An ML-driven framework for edge orchestration in a vehicular NFV MANO environment

Nina Slamnik-Kriještorac*, Miguel Camelo Botero*, Luca Cominardi†, Steven Latré*, Johann M. Marquez-Barja*
* University of Antwerp - imec, IDLab - Faculty of Applied Engineering, Belgium
† ZettaScale, France

***Abstract*—To properly orchestrate challenging services such as those deployed for Vehicle-to-Everything (V2X) use cases, MANO systems need to be intelligent and automated. Network Function Virtualization (NFV) and Machine Learning (ML) provide opportunities for automating MANO operations, and this paper presents our Ml-enhAnced Edge Service orchesTRatiOn (MAESTRO) algorithm that makes proactive ML-driven decisions for edge service relocation to ensure Quality of Service (QoS) guarantees for V2X services. Moreover, to validate the effectiveness of our proposed solution, we have performed the experimentation using real-life testbeds for high computing and smart mobility i.e., Smart Highway and Virtual Wall, located in Antwerp and Gent, Belgium. The contribution of our paper is two-fold: i) we study the interrelation between the Key Performance Indicators (KPIs) measured at the vehicle client side, and the infrastructure metrics at the edge computing nodes and ii) we propose and evaluate an ML-based quality-aware algorithm that automates edge service orchestration to decrease average latency while guaranteeing high service availability and reliability.**

***Index Terms*—management and orchestration, NFV, MEC, ML, zenoh, testbeds, experimentation, vehicular services**

I. Introduction and Motivation

Despite the benefits of ultra-low latency, ultra-high bandwidth, and ultra-high reliability, for an increased set of use cases (e.g., autonomous vehicles, smart transport and logistics, immersive reality), 5G and beyond networks consist of a complex set of decentralized and heterogeneous devices and resources that must be integrated to provide a seamless service. This imposes challenges towards traditional MANagement and Orchestration (MANO) systems, which are manual, or closed-loop but slow, and do not scale well with service diversity and increased service usage in 5G [1]. Thus, MANO systems need support from Network Function Virtualization (NFV) and Machine Learning (ML) to automate their operations, and to make them intelligent enough to cope with the increased network complexity.

Due to a boosted demand for vehicular use cases that bring more safety in automotive operations, the Vehicle-to-Everything (V2X) services created and deployed on the network edge, i.e., leveraging Multi-Access Edge Computing (MEC), to realize the aforementioned goal require extensive broadband, efficiency, and resiliency. The automation and intelligence of NFV MANO operations of such services will enable constant monitoring of network and service performance metrics, thereby automatically translating them to decisions that result in fast service re-configurations. There are some works that study integrating ML techniques to NFV MANO operations, with the goal to enforce operations and to automate them [2–5]. Nevertheless, there is still much to be investigated when it comes to realistic experimentation and testing the true impact of ML on the optimization of NFV MANO operations, as state-of-the-art work is either based on other optimization techniques (Lyapunov optimization techniques [6]) that might be complex and lengthy for service management in V2X systems, or their evaluation is based on the simulations [3,5]. To this end, in this paper we present our **Ml**-enh**A**nced **E**dge **S**ervice orches**TR**ati**O**n (MAESTRO) algorithm that makes proactive ML-driven decisions for edge service relocation in order to ensure Quality of Service (QoS) guarantees for V2X services, thereby automating edge service orchestration. To assess the performance evaluation of this algorithm and the overall NFV MANO system, and validate it in a dynamic vehicular NFV MANO environment, we utilize the high-performance real-life testbeds, such as Smart Highway[1] and Virtual Wall[2].

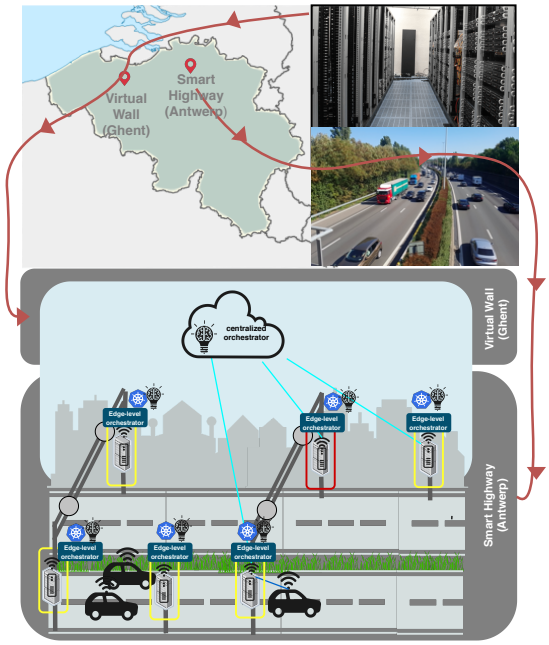


Fig. 1: The multi-domain ML-driven management and orchestration system for V2X use cases.

