IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 20 February 2023, accepted 21 March 2023, date of publication 31 March 2023, date of current version 5 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3263483

RESEARCH ARTICLE

Design and Development of a Machine Learning-Based Task Orchestrator for Intelligent Systems on Edge Networks

MARIA J. P. PEIXOTO AND AKRAMUL AZIM, (Senior Member, IEEE)
Department of Electrical, Computer and Software Engineering, Ontario Tech University, Oshawa, ON L1G 0C5, Canada
Corresponding author: Maria J. P. Peixoto (mariajoelma.pereirapeixoto@ontariotechu.net)

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

ABSTRACT This paper proposes an edge-centric workload orchestration approach that uses machine learning in a three-tier vehicular architecture (edge, cloud via roadside unit, or cloud via cellular base station). The orchestrator uses a wireless network at the edge to receive and send over-the-air requests from vehicles, considering a metropolitan network connects the entire edge structure to support the high mobility of devices, allowing vehicles to share information and resources. Additionally, suppose the edge is congested, or its resources are unavailable. In that case, cloud resources will be used via roadside or cellular networks using a wide-area network to meet the tasks' time constraints. Moreover, the proposed machine learning model uses variance-based sensitivity analysis to determine which inputs influence the model's final decision. The experiments performed on the EdgeCloudSim simulator are based on modelling computational and network resources besides the representation of vehicles. The results indicate that our approach best fits task offloading over-the-air, outperforming the comparative experiments between the one-stage(our approach) model against two-stage and random models. Furthermore, by using our one-stage model that outputs the average of the prediction interval and the variance of this interval, we can measure how confident our model is in its prediction.

INDEX TERMS Vehicle workload, cloud, edge, machine learning, mobile nodes, task offloading.

I. INTRODUCTION

Within the context of a driverless vehicle, a real-time system interacts with many other objects (cars, people and traffic lights) and systems (cameras, GPS, Lidar, Radar) in the real-world environment, which requires accurate and reliable run-time response even in failure or unknown situations. Moreover, in some real-time systems, accuracy and response time may be safety-critical, such as transportation and medical systems [1]. Thus, systems, especially safety critics, must ensure task safety (assurance of the right task and execution time) and its correct schedulability during decision-making.

In the scenario of autonomous real-time systems, we have to deal with the uncertainties arising from the operational context of these systems to obtain safe and accurate decision-making regarding the workload of these systems. Therefore, during the event identification phase, these systems often need to respond to unplanned and unknown events, which is responsible for generating tasks that need to be processed urgently. Furthermore, this whole process is responsible for generating an extra workload that needs to be allocated to some server.

One of the biggest challenges for driverless vehicles is ensuring safe operation, considering the time constraints and having to deal with a large amount of daily data. Intel, for example, estimates that each driverless vehicle produces more than four terabytes of data per day [2]. Likewise, the work by [3] states that four terabytes only refers to the generation and processing of data from an autonomous vehicle with scanning and imaging systems for each hour of autonomous driving. Based on this, we should consider factors such as

The associate editor coordinating the review of this manuscript and approving it for publication was Seifedine Kadry.

high mobility, bandwidth, latency, throughput, infrastructure, context awareness and contextual information sharing.

Considering some of the system restrictions, like mobility, bandwidth, and latency, listed above, one of the solutions was the use of cloud infrastructure, as it allows on-demand services and resource scalability [4]. Another alternative is the fog computing paradigm proposed by Cisco Systems researchers [5]. The fog computing proposal emerged amid the mobility, locality, and low latency requirements that extended cloud computing services closer to the network's border on a distributed scale [6].

Vehicle networks have to deal with highly dynamic environments where vehicle dwell time under the covers of a fixed-edge server component, such as a Roadside Unit (RSU), is short. So there is a high demand for data processing and resource sharing with low latency and increased mobility, which gave rise to the Multi-access Edge Computing (MEC) concept. Therefore, the main objective of MEC is to deploy computing resources closer to end users, with processing and storage performed closer to the origin place, further reducing the latency of requests [7].

Edge computing resources, such as RSUs, are strategically located at specific points along the road. Due to the high mobility of vehicle traffic, we consider a handover approach similar to the one proposed in [8], where the connection and resource are shared among RSUs to meet vehicle requests that move from one RSU coverage to another during the processing time of a task. For this, we use the metropolitan area network (MAN) [9], which is responsible for connecting all RSUs and sharing resources and tasks migration. Therefore, another point to be considered in this investigation is sharing context information with vehicles on the same route. Thus, in an edge-centric approach, a car can sense and detect an incident, such as an accident or temporary road closure, and send that information to an edge server. Therefore, at some point, another vehicle that accesses the edge services of an RSU and shares its itinerary promptly receives updates uploaded to the network by other vehicles about the path restrictions, allowing it to recalculate possible routes, for example.

With data processing and resource sharing between over-the-air (OTA) devices, we must highlight data security concerns from hacker attacks and malicious intrusions, as detailed in [10] and [11]. However, the cybersecurity approach in MEC networks is outside the scope of this research. Still, we stress the importance of implementing robust security measures across all system components, from edge devices to the task orchestrator. These may include data encryption, user authentication, access control and constant monitoring for suspicious and malicious activity [12], [13].

Since we have a variety of devices around us, high mobility, dynamicity of the vehicular environment, and a requirement of processing data and tasks in real-time, this paper proposes a machine learning-based task orchestrator for intelligent systems on edge networks. Our approach allows the orchestrator to manage vehicular processing and update workload through machine learning (ML) in a three-tier vehicular architecture (edge, cloud via roadside unit, or cloud via cellular base station). The advantage of our architecture is that it provides better adaptability as machine learning-based task orchestrators can adapt to changing conditions in the system, such as changes in workload or availability of resources, and adjust task scheduling and resource allocation accordingly. Furthermore, our proposed ML model is well defined in just one stage, ensuring conciseness and consistency in our approach.

Among the contributions of this paper, we can highlight the following:

1) A sensitivity analysis that the orchestrator does at the beginning of receiving data from the vehicles. This step is responsible for determining how changes in one or more input variables can affect the outputs or results of the model, helping to identify the most critical input variables that the orchestrator should focus on.
2) Our proposed ML model has only a one-stage. This ensures that the orchestrator's decision-making is faster, which helps to reduce latency between nodes. We prove this with the Quality of Experience (QoE) assessment that evaluates the service time provided by the orchestrator and the number of lost tasks.
3) A detailed evaluation of different ML algorithms to estimate the server with the best service time. Implementing the orchestrator with the most suitable ML algorithm allowed effective collaboration among context-aware intelligent systems on edge networks.
4) A validation of our proposal from the comparative evaluation between our approach and another approach with a two-stage ML model.

We organize the remainder of this paper as follows: initially, we analyzed the related works to this research in Section II. Then, we state the definition of the problem for this study in Section III. Next, Section IV describes the model setup steps we follow to design and develop our approach. Next, Section V presents the description and details of our performance evaluation, results, and discussions about them. Finally, this study's conclusion and future works are in Section VI.

