



Artificial Intelligence Workload Allocation Method for Vehicular Edge Computing

Sarah A. Rafea ^{1*}, Ammar D. Jasim ²

¹ Ph.D candidate, College of Information Engineering, Al-Nahrain University, Baghdad, Iraq

² Assistant Professor, College of Information Engineering, Al-Nahrain University, Baghdad, Iraq

* Corresponding Author: sara.ammar@coie-nahrain.edu.iq

Citation: Rafea, S. A., & Jasim, A. D. (2024). Artificial Intelligence Workload Allocation Method for Vehicular Edge Computing. *Journal of Information Systems Engineering and Management*, 9(3), 30380. <https://doi.org/10.55267/iadt.07.15495>

ARTICLE INFO

Received: 18 Aug 2024

Accepted: 30 Aug 2024

ABSTRACT

Real-time applications such as smart transportation systems require minimum response time to increase performance. Incorporating edge computing, processing units near end devices, achieving fast response time. The collaboration between edge servers and cloud servers is beneficial in achieving the lowest response time by using edge servers and high computational resources by using cloud servers. The workload allocation between edge–cloud servers is challenging, especially in a highly dynamic system with multiple factors varying over time. In this paper, the workload allocation decisions among the edge servers and cloud are considered for autonomous vehicle systems. The autonomous vehicle system generates multiple tasks belonging to different AI applications running on the vehicles. The proposed method considers allocating the tasks to edge or cloud servers. The cloud servers can be reached through a cellular network or a wireless network. The proposed method is based on designing a neural network model and using a high number of features that contribute to the decision-making process. A huge dataset has also been generated for the implementation. The EdgeCloudSim is used as a simulator for implementation. The competitor's methods considered for the comparison are random, simple moving average (SMA) based, multi-armed bandit (MAB) theory-based, game theory-based, and machine learning-based workload allocation methods. The result shows an improvement in the average Quality of Experience (QoE), ranging from 8.33% to 28.57%, while the average failure rate achieved enhancement up to 50%.

Keywords: Vehicular Edge Computing, Cloud Computing, Workload Allocation, AI Application, Neural Network.

INTRODUCTION

Given the rapidly expanding Internet of Things (IoT) users, Zhang, Cao and Dong (2020) argue that edge computing significantly pushes real-time computing near the end devices. The three-layer hierarchical architecture framework has recently been used for distributed workloads among cloud servers, edge servers, and end devices. This architecture achieves better user experience and performance acceleration by incorporating artificial intelligence (AI) technology at the network edge, as presented by X. Wang et al. (2018). The majority of edge AI applications, like self-driving cars, have strict end-to-end response time requirements and are sensitive to latency. In this context, end-to-end response time, as defined by L. Liu, Chen, Pei, Maharjan and Zhang (2021), refers to the whole length of time needed—including processing time across multiple layers—from the time an end device generates a task until receiving the response. Thus, it is imperative to determine ways to accelerate these edge AI applications' response times (Biswas & Wang, 2023).

In the context of an autonomous car, a system functioning in real time interacts with many other vehicles,

people, and traffic signals. This calls for accurate and dependable run-time response, even in the event of failure or unanticipated circumstances. Furthermore, Koulamas and Lazarescu (2018) emphasize that in certain real-time systems, accuracy and response time may be vital to safety. Therefore, the system must guarantee task safety overall. Task safety includes a guarantee of the appropriate execution time and its proper schedulability during decision-making, particularly in safety incidents. Addressing the uncertainty in such a highly dynamic system resulting from their operational environment is mandatory in order to make safe and accurate decisions about the workload these systems must handle. As a result, these systems frequently have to react to unexpected circumstances, which results in the generation of tasks that require immediate processing. Additionally, the increased workload that results from this entire process must be assigned to one or more servers. The usage of cloud infrastructure was one of the solutions, taking into account some of the system constraints, such as mobility, bandwidth, and latency. The cloud enables on-demand services and resource scalability (Naha et al., 2018). The need for low-latency, high-mobility data processing and resource sharing led to the development of the Multi-Access Edge Computing (MEC) concept. To further reduce request latency, MEC's primary goal is to deploy computing resources closer to end users. Processing and storage are thus carried out closer to the origin location.