In Fig. 1, we illustrate the deployment of ML-driven MANO system for V2X services, with cloud and edge orchestration layers, which are enabled to autonomously operate, but also to collaborate and balance their operations towards achieving the desired Key Performance Indicators (KPIs). The MAESTRO algorithm we created is a hybrid edge service relocation algorithm, which is a Multi-Criteria Decision-Making (MCDM) algorithm based on The Technique for Order of Preference (TOPSIS) [7] and Support Vector Regression (SVR) [8]. The

[1]Smart Highway: https://www.fed4fire.eu/testbeds/smart-highway/
[2]Virtual Wall: https://www.fed4fire.eu/testbeds/virtual-wall/

SVR model is trained at the edge orchestration layer, and it uses collected data to learn the interrelation between the infrastructure and service performance metrics, and to predict the average response time of a V2X service running on the edge computing node (e.g., Roadside Unit (RSU) in our Proof-of-Concept (PoC) deployment illustrated in Fig. 1). Further, the model is used by the cloud orchestrator to proactively decide on whether the service relocation should be performed from one edge to another, in order to avoid service disruptions due to mobility and low service performance.

With the performance analysis we conducted using this realistic testbed setup, the contribution of our paper is two-fold: i) we study the interrelation between MEC infrastructure (measured at NFV Infrastructure (NFVI)) and service performance metrics that are being monitored by the cloud and edge orchestrators (measured at client side), and ii) we propose and evaluate an ML-based quality aware algorithm, i.e., MAESTRO, to automate edge service orchestration, thereby minimizing average service response time, while ensuring high service availability and reliability.

II. Intelligent and Automated Edge orchestration

A. AI-enhanced NFV MANO framework

Our NFV MANO framework consists of two layers, i.e., cloud and edge, which perform autonomous MANO operations, but enforce an interplay between them for managing orchestration decisions, or for retrieving data from distributed data engineering pipelines available in all edge domains. To enable the cooperation between different orchestration entities, certain management-level agreements need to be ensured, and more information about this type of agreement can be found in our previous work [9]. Each edge domain that consists of one or multiple edge nodes (i.e., MEC hosts) is governed by one edge orchestrator, which is, following ETSI NFV MANO framework, in charge of lifecycle management (e.g., instantiation, scaling, termination) of all MEC applications. On the other hand, the cloud orchestrator is rather in charge of global optimization in the system, thereby making less-granular decisions depending on the e.g., locations and density of vehicles on the roads for our particular real-life use case. One particular example of these decisions is service relocation from one edge domain to another, triggered by higher density of vehicles (i.e., edge service consumers) in one edge domain, mobility, or by need for optimization of CPU/memory/energy consumption in MEC hosts across edge domains.

Two MANO layers communicate with each other in the two following ways: i) via Edge-Cloud reference point, which is used to either offload decision-making tasks between two orchestrators or to pass the already taken decision, and ii) via message brokers, which exchange data in a controlled way depending on the type of ML technique that has been applied in the system, thereby using that data to either perform training or model adjustments and online learning. Thus, depending on the time-scales of optimization (global or local, i.e., edge-specific), it is required that MEC hosts can connect data to ML models in a transparent and efficient way.

This intelligent NFV MANO framework is suitable for orchestrating edge deployments of V2X services that require low-latency and high-reliability (e.g., service continuity enforced from the orchestration layer), as decision-making process is distributed, taking into account KPIs measured at the user's side. One example of edge application that might benefit from intelligent orchestration is a Back-Situation Awareness (BSA), used in our performance evaluation presented in Section IV. The BSA type of edge V2X application is a containerized application used for creating topological in-advance area-specific notifications for vehicles based on the events that occurred behind them [10]. The notifications are disseminated to different topological areas, and they contain instructions/warnings for the vehicles, while requiring some action from them to improve the driving conditions on the road, such as to clear the lane, to increase/decrease the speed, or to exit highway. These events can be either reported to edge application by specialized vehicles (e.g., emergency vehicles), or detected and reported by infrastructure sensors. In particular, applying ML to the orchestration of such V2X application enables i) optimized service instantiation, ii) learning utilization patterns for compu-

Algorithm 1: MAESTRO algorithm.