II. RELATED WORK

When working with real-time autonomous systems, we have to handle several constraints, such as resource limitations, energy savings, storage, and processing capacity. As a result, choosing a suitable mechanism that correctly manages tasks is a crucial factor for the system's success. Various proposals in the literature for managing the workload on remote servers use fuzzy-based models [14], [15], [16], [17]. For example, a study presented in [14] proposes a task administration system founded on approximate fuzzy inference. The authors of that work consider the popularity of tasks being produced as a critical requirement to determine whether to perform the task locally or to download it to some remote server. The popularity of the tasks in question is calculated based

on the monitoring of recent historical data, which allows the acquisition of the demand rate of each task within a time window. Thus, the higher the demand rate, the higher the popularity of the task.

The paper [15] is another work that also uses fuzzy logic as a basis for its study. The authors use fuzzy logic to propose an edge computing infrastructure that must orchestrate the workload coming from mobile devices. Furthermore, the investigation in [15] declares that fuzzy logic is responsible for orchestration actions that consider the requirements of the network, computation, and tasks to decide where to execute the tasks in their proposal. The justification for the wide acceptance of the use of that technology is that fuzzy logic is a well-known method to deal with uncertain systems in the face of rapid changes [18].

Although several related works use fuzzy logic as a basis, the difficulties in establishing rules correctly, the need to perform numerous simulations and tests, and also the fact that there are no precise mathematical definitions [19] are disadvantages that need to be pointed out. Due to this, works such as the ones developed in [20], [21], and [22] highlight the importance of approaches focused on intelligent offloading. Furthermore, the authors of [20], [21], and [22] emphasize that due to random and uncertain contexts that vary over time, decision-making scenarios have elevated complexity to be optimized using traditional approaches, such as game theory and fuzzy logic. Consequently, that study recommends using artificial intelligence based on machine learning for multi-access edge computing problems, with the justification that the edge network can self-optimize and self-adapt in the constitution of an intelligent decision-making system.

Inspired by the intelligent edge communication works, researchers at a university in Istanbul, Turkey, published a study [23] that uses machine learning as the basis of a workload orchestrator for vehicular edge computing work. That study uses supervised learning and two-stage architecture for the orchestrator's decision-making. In the first step, a classification indicates whether the target device has a chance of success or failure. Thus, only for cases where there is a prediction of success, the orchestrator moves to the second ML step, which estimates the service time in that target device. In this way, the orchestrator chooses the remote server with the lowest expected service time. Unfortunately, despite using machine learning, the proposal of [23] becomes very extensive and expensive since it is necessary to train six different models with different input features for each of those models.

Another research paper about intelligent offloading [24] introduces a decision-making model based on Reinforcement Learning (RL). That work categorizes offloading options into three levels, giving the RL-based decision maker the ability to select between the edge server closest to the source of demand, the best edge server based on factors like execution time and energy consumption, or a cloud server. Nonetheless, additional information is required to comprehend the testing process in that study [24], and the results necessitate more detailed data regarding the success or failure rate of tasks.

III. PROBLEM STATEMENT

Considering that some specific systems, especially connected autonomous vehicles, have limited processing capacity locally, they can take advantage of the resources on remote servers that can support processing their workload in many situations. In this sense, this paper proposes a workload management system that is processed remotely and not on-board. The offloading management of the tasks is the responsibility of a context-adaptive orchestrator at runtime. This orchestrator, located at the edge, predicts each available server's service time (ST). The task processing possibilities can be at the edge, in the cloud via Roadside Units (RSU), or in the cloud via Cellular Base Station (CBS). The service time of the job offloaded is calculated using the following equation:

$$ST = communication\_cost + \frac{C_\tau}{C_{server}} \tag{1}$$

where C_τ is the computation required to complete a specified task and C_{server} is the computational capability of each server which can be at the edge, cloud via RSU or cloud via CBS. The $communication\_cost$ is calculated between edge and cloud via RSU or between edge and cloud via CBS according to the following equation:

$$communication\_cost = NL + \frac{D}{B} \tag{2}$$

where NL is the network latency between the origin node and destination node, D is data generated by the vehicle, and B is the bandwidth between the origin node and destination node. Then, the orchestrator chooses the server to process a task assisted by an ML model to infer which server has the lowest estimated service time. Machine learning (ML) is already well-known for pattern detection, and decision-making with supervised learning [25]. Therefore, the orchestrator must guide its choices based on time constraints, available processing power, bandwidth and network latency.

We have a given the task τ to execute on a remote server S on node $N = \{n_{E_1}, n_{RSU_1}, n_{CBS_1}, \ldots, n_{E_p}, n_{RSU_p}, n_{CBS_p}\}$, where n_E represents the edge node, n_{RSU} the cloud node via RSU and n_{CBS} the cloud node via CBS. Therefore, the goal is to define the machine learning-based orchestrator O to assign τ to n that belongs N concerning finding the shortest estimated service time for processing the task τ to minimize the processing time response R to the autonomous vehicle that requested the processing of the task τ.

Accordingly, we propose a concise one-stage ML-based task orchestrator to solve the limited on-board processing capacity and minimize the response time for intelligent vehicles. During the autonomous agents' workload orchestrating, the orchestrator counts on the support of ML to estimate the services time that that workload would take to complete at the edge, cloud via RSU or cloud via CBS. Based on this estimation, the orchestrator chooses the target device with the lowest estimated service time. Also, the orchestrator receives

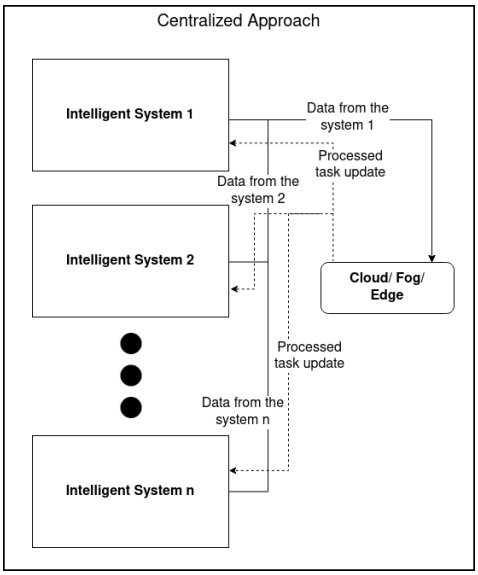

FIGURE 1. Centralized approach for remote workload processing.