The execution of workloads is impacted by the restricted processing and storage capacity of edge devices. The amount of data that has to be processed also has a great impact. The choice of whether to run an Internet of Things (IoT) application's workload in the cloud or on edge servers influences how quickly requests are handled for each application. One of the most important factors affecting user QoE is the allocation of the workload, as presented by Hao, Zhan, Hwang, Gao and Wen (2021). To improve the program's performance, the AI application expanded and interacted with edge computing systems in various ways. For example, it suggested workload distribution strategies and lightweight AI models.

Several methods are proposed to enhance the performance of AI Applications by proposing workload allocation decision methods. The methods presented are mainly based on a mathematical approach or AI approach. Mathematical approaches such as heuristic algorithms and fuzzy logic are mathematically complex and computationally intensive in high-dimensional problems (Ammar & Dawood, 2024). The workload allocation decision problem in highly dynamic environments is considered an np-hard problem, as presented by H. Wang, Peng and Pei (2020). They also mentioned that the AI approach is more suitable for reducing complexity and adjusting to varying environments that lack certainty.

The rest of this paper presents the literature review section followed by the proposed method for workload allocation. The performance evaluation section is also presented. Finally, the conclusion section is presented as the final section of this paper.

LITERATURE REVIEW

Long, Luo, Zhu, Luo and Huang (2020) suggested the Computation Offloading scheme Through mobile Vehicles (COTV) system, which enabled the sensor devices to transmit their tasks to the cars. Next, the autonomous car determined whether the task was carried out locally, via the edge server, or in the cloud. The COTV scheme's system architecture, which was made up of the cloud centre layer, MEC server layer, mobile vehicle layer, and sensing device layer, was examined. Sensor-generated tasks were processed at cloud centres, MEC servers, or mobile vehicles. The authors used reinforcement learning to reduce energy usage and delay time for IoT devices with limited resources. The outcome showed that the suggested strategy outperformed the baseline strategy in terms of delay time. The main drawback and limitation is that the authors mainly take into account the tasks created by the sensor devices; if an IoT sensor's speed is less than the vehicle's speed, this could result in a task failure. Also before the task is completed, the car can drive out of the edge server's and the sensor's coverage area. Nevertheless, the work did not take the devices' speed into account.

H. Wang et al. (2020) provided MEC with a method that made use of underutilized resources at edge servers to facilitate cooperation between edge servers. When the primary edge server was unable to meet the task's delay requirement, the device's created task could be sent to the assisted edge server. The authors demonstrated the complexity of the workload allocation problem and classified it as an np-hard problem. A heuristic algorithm was applied based on the priorities of mobile devices and the MEC servers. The suggested method was contrasted with one that was entirely executed at the edge or locally executed on a mobile device. The findings demonstrated that the suggested method enhanced system reliability and decreased latency. Since the approach relies on a heuristic algorithm, the rise in mobile devices and/or edge servers may complicate the model and reduce performance. The authors considered a limited amount of mobile and edge devices.

Sonmez, Tunca, Ozgovde and Ersoy (2021) suggested a two-phase machine learning strategy for vehicular

edge computing (VEC). Classifying the task as successful or unsuccessful on the edge, in the cloud through RSU, or in the cloud through CN was the initial step. If a task was deemed effectively completed on a particular layer, the service time during execution was estimated using the regression model. Ultimately, the task was completed on the layer with the shortest service time. Three regression models and three classification models were used. The suggested model was tested on EdgeCloudSim. The suggested model achieved better results than its competitors in terms of average job failure rates and service time, hence improving the overall QoE. The main drawback of this model is that it requires multiple AI models - six different AI models. Each model has different features, causing effort to train each of these models. Moreover, this will increase system complexity and overhead.