Result: Edge node selected for V2X application deployment
Edge application A_i deployed at the edge node N_k, orchestrated by Edge orchestrator E_j Start;
step 1; **while** *V2X application A_i is active* **do**
  Read KPI measurements for all edge nodes;
  Retrieve SVR model updates from Edge orchestrator O_j;
  Prepare CPU data for prediction of average response time;
  Predict $\bar{t}$ of A_i for all edge nodes E_k, $k \in (1, \dots N_E)$;
  if $\bar{t}$ *during* $\Delta T > t_{max}$ **then**
    Apply MCDM TOPSIS to make final decision for application relocation;
    Get decision;
    if *Application A_i is already deployed on the selected E_k* **then**
      go to step 1
    else
      Send notification about relocation to the source Edge orchestrator O_j;
      if *Edge orchestrator O_j accepts the decision* **then**
        Edge orchestrator O_j sends request for proactive application deployment to O_{j+1};
        if *O_{j+1} accepts the decision* **then**
          Deployment on E_{k+1} starts;
          The state/metadata is being transferred;
          O_j generates notification for vehicle edge client to reconnect from edge E_k to edge E_{k+1}
        else
          go to step 1, add flag to O_{j+1}
        end
      else
        go to step 1, add flag to O_j
      end
    end
  else
    go to step 1
  end
end

TABLE I: Parameters.

Parameter	Definition
A_i	V2X Application, $i \in \{1, \dots N_A\}$
N_A	Number of deployed V2X applications
O_j	Edge orchestrator, $j \in \{1, \dots N_O\}$
N_O	Number of Edge orchestrators
E_k	Edge node, $k \in \{1, N_E\}$
N_E	Number of edge nodes
t	Response time
$\bar{t}$	Average response time
t_{max}	maximum tolerable response time

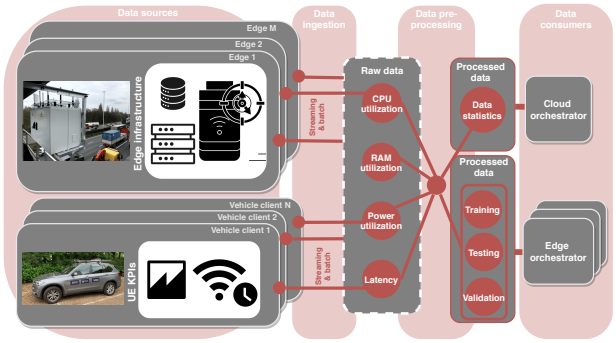


Fig. 2: Data engineering pipeline in our PoC setup.

tational resources of virtualized network services, iii) prediction models for proactive resource allocation/relocation, and iv) optimized service relocation. However, ML techniques impose some additional challenges that need to be taken into account, such as vulnerabilities in terms of i) security, scalability, and transferability, ii) high computation power (not available in resource constrained edge nodes), and iii) need for quality data to train the ML algorithms, as their performance on making decisions will depend on how close was the training data (e.g., synthetic), to the actual data used in production environments. With respect to this last point, this work aims to provide initial insights from the design, development, deployment, and experimental validation of ML-based algorithms for MANO in V2X use cases using data from real infrastructure. In this way, we can close the gap between the data generated for training and the one used for the production environments. A crucial step towards enabling intelligence and automation is a robust data collection, which is then used for training, testing, and validation purposes, and in Fig. 2 we illustrate the data engineering pipeline in our PoC deployment. The pipeline includes the following types of data sources i) infrastructure metrics, i.e., CPU, memory, and power, utilization, and ii) network-related metrics, such as latency and bandwidth. Data is being ingested into message broker instances in each edge node, and then processed by specialized helper services, i.e., MEC value-added services that perform data pre-processing, thereby making it suitable for further use.