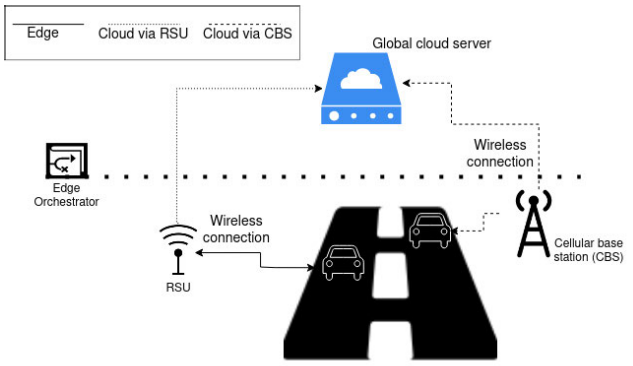

FIGURE 2. Detailed centralized approach with edge orchestrator.

numerous input features, and we cannot use all of them to estimate the service time, as it would cause the orchestrator to overload. Therefore, we use sensitivity analysis to decide which input features of the model are decisive for predicting the required service time.

IV. METHODOLOGY

Real-time intelligent systems are required to process and interpret numerous data simultaneously, demanding high-performance computing and rigorous real-time responses to requested tasks. In the case of autonomous vehicles, we deal with limiting factors such as battery life and constricted computing power that restricts the processing load supported on-board. In this scenario, we must take advantage of external devices that can support the processing performed by the intelligent system in real-time. Furthermore, with over-the-air communication, the vehicle can update context information regarding the places where it will be soon. This context information can be loaded in real-time by other cars at that location. Therefore, we can count on computing assistance in the cloud, fog or edge. For example, Figure 1 presents a centralized architecture that each intelligent system sends its workload to be processed in a digital infrastructure (cloud, fog or edge), which updates the tasks those systems are supposed to perform in real-time.

The features of fog and edge computing have attracted extensive interest from academia and industry, as they significantly reduce the network latency, concentrating computing services closer to consumers [26]. On the other hand, cloud servers have unrestrictedly more extensive processing and storage capacity than in the edge, and fog [26]. Therefore, our system requires low network load, low execution and response times, and occasionally high processing power. Due to these characteristics, it is necessary to decide on the processing target device based on the characteristics of each task and the moment it arrives at the orchestrator, as we must consider vehicular network context situations. Thus, this study intends to prioritize edge usage due to previously listed advantages. Also, a recent study [27] shows that edge-offloaded services can emit less CO2, which is highly relevant considering the current climate change situation on our planet.

The communication system established by the proposed orchestrator model has a significant impact on the edge-centric connected autonomous vehicular system. As our proposal predicts the server's service time, we always consider the latency between the task point of origin and destination for the orchestrator's decision-making. Thus, the lowest latency will be the best option [28] depending on the criticality of the task to be executed. Therefore, as some papers suggest [29], [30] and considering the available resources, the recommended communication latency should be on the order of 1 ms. Then, we need to consider the task deadline because if the server's predicted service time is less than or equal to the task deadline, the communication latency meets system requirements.

Figure 2 presents this paper architecture in more detail. First, characterizing our systems as autonomous vehicles, we offload the tasks using a wireless local-area network (WLAN) at the edge, which has an ML-based orchestrator that will estimate the best target device for that task processing, whether on edge or cloud. If the chosen target device is in the cloud, the best service time related to the task sending should be calculated, whether using RSU or CBS. Communication between the edge and the cloud is established using a wide-area network (WAN). Finally, the orchestrator returns the result of the processed task to the vehicle.

Figure 3 presents the workflow of our edge orchestrator. We start with connected autonomous vehicles sending the tasks to the orchestrator. Then the shortest service time to process the task is responsible for defining the node where the task will be processed. After being processed, the task result is sent back to the orchestrator, which sends it back to the autonomous requesting vehicle. Besides workflow,

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

Algorithm 1 How Our Orchestrator Handles Task Offloading

Require: **INPUT** ← task

var *device* : *integer*;
var *selectedVM* : *VM*;
var *predictedServiceTime* : *double*;

Initialize servers options list ← [
EDGE_DATACENTER,
CLOUD_DATACENTER_VIA_RSU,
CLOUD_DATACENTER_VIA_CBS];

for each server option in list **do**
predictedServiceTime = *ML_model*(*selected_input_features*);
end for

if Edge ← shortest predictedServiceTime **then return** device ← EDGE_DATACENTER;
else if Cloud_via_RSU ← shortest predictedServiceTime **then return** device ← CLOUD_DATACENTER_VIA_RSU;
else
return device ← CLOUD_DATACENTER_VIA_CBS;
end if

if device = CLOUD_DATACENTER_VIA_RSU **or** device = CLOUD_DATACENTER_VIA_CBS **then**
var CloudHosts ← get the number of cloud hosts available;
var hostIndex ← get an index based on CloudHosts **and** user location;
var vmIndex ← get an index based on hostIndex;
return selectedVM with hostIndex **and** vmIndex;
else
var EdgeHosts ← get the number of edge hosts available;
var hostIndex ← get an index based on EdgeHosts **and** user location;
var vmIndex ← get an index based on hostIndex;
return selectedVM with hostIndex **and** vmIndex;
end if

OUTPUT ← selectedVM

Since vehicles have several sensors and send different data to a remote server, we need to select which of these data are essential for what is being processed at the moment. Thus, if the vehicle sends a task related to navigation to be processed over-the-air, it does not matter at that moment the data received and related to the infotainment system, for example. Thus, according to our model presented in Figure 4, one of the first steps at the beginning of our approach is selecting the most relevant input features using sensitivity analysis.

Sensitivity analysis is critical because it indicates how much each uncertain parameter contributed to generating output uncertainty. Then we chose to do a variance-based sensitivity analysis to determine how much the input variation influences the output [31], [32]. Thus, we used the first-order Sobol indices and the total Sobol indices. First-order Sobol indices identify the input parameters that have the most significant influence on outcome variability. Total Sobol indices assess the impact of interactions between the various inputs on outcome variability [33], [34], [35].

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

$$S_i = \frac{Var(E(H|P))}{Var(H)} \tag{3}$$

where S_i is the sensitivity index, H is the output resulting from the model with uncertain input parameters $P = [P_1, P_2, P_3 \ldots P_n]$. Thus, if we have a low sensitivity index, which can vary with the range [0, 1], we will find that the variance in this parameter will have little effect on the variance of the final result. Therefore, if the parameter's sensitivity index is high, any variation in this parameter will lead to significant variations in the model's output.

The total Sobol index for the parameter P_n is defined according to the following equation:

$$S_{\tau i} = 1 - \frac{Var(E(H|P_{-n}))}{Var(H)} \tag{4}$$

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

Initially, we had 12 different inputs to estimate the service time for the chosen server. Thus, as shown in Figure 5, we had the following features: Decision, VehicleLocation, SelectedHostID, TaskLength, WANUploadDelay, WANDownloadDelay, GSMUploadDelay, GSMDownloadDelay, WLANUploadDelay, WLANDownloadDelay, AvgEdgeUtilization and NumOffloadedTask.