Dos Anjos et al. (2021) suggested the TEMS algorithm and dynamic cost model. The suggested work aimed to select a layer for job execution to reduce execution time and energy usage. The authors took into account a number of parameters, including energy, processing time, data transmission costs, and idle device energy consumption, in order to optimize the energy consumption and task processing time for IoT devices in MEC environments. The cloud, edge server, and end device layer were the layers taken into consideration. The experiment demonstrated how the suggested method lowered wait times and energy consumption. The authors only took into account a limited selection of mobile and edge devices. As noted by the authors, the algorithm's $O(n^2)$ complexity could raise the model's complexity for large input sizes, causing performance issues.

Nguyen et al. (2021) suggested a method for allocating workloads between edge servers and the cloud that used fuzzy reasoning. Numerous factors were taken into account, including WAN bandwidth, task duration, and edge utilization delay. The output presented the execution location. In order to reduce latency, the authors also suggested a method for dividing up the tasks to be completed between edge and cloud computing. The EdgeCloudSim simulator was used to implement the suggested approach. The outcome indicated that the suggested strategy outperformed competitors in terms of failure rate and service time. The author noted that minimizing delay times in highly dynamic circumstances was a challenge that required attention. The authors proposed adding additional features in future work to shorten task completion times. It can get challenging to grasp the decision-making process when more features are added to fuzzy logic models as the complexity of the fuzzy rules rises as well.

Dai, Liu, Mo, Xu and Huang (2022) used a reinforcement learning (TODQN) strategy for an application for VEC. The suggested approach considered carrying out the workload on cars, the edge server, or the cloud. The suggested approach balanced the exploitation and exploration processes to find the best workload distribution schemes using deep Q-learning in a highly dynamic environment. During the issue formulation phase, the authors employed tasks, cars, wireless channels, MEC servers, and the available computing and communication resources as measurements. When it came to delay time, the suggested strategy outperformed the baseline technique. The primary drawback is that the authors employ a limited number of tasks and treat one edge server as a DQN agent that makes decisions. The authors also discussed the model's $Q(n_3)$ complexity in the absence of the deep neural network's complexity. Because the $O(n^3)$ complexity of the technique may not scale well in complex circumstances, the runtime increases noticeably as the input size increases.

Yang, Lee and Huang, (2022) suggested a deep learning-based strategy for resource allocation and task allocation (DSLO). In order to address the allocation choice problem, the authors took into account both the convolution neural network and the deep neural network also utilized convex optimization to allocate resources as well. The binary decision was either carrying out the task locally or forwarding it to the cloud. The main limitation is that the authors consider one edge server and a small number of devices reached 15 devices. Also, the simplicity of the dataset could be the reason for the fast convergence of the model since it contains a small number of features.

Z. Liu, Jia, and Pang (2023) considered how to allocate resources and workload in light of multi-user VEC. They proposed a hybrid technique (HTCO) for decision-making that used DRL to choose the execution locally. The primary constraint is that the method solely examines the offloading approach for multiple vehicle users under a single-edge server, which is incapable of extending task offloading outside the server's coverage. The network characteristics are not considered, such as channel congestion not reflecting the real environment.

Ullah, Lim, Seok and Han (2023) chose whether to carry out the tasks on the edge server or in the cloud by using the Double Deep Q-Network (DDQN) algorithm for edge-cloud (DDQNEC). For additional advancement, the author suggested investigating more sophisticated machine learning and AI techniques. In order to maximize job offloading, taking additional elements into account might have also improved the algorithm's performance and end devices' capabilities. Because the approach might not scale effectively in complex circumstances, the runtime grows dramatically as the input size increases.