As illustrated in Fig. 2, processed data is made available for retrieving data statistics, which might be of interest for the cloud orchestrator (data consumer) to get insights into edge infrastructure and edge service performance. Importantly, data is also exposed for training, testing, and validation, so that edge orchestrators (data consumers) can consume the generated datasets to train their local ML models. This is especially important in distributed environments where data privacy is fundamental, so the training is performed locally.

B. MAESTRO algorithm

Here we briefly describe our hybrid edge application relocation algorithm, MAESTRO algorithm (Algorithm 1, parameters described in Table I), which performs selection of a new target node to which the observed edge V2X application deployment should be relocated in order to maintain/achieve the required service response time. It works in an automated and intelligent way thanks to the MCDM mechanism that takes into account various metrics, such as CPU, memory, and power, utilization, as well as the predicted average response time for a vehicle client. The prediction is based on the SVR model that is trained at the edge orchestrator level. We use the TOPSIS class of MCDM algorithms, which is based on the comparison between all the alternatives included in the problem statement, and it is often used in solving large-scale decision-making problems in automotive industry [7]. On the other hand, we apply SVR, as a supervised learning technique, to find a function that approximates mapping from an input CPU load to average response time based on the training sample. Since edge orchestrators do not collect data from the other edge domains due to the security reasons, they cannot make decisions based on an extended perception that includes NFV infrastructure managed by other edge orchestrators. Therefore, the local SVR model, trained at the edge orchestrator level by using locally collected data, is then shared with the cloud orchestrator.

The cloud orchestrator predicts the average response time of edge V2X application for the next period of time ΔT. If predicted average response time does not exceed the t_{max}, which is the maximum tolerable response time for the edge application to provide a meaningful response (e.g., a credible record about the location and estimated time of arrival of the firetruck from behind) to the vehicle client, there is no need for relocation. The treshold t_{max} can be defined per application type, or even network slice type, so that orchestrators can correspondingly adjust their criteria. Also, this value should be low enough to enable proactive relocation, meaning that the average response time will not be degraded in the meantime while relocation is being performed. Such an automated and proactive approach is in line with the level 3/4 of autonomous networks proposed by European Telecommunications Standards Institute (ETSI) in [1], which refer to automated distinction between different kinds of services, thereby analyzing the service performance and (proactively) adjusting the service based on the changing conditions in network and infrastructure. If the decision is to relocate the edge application, the cloud orchestrator applies MCDM mechanism using the predicted value of average response time based on the CPU load for all edge domains (more details in Section IV), as well as other collected metrics (memory, power), in order to avoid relocating edge application to an edge node that e.g., experiences a high power consumption. Thus, at the same time, the cloud orchestrator is making a quality-aware decision, and trying to optimize the resource usage in all edge domains.

Following the steps provided in Algorithm 1, once the cloud orchestrator selects the edge orchestrator to be in charge of deploying relocated application instance, it sends the decision to the source edge orchestrator and triggers the relocation. If the edge orchestrator accepts this decision, it starts to proactively relocate the application to the selected target edge orchestrator. In case of stateful applications, whenever application instance is available at the target edge node, the transfer of state also needs to be performed before vehicle reconnects to it. This can be done by applying the container checkpoint and restore

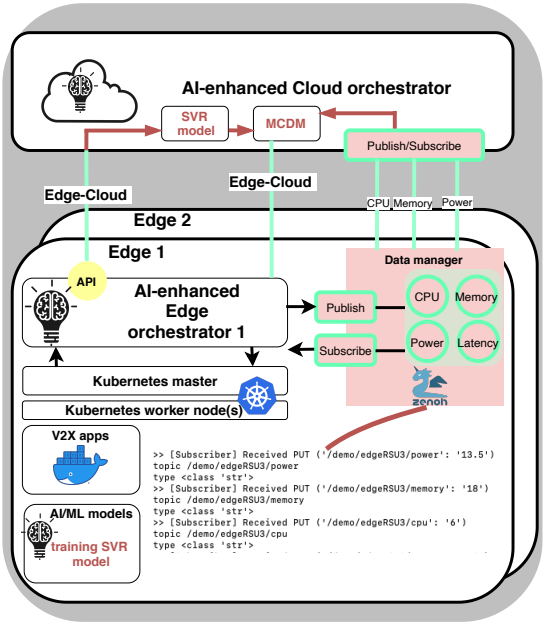


Fig. 3: PoC: The intelligent and automated management and orchestration system mapped to the real-life testbed environment.