Furthermore, we performed a sensitivity analysis to determine the importance of each of these 12 features. Thus, the most priority feature to estimate the service time of a server is WLANUploadDelay, followed by Decision, TaskLength, NumOffloadedTask, then SelectedHostID and, finally, GSMUploadDelay. The other features that were not chosen did not present relevance to impact the machine learning model decision, as we can notice in Figure 5.

We used the sensitivity analysis explained above and extracted our experiment's main input features. Table 1 brings

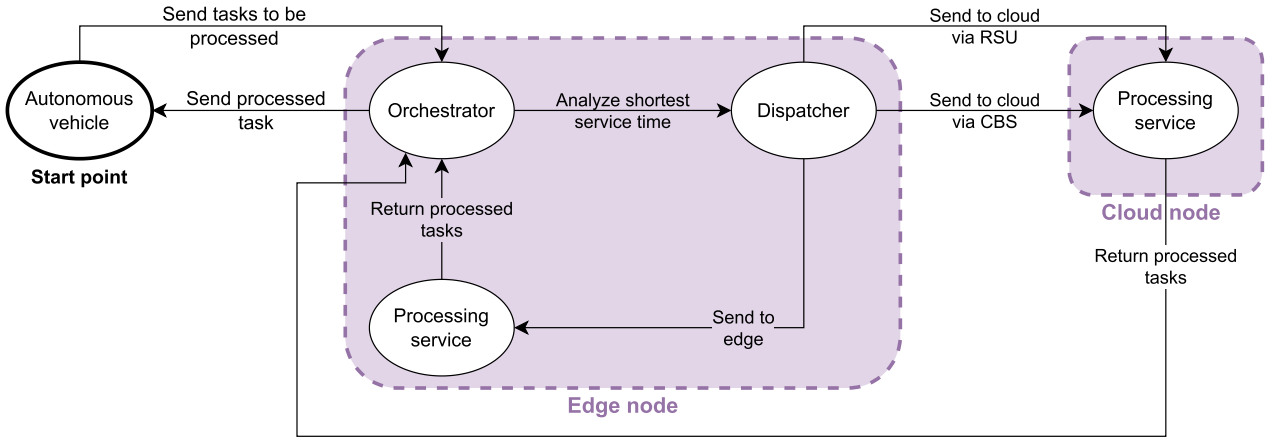

FIGURE 3. Edge orchestrator workflow.

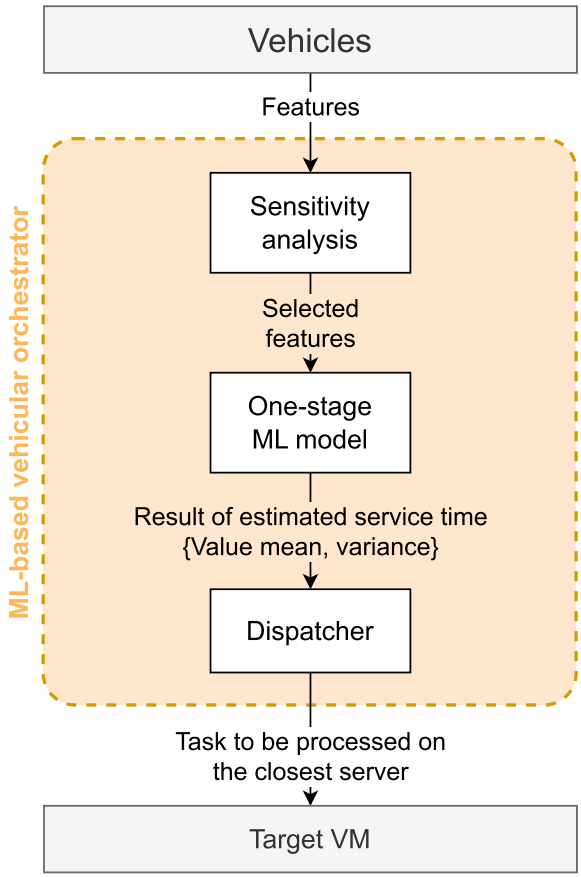

FIGURE 4. Our one-stage proposed architecture.

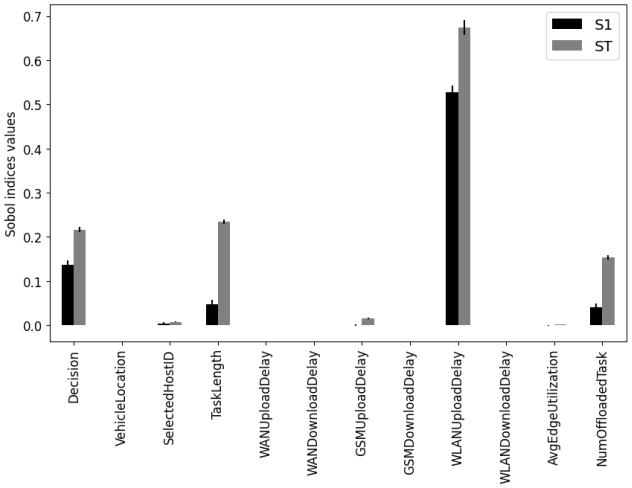

FIGURE 5. (S1 = first-order Sobol index, ST = total Sobol index) - Variance-based sensitivity analysis to define 'Selected features'.

TABLE 1. Details of features selected as input.

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

the specification of these features with a description for each one.

Working with sensitivity analysis and feature selection has been demonstrated to be essential for the system's success. Our goal is to allow our system to work only with the inputs essential for its decision. In the literature, many authors work with the prior selection of inputs in their systems. For example, the paper [36] points out that including excessive input

variables that are not relevant to the target variable can not only complicate the estimation and selection of the model but also impair its performance. Similarly, the work [37] says that incorporating an excessive number of variables in a model can produce significant outcomes but may not hold true in the current population context.

V. EXPERIMENTAL ANALYSIS

We used the EdgeCloudSim simulator [38] to analyze our centralized approach that uses machine learning-based load orchestration. We have chosen that simulator because it simulates computational and network resources inherent to edge computing and supports cloud computing as well. Therefore, the most vital point of EdgeCloudSim is the simulation of mobile devices, making it possible to simulate WLAN (wireless local-area network) and WAN (wide-area network) networks. Those main simulator features are essential for us, considering that we are working in the autonomous vehicle domain, which has high dynamicity.

To carry out our experiment, we used as a basis the work developed in [23]. The authors of [23] also propose a centralized, multi-tier architecture for vehicular networks, with workload orchestration based on machine learning. However, the two-stage structure based on ML presented (the red highlight in Figure 6) may be simplified and improved to achieve better results. The suggested two-stage architecture, shown in Figure 6, is divided to respond to three tiers: i) edge; ii) cloud via RSU (roadside unit); iii) cloud via CBS (cellular network); and each of these layers needs two ML models. The first model is to classify whether the unloaded task will succeed or fail, and the other model calculates the service time of the tier predicted to be successful in the offloading process. In total, the two-stage architecture needs to train six different machine learning models to make its predictions. In addition, each of the six models uses various input features, and the reasons for such a choice in the two-stage proposal are unclear [23].

