, Pengyu Liu, Keshu Liu, Liu Cao, Lytianyang Zhang, Yayu Gao, and Xiaojun Hei. 2020. ns3-Ai: Fostering Artificial Intelligence Algorithms for Networking Research. In *Proceedings of the 2020 Workshop on ns-3*. 57–64.

[32] Shan Zhang, Jiayin Chen, Feng Lyu, Nan Cheng, Weisen Shi, and Xuemin Shen. 2018. Vehicular Communication Networks in the Automated Driving Era. *IEEE Communications Magazine* 56, 9 (Sept. 2018), 26–32.