technique [4], which involves service downtime. Otherwise, if there is no state, but a certain metadata (e.g., location and speed of the firetruck) that will be used to configure the application, then it also needs to be transferred. Once the context and/or metadata are transferred, source edge orchestrator is sending notification to the vehicle client (as described in our previous work [11]) to change the endpoint of the edge application. The client on the vehicle side needs to be configured in the way to automatically process the notification from orchestrators, and to apply the rule of configuring service endpoints. Afterwards, vehicle is reconnected to the target edge application instance, which is orchestrated by the target edge orchestrators. Finally, in case any of the edge orchestrators do not accept the decision made by the cloud orchestrator (as described in Algorithm 1), it adds certain flags to those edge orchestrators. Such flags should be further studied by the cloud orchestrator to learn about the reasons for rejecting the decisions, which should also help to retrain and reconfigure ML models. This management of ML models is out of scope of this paper, but part of our future work.

III. Proof-of-concept deployment

In Fig. 3, we illustrate the PoC that we built for conducting realistic experimentation with automated and intelligent edge orchestration of V2X services on the Smart Highway testbed, which is built along the E313 highway (Antwerp, Belgium). To build an edge network, we provide the NFVI by virtualizing computational resources in RSUs, with the help of Kubernetes (K8s). These computational resources are used for deploying V2X services, and for performing their lifecycle management. The edge orchestrator is realized as an enhanced version of a K8s master, because i) it supports cross domain operations, i.e., edge-cloud and edge-edge interaction, and ii) it is capable of training and using ML models for making intelligent decisions in an automated way. The cloud orchestrator is running on the bare metal of the Virtual Wall testbed (Ghent, Belgium). It is realized as a web server capable of i) collecting CPU, memory, and power consumption data, from edge nodes, ii) running ML models trained by the edge orchestrators, and iii) injecting decisions on the north-bound interface of edge orchestrators to instruct them to proactively migrate/relocate services from one edge to another.

To realize a data engineering pipeline we use Zenoh framework, which deals with data in motion (e.g., real-time stream of location/speed/destination of vehicles) and data at rest (e.g., historic data for computational resource utilization and energy consumption of vehicles and edge nodes). With its minimal network overhead (5B) and footprint (60kB), Zenoh is suitable for publishing data from edge nodes and vehicle clients in a transparent way, so that it can be used for training and online learning. To mimic the scenario with an increased edge service consumption, which is used in performance evaluation in Section IV, we run Locust stress test inside the vehicle. This tool is stressing the load on the edge computing node by generating REpresentational State Transfer (REST) requests to retrieve warnings/notifications from a BSA V2X application, and mimics an increase in the number of vehicles.

Finally, to realize the vehicle client that connects to edge V2X applications, we deploy the client application in the On-board Unit (OBU) of the vehicle, which utilizes a long-range 4G to exchange messages with edge services.