As exemplified in Figure 4, we have our one-stage proposed architecture. First, we transmit the vehicle output to the orchestrator, where we initially perform the sensitivity analysis. Then the selected features go to our one-stage ML model responsible for estimating the service time in interval ranges for each tier: edge, cloud via RSU or cloud via CBS. We have an output with the interval mean and its variance based on the predicted ranges for each tier. Based on the variance, we can estimate the degree of uncertainty of the model regarding its prediction. Thus, the greater the variance, the more uncertain the model prediction. From this, the choice for the offloading process is made considering the lowest value of the mean and the lowest variance value. Therefore, in addition to being concise and consistent, our proposal considers the forecast's uncertainty, which is not made in absolute numbers, but in intervals that estimate the minimum and maximum of the service time may take. Consequently, we always consider the service time estimation with the most minor predicted interval for offloading.

In our experiment, the EdgeCloudSim simulator was configured to generate data during a vehicular mobility simulation, in which 100 to 1800 vehicles were randomly distributed at the beginning of the simulation with dynamic speeds, varying in each segment along the road to represent different traffic densities. Figure 7 represents the number of edge servers provided in the simulation, a total of 40 points distributed in the RSUs identified on the road by the ''places IDs''. This road was modelled in a circular path so that the number of automobiles in the simulation remained constant. From Figure 7, we can see the distribution of vehicles by area as the number of these vehicles grows. The red areas are the hotspot locations that represent the occurrence of traffic jams. For all experiments, we used a notebook with the following specification: AMD Ryzen 7 3800X 3.9 GHZ 8-core processor, 32GB DDR4 3600, SSD NVMe 500GB, NVIDIA® GeForce RTX™ 2060 8GB and Ubuntu-based Linux 20.04 operating system.

The data obtained from the simulation are related to three applications: a navigation app, a danger assessment app, and an infotainment app. Furthermore, the produced data are as follows: Decision (edge, cloud via RSU or cloud via CBS), Result (success or failure), ServiceTime, ProcessingTime, VehicleLocation, SelectedHostID, TaskLength, TaskInput, TaskOutput, WANUploadDelay, WANDownloadDelay, GSMUploadDelay, GSMDownloadDelay, WLANUploadDelay, WLANDownloadDelay, AvgEdgeUtilization and NumOffloadedTask. Therefore, considering these data, we used only Result-related entries with success values, which gave us 11,533,902 entries. Furthermore, Table 2 presents a summary of the main characteristics defined for each of the applications used.

The first part of our proposal is related to the selected features to use as input entries for our ML model stated in Figure 4. For this, we use variance-based sensitivity analysis, with the first- and total-order Sobol indices [31], to determine the inputs that most influence the ML model's final decision. From that, we select the most relevant input features. As Figure 5 shows, the main features chosen for our experiment are Decision, SelectedHostID, TaskLength, GSMUploadDelay, WLANUploadDelay, and NumOffloadedTask.

In a comparative study, we trained six models precisely as specified in paper [23] for the two-stage architecture. First, we prepared three multilayer perceptron (MLP) models for each tier (edge, cloud via RSU or cloud via CBS) to classify success and failure cases. The features used by the authors in [23] for each model were divided as follows: the edge classifier used NumOffloadedTask, WLANUploadDelay, WLANDownloadDelay, TaskLength and AvgEdgeUtilization as input; the cloud classifier via RSU used NumOffloadedTask, WANUploadDelay, and WANDownloadDelay as input; the cloud classifier via CBS used as input NumOffloadedTask, GSMUploadDelay, and GSMDownloadDelay. Then, to predict the service time for task offloading, three linear regression models also used different entries: the edge regressor used TaskLength and

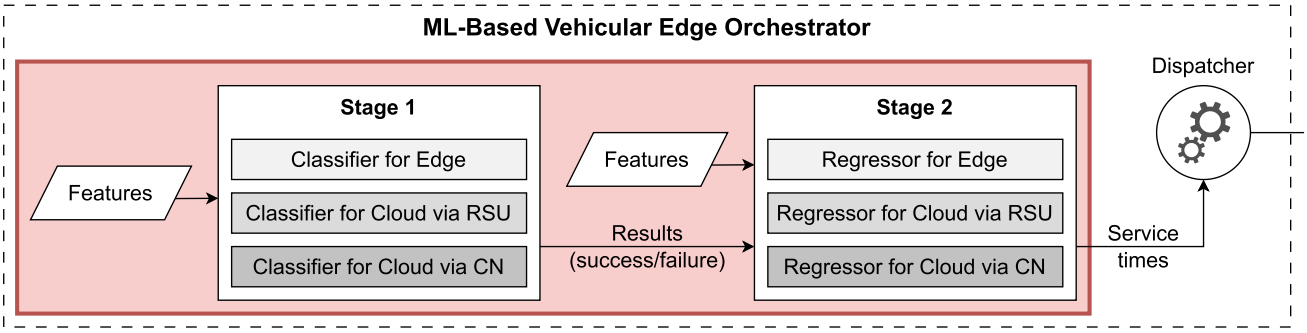

FIGURE 6. Two-stage architecture proposed in [23].

TABLE 2. Defined characteristics for each of the applications used.

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

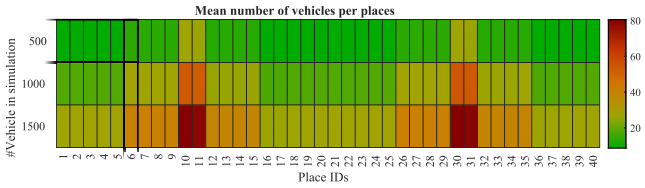

FIGURE 7. Distribution of vehicles by area.

AvgEdgeUtilization as input; the cloud regressor via RSU used TaskLength, WANUploadDelay and WANDownloadDelay as input; the cloud regressor via CBS used TaskLength, GSMUploadDelay and GSMDownloadDelay as input. The regression models only estimated the service time of the classifier whose prediction was equal to success, allowing the orchestrator to choose the shortest service time among the estimated ones.

In the one-stage architecture proposed in this work, we studied machine learning algorithms suitable for solving our regression problem. Therefore, as shown in Table 3, we analyzed the following algorithms: linear regression, multilayer perceptron, M5Rules, support vector machine (SVM) and random forest. Having the cross-validation results of Table 3 as a basis, we chose the Random Forest algorithm that presented the best performance and, consequently, is the one that best fitted the problem after our investigations.

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

We present in Figure 8 the result of the performance evaluation for the two approaches to predict the task offloading service time. We use the R2 score, mean squared error (MSE) and mean absolute percentage error (MAPE) as the proposal's quality evaluation metrics. As shown in Figure 8, the result of our approach, the one-stage architecture, significantly outperforms the results of the two-stage architecture. The one-stage architecture's R2 is incredibly better than the two-stage architecture's R2. Furthermore, the two-stage architecture's MAPE has a significantly poor result, and its MSE is also worse than the one-stage architecture's MSE in all three tiers. The results in Figure 8 demonstrate that in addition to our proposal being concise, the results of the one-stage architecture are still much more promising.

After these preliminary results, we used the EdgeCloudSim simulator to perform task-offload simulations on a remote server managed by the ML-based orchestrator. Therefore, we compared the performance of our one-stage orchestrator, a two-stage orchestrator by [23], and a random orchestrator. The experiments performed on the EdgeCloudSim simulator were based on modelling computational and network resources and the representation of mobile vehicles. We use the random orchestrator to select the offload server randomly, so the probability of selecting a specific target server is equal to the possibility of choosing any of the other available servers. A random technique is used to represent the worst-case scenario.

TABLE 3. Performance evaluation with 10-fold cross-validation.

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

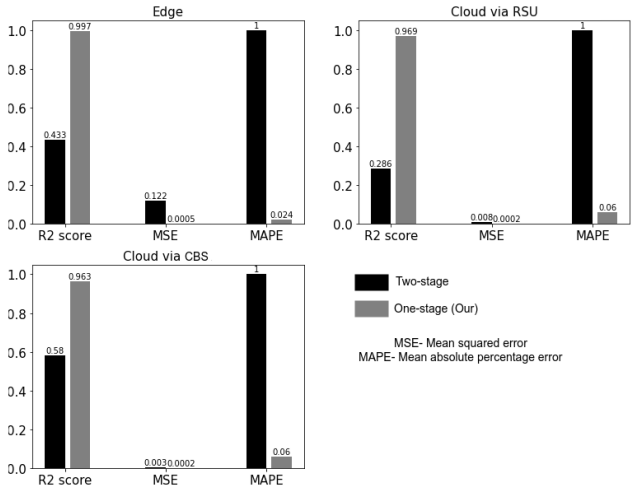

FIGURE 8. Comparison of our proposal with the proposal presented in [23].

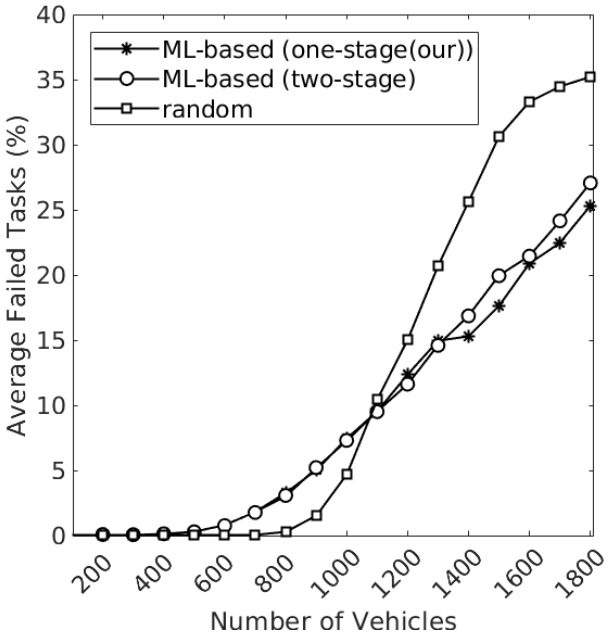

FIGURE 9. Number of tasks that failed during simulation in EdgeCloudSim.

In the simulated vehicular network, vehicles had some tasks that did not need to be processed locally, so sending them to remote servers on the edge or cloud is possible. Cloud servers can be accessed in two different ways, through RSU or CBS. Therefore, the orchestrator is responsible for choosing the remote server based on the computational load of the task and the execution time estimated through machine learning. The base configuration of the simulator used in our experiment was the same as described in the paper [23]. Thus, the vehicle applications used to generate the offloaded tasks had different characteristics regarding the arrival time, duration, and size of the upload and download data.

The results of the experiments performed with the EdgeCloudSim simulator are presented below. In these results, we can visually compare the performance of the approaches following different criteria. For example, Figure 9 demonstrates the average failure rate of tasks according to the number of vehicles in the simulation. In this way, our one-stage proposal outperforms the other two approaches. Initially, the two ML-based model starts with the same margin of task failures. Then, as the number of vehicles increases and passes 1,200, the one-stage model starts to reduce the average number of failed tasks and consolidates as the best model.

Figure 10 shows the overhead suffered by the orchestration algorithms as the number of vehicles in the simulation increases. The random model has little overhead because it pushes random choices, making decisions fast, and not generating overhead, but there is no service quality guarantee. The other two models, based on ML, need time to make their decisions, which results in overhead. However, our one-stage proposal presents the best results, considering that it also has the lowest average of failed tasks.

Figure 11 illustrates the average network delay at the time of operation of each approach. The random model has many inconsistencies. However, the two ML-based models, one-stage and two-stage, have the same average network delay during their performance, demonstrating that both have similar quality in this criterion.

Figure 12 presents the simulation time in minutes for each orchestrator. Based on this, the random algorithm has the shortest simulation time because of the arbitrary decisions made with no elaborated rules. The other two algorithms spend more simulation time. However, the one-stage model still has a significantly better simulation time than the two-stage model.

Overall, the results indicate that our approach best fits what is proposed, having outperformed the experiments. In addition, we can save computational resources when conducting sensitivity analysis, as there is no need to process all the data

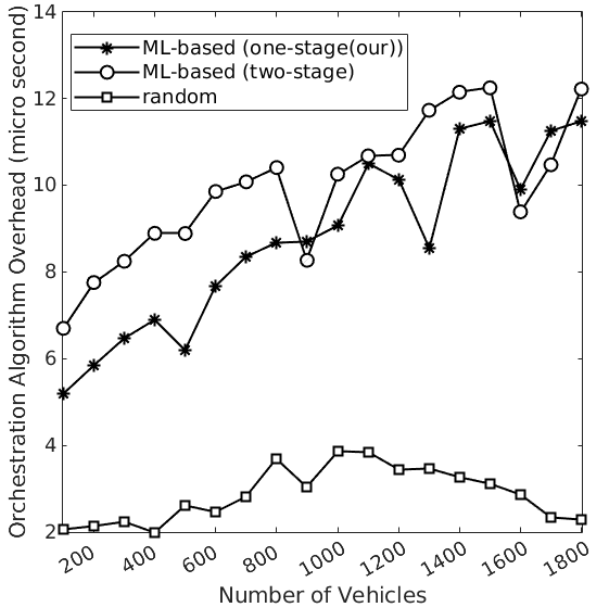

FIGURE 10. Orchestration algorithm overhead during simulation in EdgeCloudSim.