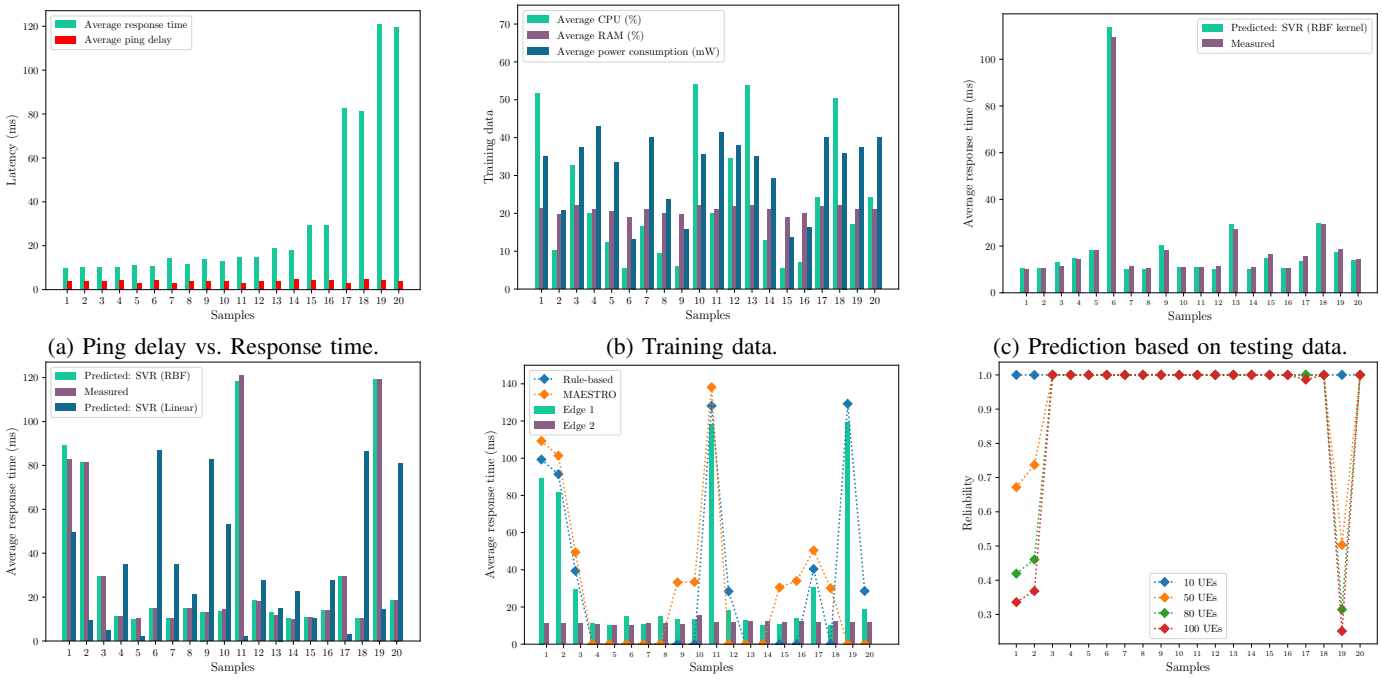
IV. Performance Experiments

A. Collecting and analyzing training data

To collect training data, we utilized the PoC described in Section III, and we gradually stressed the edge V2X deployment on the RSU *Edge 1*. While performing the stress test at *Edge 1*, we have been collecting the response time **measured at the client application in vehicle**. The overall response time consists of communication (uplink and downlink) and processing/computational delay. If the average response time presented in Fig. 4a is observed, we can see how much are communication and computational delays contributing to the overall edge service response time. Samples indicate 20 batches of successive measurements, where each of the measurements lasted for one minute, and is represented by the mean value. The stress test in our scenario caused an increase in average response time, and as we can see in Fig. 4a, communication latency remains stable despite the stress test, thus, the computational latency on the edge node is affected.

In Fig. 4b, we show the average values of CPU load, RAM load, and power consumption, in the Kubernetes cluster at the *Edge 1*. Samples of measurements correspond to the samples of edge service response time in Fig. 4a. Given that scenario indicates a sporadic stress test from sample 1 to sample 20, in Fig. 4b we can notice the changes in the CPU load. Therefore, the goal is to explore the dependency of service quality experienced by user (i.e., vehicle) on the infrastructure metrics, such as CPU load. Based on the results of this data exploration on the collected metrics, we further exploit the dependency between the CPU load and the average response time to improve the service quality experienced by user (i.e., vehicle). Other collected metrics such as memory and power consumption will be still used by the MCDM algorithm to improve the final relocation (e.g., avoiding to use an edge node with high power consumption).

As we collect both input (average CPU load) and output (average response time) data, we can apply any suitable supervised learning technique to determine the function of mapping



(a) Ping delay vs. Response time.

(b) Training data.

(c) Prediction based on testing data.

(d) Prediction based on training data.

(e) Gain achieved by relocation.

(f) Reliability.

Fig. 4: Results.

TABLE II: Results.

Model	**R-squared**	**Mean Squared Error (MSE)**	**Average difference between predicted and measured data**	**Standard deviation**	**p-value in Kruskal Wallis test**
SVR (RBF Kernel)	0.9979	2.64471	0.6651ms	1.484ms	0.9784
SVR (Linear Kernel)	0.8277	221.8706	9.985ms	11.7372ms	0.7251

input data to the expected output. In this experimentation setup, we used python[3] to apply two types of SVR depending on the kernel, i.e., Radial Basis Function (RBF) and Linear. Finally, we create two datasets, one for training, and another for testing.