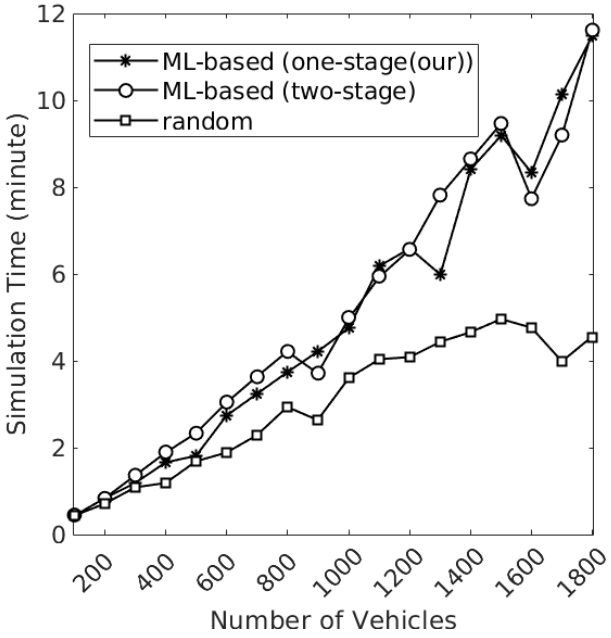

FIGURE 12. Algorithm simulation time duration in EdgeCloudSim.

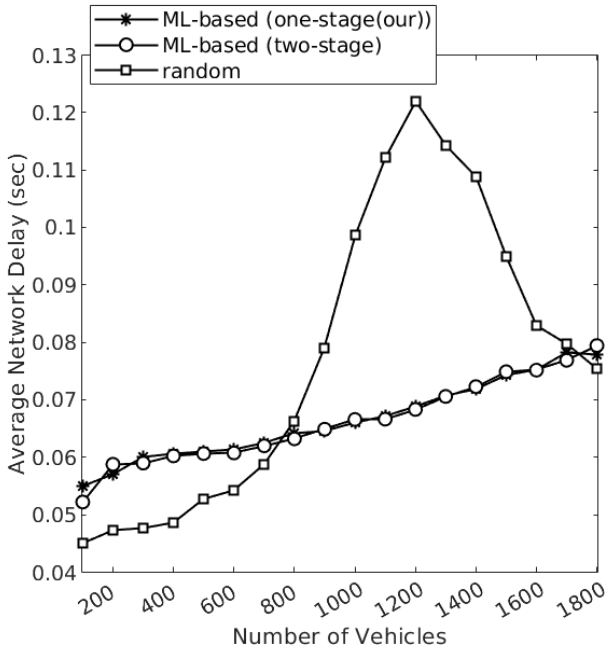

FIGURE 11. Network delay of approaches during simulation in EdgeCloudSim.

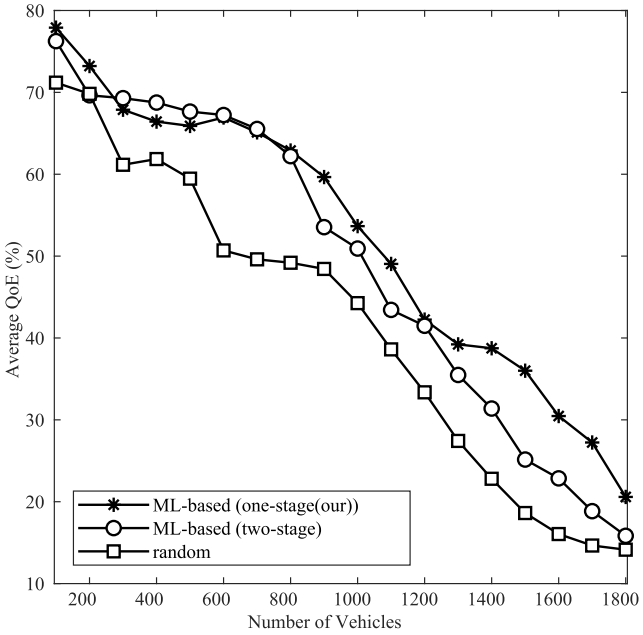

FIGURE 13. Average QoE for the number of vehicles.

received. Furthermore, by using the one-stage ML model that outputs the average of the prediction interval and the variance of this interval, we can measure how confident our model is in its prediction. Finally, the Equation formula 5 defines the Quality of Experience (QoE) that evaluates the service time provided by the orchestrator and the number of lost tasks.

$$QoE_i = \begin{cases} 0, & \text{if } T_i \geq 2R_i \\ (1 - \dfrac{T_i - R_i}{R_i}).(1 - S_i), & \text{if } R_i < T_i < 2R_i \\ 1, & \text{if } T_i \leq R_i \end{cases} \tag{5}$$

where T_i, as defined in [23], is the current service time, R_i is the delay requirement, and S_i is the delay sensitiveness

of a task τ_i. The delay requirements define the maximum available service time for the corresponding task. The average QoE value decreases as task τ_i is completed after R_i. Delay sensitivity guides delay tolerance. This value ranges from 0 to 1 and higher for applications with delay intolerance. Figure 13 shows the average QoE associated with the number of automobiles. We determine the delay requirements (by task sizes) and task delay sensitivities as in Table 2.

As shown in Figure 13, the average QoE of models that use machine learning presents the best results, as its objective is to minimize the service time for task processing. On the other hand, the random approach does not seek to reduce the service time and has the worst result.

VI. CONCLUSION

The purpose of this work was to develop an edge-centric workload orchestration proposal that takes advantage of the benefits offered by machine learning. In this way, we managed the limited on-board processing capacity and minimized the response time for connected autonomous vehicles. Thus, we could show that our proposal is more concise and consistent through the results of the evaluations and comparisons.

In future work, we intend to implement vehicle-to-vehicle (V2V) communication and fog layer in the existing three-tier architecture, further improving our ML-based orchestration proposal. In addition, we also plan to compare the performance of this centralized architecture with a distributed offloading architecture. Finally, we also intend to evaluate our proposal with other intelligent connected vehicles, such as object localization drones.

REFERENCES

[1] C. Koulamas and M. T. Lazarescu, "Real-time embedded systems: Present and future," *Electronics*, vol. 7, no. 9, p. 205, 2018.

[2] D. Aguiari, A. Ferlini, J. Cao, S. Guo, and G. Pau, "Poster abstract: C-continuum: Edge-to-cloud computing for distributed AI," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 1053–1054.

[3] E. Curry and A. Sheth, "Next-generation smart environments: From system of systems to data ecosystems," *IEEE Intell. Syst.*, vol. 33, no. 3, pp. 69–76, May 2018.

[4] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47980–48009, 2018.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, New York, NY, USA, Aug. 2012, pp. 13–16.

[6] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for Internet of Things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Cham, Switzerland: Springer, 2014, pp. 169–186.

[7] Y. Jararweh, A. Doulat, A. Darabseh, M. Alsmirat, M. Al-Ayyoub, and E. Benkhelifa, "SDMEC: Software defined system for mobile edge computing," in *Proc. IEEE Int. Conf. Cloud Eng. Workshop (ICEW)*, Apr. 2016, pp. 88–93.