B. Performance results and discussion

The performance results are shown in Fig. 4, where Fig. 4d shows the prediction of an average response time based on the training data, and Fig. 4c the prediction based on the testing data. As we notice that SVR with RBF kernel produces larger R-squared value[4] (better fits the input to output), and lower MSE (determines the accuracy of our model), this model is further used and applied in our algorithm for selecting the edge deployment. As it can be noticed in Table II, the SVR model achieves a high value of R-squared, i.e., 0.9979, and produces an MSE of 2.64471. The aforementioned result can be considered as a satisfactory level of prediction accuracy, given that average difference between predicted and measured data is less than 1ms (0.6651ms), which can be considered as negligible even for V2X applications such as BSA one that we used in this performance analysis, and described in Section II-A.

In case vehicle is moving, t_{max} is a tolerable latency for retrieving important warnings, and as it depends on the type of the use case and the average speed, t_{max} should be defined as a functional requirement (e.g., by use case developers). For the type of V2X use cases where notifications/warnings are generated and collected from edge services (as a result of processing data from sensors and other vehicles), to extend the contextual perception of a vehicle, the result we obtained can be considered as satisfactory due to the following reasons. In case the average speed of the vehicle is 80km/h, we can define t_{max} to be 15ms, as vehicle moves only 0.33m until it gets a new notification. This functional requirement needs to be studied with a more prominent attention in case of autonomous driving, or teleoperation of a vehicle, for which more ML models need to be studied and compared against each other to determine the satisfactory level of prediction accuracy. In Table II, we also show the result of the Kruskal Wallis[5] test we applied to obtain a statistical significance of the difference between measured and predicted values of average response time. As $p - value$ is larger than 0.05, this result shows there is no statistically significant difference between measured and predicted data.

[3]For this work, we have used the implementation of the SVR algorithm provided by the python library scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[4]R-squared is a coefficient of determination, a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model: https://www.investopedia.com/terms/r/r-squared.asp

[5]The Kruskal Wallis test is one of the non-parametric tests that is used as a generalized form of the Mann Whitney U test: https://www.statisticssolutions.com/kruskal-wallis-test/.

Finally, in Fig. 4e we show the result of the gain in average service response time that can be achieved by performing edge V2X service relocation in a proactive and automated way, i.e., by applying MAESTRO algorithm. First, the result shows the average response time measured at the client side for application instance running on *Edge 1*, and *Edge 2*. Second, it shows the behavior of MAESTRO algorithm against a simple rule-based algorithm, thereby examining the way they trigger service relocation. As the cloud orchestrator is constantly monitoring CPU data from different edge domains, it applies SVR model to predict the average response time for a particular type of edge V2X application. In case of MAESTRO, if predicted values of average response time in the upcoming three samples ($\Delta T = 1min$, three minutes upfront in total) is larger than t_{max}, which we consider as 15ms for a used type of service, then the cloud orchestrator applies MCDM, and potentially requests an application relocation to *Edge 2* from *Edge 1*. On the other side, a rule-based algorithm simply compares the current average response time with the threshold (i.e., 15ms), and triggers the relocation. In Fig. 4e, for each of the samples it can be seen whether these two algorithms trigger relocation for service deployment on the *Edge 1* or not. For instance, in sample 10, MAESTRO is triggering the service relocation from *Edge 1* to *Edge 2* in proactive way, which prevents the vehicle user to experience an increased response time, as in case of relocation the response time will be lower than 15ms, while on the contrary it will reach 120ms on *Edge 1*. This exemplifies how MAESTRO is outperforming rule-based algorithms, which are most-commonly used in state-of-the-art NFV MANO systems.

However, we also need to check how these decisions affect the reliability of the service. As the service reliability can be defined as a ratio of served and received requests, in Fig. 4f we show how it changes from sample to sample in case service is placed on the *Edge 1*, and if multiple users (10, 50, 80, and 100) are consuming the service. The type of service we used in this performance analysis is capable of serving three concurrent requests, i.e., if concurrently served, they achieve average response time shown in Fig. 4e. In case of 89ms response time (sample 1), the BSA application is capable of serving 33.59 requests/s (served). Clearly, the reliability of service will depend on the overall number of received requests, i.e., number of users. If there are 80 vehicles consuming the service at the same time (80 requests/s), the service reliability drops down to 0.42 in case service is consumed from *Edge 1*, which is completely unacceptable for most of the V2X services that require reliability of at least five nines (99.999%). Further, in sample 17, the reliability would drop to 0.9862 for 100 vehicles if service is not proactively relocated, which would happen in case of the rule-based algorithm as it does not proactively trigger the relocation in the 16th sample, as MAESTRO does. Same applies to sample 19, which brings completely intolerable reliability values if service is not previously relocated, as in case of MAESTRO being the one that triggers relocation in sample 18 (Fig. 4e). Such results show the true benefit of the quality-aware MAESTRO algorithm performing edge orchestration in a proactive and automated way, thereby re-attaching user from one edge to another when the algorithm triggers the relocation.