[8] P. Liu, J. Li, and Z. Sun, "Matching-based task offloading for vehicular edge computing," *IEEE Access*, vol. 7, pp. 27628–27640, 2019.

[9] J. Crowcroft, *Metropolitan Area Network (MAN)*. Hoboken, NJ, USA: Wiley, 2003, pp. 1155–1157.

[10] R. A. Nafea and M. A. Almaiah, "Cyber security threats in cloud: Literature review," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, Jul. 2021, pp. 779–786.

[11] W. Ahmad, A. Rasool, A. R. Javed, T. Baker, and Z. Jalil, "Cyber security in IoT-based cloud computing: A comprehensive survey," *Electronics*, vol. 11, no. 1, p. 16, Dec. 2021.

[12] P. Dini and S. Saponara, "Analysis, design, and comparison of machine-learning techniques for networking intrusion detection," *Designs*, vol. 5, no. 1, p. 9, Feb. 2021.

[13] P. Dini, A. Begni, S. Ciavarella, E. De Paoli, G. Fiorelli, C. Silvestro, and S. Saponara, "Design and testing novel one-class classifier based on polynomial interpolation with application to networking security," *IEEE Access*, vol. 10, pp. 67910–67924, 2022.

[14] C. Anagnostopoulos, T. Aladwani, I. Alghamdi, and K. Kolomvatsos, "Data-driven analytics task management reasoning mechanism in edge computing," *Smart Cities*, vol. 5, no. 2, pp. 562–582, Apr. 2022.

[15] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 2, pp. 769–782, Jun. 2019.

[16] V. Nguyen, T. T. Khanh, T. Z. Oo, N. H. Tran, E.-N. Huh, and C. S. Hong, "Latency minimization in a fuzzy-based mobile edge orchestrator for IoT applications," *IEEE Commun. Lett.*, vol. 25, no. 1, pp. 84–88, Jan. 2021.

[17] M. D. Hossain, T. Sultana, M. A. Hossain, M. I. Hossain, L. N. T. Huynh, J. Park, and E.-N. Huh, "Fuzzy decision-based efficient task offloading management scheme in multi-tier MEC-enabled networks," *Sensors*, vol. 21, no. 4, p. 1484, Feb. 2021.

[18] D. Zhou, F. Chao, C.-M. Lin, L. Yang, M. Shi, and C. Zhou, "Integration of fuzzy CMAC and BELC networks for uncertain nonlinear system control," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2017, pp. 1–6.

[19] G. Gursel, "Healthcare, uncertainty, and fuzzy logic," *Digit. Med.*, vol. 2, no. 3, pp. 101–112, 2016.

[20] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, "Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework," *IEEE Commun. Mag.*, vol. 57, no. 3, pp. 56–62, Mar. 2019.

[21] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, "Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading," *IEEE Wireless Commun.*, vol. 27, no. 1, pp. 92–99, Feb. 2020.

[22] N. A. Abu-Taleb, F. H. Abdulrazzak, A. T. Zahary, and A. M. Al-Mqdashi, "Offloading decision making in mobile edge computing: A survey," in *Proc. 2nd Int. Conf. Emerg. Smart Technol. Appl. (eSmarTA)*, Oct. 2022, pp. 1–8.

[23] C. Sonmez, C. Tunca, A. Ozgovde, and C. Ersoy, "Machine learning-based workload orchestrator for vehicular edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2239–2251, Apr. 2021.

[24] O. Baslaim and A. Awang, "Intelligent offloading decision and resource allocation for mobile edge computing," in *Proc. Int. Conf. Future Trends Smart Communities (ICFTSC)*, Dec. 2022, pp. 204–209.

[25] S. M. D. A. C. Jayatilake and G. U. Ganegoda, "Involvement of machine learning tools in healthcare decision making," *J. Healthcare Eng.*, vol. 2021, pp. 1–20, Jan. 2021.

[26] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 38–67, 1st Quart., 2020.

[27] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, fog, or edge: Where to compute? *IEEE Internet Comput.*, vol. 25, no. 4, pp. 30–36, Jul./Aug. 2021.

[28] Y. Wang, Q. Cui, and K.-C. Chen, "Machine learning enables predictive resource recommendation for minimal latency mobile networking," in *Proc. IEEE 32nd Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2021, pp. 1363–1369.

[29] C. Li, C.-P. Li, K. Hosseini, S. B. Lee, J. Jiang, W. Chen, G. Horn, T. Ji, J. E. Smee, and J. Li, "5G-based systems design for tactile internet," *Proc. IEEE*, vol. 107, no. 2, pp. 307–324, Feb. 2018.

[30] R. Chen, C. Li, S. Yan, R. Malaney, and J. Yuan, "Physical layer security for ultra-reliable and low-latency communications," *IEEE Wireless Commun.*, vol. 26, no. 5, pp. 6–11, Oct. 2019.

[31] I. M. Sobol, "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates," *Math. Comput. Simul.*, vol. 55, nos. 1–3, pp. 271–280, 2001.

[32] S. Tennøe, G. Halnes, and G. T. Einevoll, "Uncertainpy: A Python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience," *Frontiers Neuroinform.*, vol. 12, p. 49, Aug. 2018.

[33] A. Saltelli, "Making best use of model evaluations to compute sensitivity indices," *Comput. Phys. Commun.*, vol. 145, no. 2, pp. 280–297, May 2002.
[34] A. Saltelli, *Global Sensitivity Analysis: The Primer*. Hoboken, NJ, USA: Wiley, 2008.
[35] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, "Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index," *Comput. Phys. Commun.*, vol. 181, no. 2, pp. 259–270, Feb. 2010.
[36] H. Yang and J. Moody, "Feature selection based on joint mutual information," in *Proc. Int. ICSC Symp. Adv. Intell. Data Anal.*, vol. 23, 1999, pp. 1–8.
[37] M. Z. I. Chowdhury and T. C. Turin, "Variable selection strategies and its importance in clinical prediction modelling," *Family Med. Community Health*, vol. 8, no. 1, 2020, Art. no. e000262.
[38] C. Sonmez, A. Ozgovde, and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of edge computing systems," in *Proc. 2nd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, May 2017, pp. 39–44.

MARIA J. P. PEIXOTO received the bachelor's degree in digital systems and media and the master's degree in computer science from the Federal University of Ceará, Brazil. She is currently pursuing the Ph.D. degree in electrical and computer engineering with Ontario Tech University. She has experience in machine learning, multimedia systems, ubiquitous computing, and robotics.

AKRAMUL AZIM (Senior Member, IEEE) is currently an Associate Professor with the Department of Electrical, Computer, and Software Engineering and the Head of the Real-Time Embedded Software (RTEMSOFT) Research Group, Ontario Tech University, Oshawa, ON, Canada. His research interests include real-time systems, embedded software, software verification and validation, safety-critical software, and intelligent transportation systems. He is a Professional Engineer in Ontario.