Concerning the re-attachment of vehicle client from one edge to another, in this experiment we utilized Zenoh framework to disseminate notifications from edge orchestrator to vehicle client. Furthermore, this client on the vehicle is capable of dynamically changing the service endpoint depending on the input received from edge orchestrators, by applying a new rule on its programmable data plane. As this concept is out of scope of this paper, we leave it out for our future work. Also, in our future work, we plan to i) further extend the experimentation by adding more diversity to scenarios that can happen on the highways, thereby studying the impact of mobility, ii) study service relocation costs besides service reliability, and include them in the decision-making process, and iii) analyze and examine the efficiency of managing the decisions (that can be contradictory) made at the cloud and edge orchestration layers at different timescales.

V. Conclusion

In this paper, we present and evaluate MAESTRO, an algorithm that makes proactive ML-driven decisions for edge service relocation in order to ensure QoS guarantees for V2X services. For the performance evaluation, we utilized the real-life testbeds, Smart Highway and Virtual Wall, and created a PoC, thereby validating the impact that ML models have on the edge orchestration. The results show an improved MANO operation of service relocation towards achieving required service quality, by applying an ML-based quality-aware concept that automates service relocation, while decreasing average response time and improving service reliability.

VI. Acknowledgement

This work has been performed within the European Union's Horizon 2020 projects: DAEMON (Grant Agreement No. 101017109), 5G-Blueprint (Grant Agreement No. 952189), and the Horizon 2020 Fed4FIRE+ project (Grant Agreement No. 723638).

References

[1] ETSI, "Autonomous Networks, supporting tomorrow's ICT business," *ETSI White Paper No. 40*, 2020.

[2] J. Baranda and et al., "On the Integration of AI/ML-based scaling operations in the 5Growth platform," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 105–109, 2020.

[3] L. Pacheco, D. Rosário, E. Cerqueira, L. Villas, T. Braun, and A. A. F. Loureiro, "Distributed user-centric service migration for edge-enabled networks," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 618–622, 2021.

[4] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2019.

[5] P. Soto, D. De Vleeschauwer, M. Camelo, Y. De Bock, K. De Schepper, C.-Y. Chang, P. Hellinckx, J. F. Botero, and S. Latré, "Towards Autonomous VNF Auto-scaling using Deep Reinforcement Learning," *Zenodo*, Dec. 2021.

[6] G. Shuxin, M. Cheng, X. He, and X. Zhou, "A Two-Stage Service Migration Algorithm in Parked Vehicle Edge Computing for Internet of Things," *Sensors*, vol. 20, no. 10, 2020.

[7] M. Sabaghi, C. Mascle, and P. Baptiste, "Application of DOE-TOPSIS Technique in Decision-Making Problems," *15th IFAC Symposium on Information Control Problems in Manufacturing*, vol. 48, no. 3, pp. 773–777, 2015.

[8] M. Awad and R. Khanna, "Support Vector Regression. In: Efficient Learning Machines.," *Apress, Berkeley, CA*, 2015.

[9] N. Slamnik-Krijestorac, G. M. Yilma, M. Liebsch, F. Z. Yousaf, and J. Marquez-Barja, "Collaborative orchestration of multi-domain edges from a Connected, Cooperative and Automated Mobility (CCAM) perspective," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[10] 5GAA, "Safety Treatment in Connected and Autonomous Driving Functions Report," *5GAA Automotive Association*, 2021.

[11] N. Slamnik-Kriještorac, S. Latré, and J. M. Marquez-Barja, "An optimized application-context relocation approach for Connected and Automated Mobility (CAM)," in *IEEE 5G for CAM - 5G Virtual Summit*, 2021.