Peixoto and Azim (2023) provided a system for task orchestration in very high mobility scenarios. The system was designed for VEC and was based on machine learning. The technique used three layers: edge, cloud RSU, and

cloud through the cellular base station. A sensitivity analysis was used to determine which features were more impacted by the model's choice. The main drawback of the method is that using the host ID as one of the five features in the Machine learning models to predict service time in each layer is inaccurate because the host ID is known only after the execution location has been identified and cannot feed to the model as input. Also, the authors train the model on a large dataset using the random forest algorithm, which generates a large-size model for inference.

Chen, Fan, Yuan and Hao (2024) suggested using reinforcement learning to relocate some of the work to the edge server or the collaborative vehicle. The suggested methodology reduced the failure rate and overall delay compared to the baseline approaches. Due to the very small number of cars taken into consideration, the task failure rate and network conjunction effect were inaccurate. Prior to deciding where each portion was conducted, the task had to be given to the edge. The edge decided which parts were executed at the edge, in the vehicle, and in the collaborative vehicle. Due to the vehicle's high mobility, the tasks ultimately failed after the results were gathered by the edge and given to the vehicle that created it in the first place. This reduces the possibility of offloading at high speeds.

Wu, Jia, Pang and Zhao (2024) developed an RL-based workload and resource allocation technique named (TOLB). The suggested strategy was to choose places for task execution between cars or edges. Additionally, it offered load balancing and work splitting through task migration between edge servers. The task-splitting strategy works well because the implementation takes into account a small number of cars and a comparatively large number of edge servers in comparison to the number of vehicles that may give very high coverage. The car could discontinue gaining coverage from the edge server before getting the remaining portion of the task if there was less coverage. A large number of vehicles may also cause the model's complexity to rise.

Sun, Chen and Li, (2024) proposed a system for allocating workloads amongst cars, edge servers, and the cloud known as L-MADRL, which mixed deep learning and reinforcement learning. Reinforcement learning was used to choose edge servers, while deep learning was utilized to predict the location of vehicles. The agent's excessive preference for long-term returns, which is brought on by larger discount factors, could lead the learning process to progress and compromise the stability of the learning outcomes sluggishly.

In this work, a neural network model is designed to allocate the workload generated by the vehicles. The three-layer architecture is considered in this work. The options for allocation are either the edge or the cloud with two paths. The allocation to the cloud is through a cellular network (CN) or through a wireless network. Incorporating the CN offered high coverage that is needed in high-speed vehicles.

The main contributions of the work are:

Propose a workload allocation method based on AI technology in the form of a neural network to achieve generalization, minimize complexity, and balance resource load to meet AI application requirements and provide a high Quality of Experience (QoE).

A dataset created to train the model, the dataset contains more than a million records with 21 features.

18 features are considered that affect the allocation performance. Four features contribute to the dataset creation, while 14 features are used as input to the neural network model.

A comparison with the other workload allocation methods has been presented by conducting extensive simulation tests. The competitor's methods are random, simple moving average (SMA) - based, multi-armed bandit (MAB) theory-based, game theory-based, and machine learning-based workload allocation methods.

THE PROPOSED SYSTEM MODEL

Vehicular System Architecture

In the Internet of Vehicles (IoV), the three-layer architecture is considered as presented in **Figure 1**. The first layer represents the vehicles generating tasks that need to be executed. The second layer represents the RSU. Each RSU is equipped with an edge server. The third layer is the cloud centre. The vehicles have three options to transmit their task to the edge server to the cloud through RSU or to the cloud through a cellular network.

The vehicle transmits its task to the edge server through a wireless connection (WLAN). The WLAN represents the vehicle-to-road connection (V2R). And use cellular networks as vehicle-to-infrastructure (V2I) communication. Vehicle tasks can also be sent to the cloud through an internet connection (WAN). A metropolitan area network (MAN) is also connected to the RSUs. Task migration is one way that MAN connection

enables RSUs to share their computational resources.

If the connected vehicle leaves the serving RSU's range before the offloaded task is completed, the result is sent to it in a multi-hop manner through the other RSUs. The vehicle must be out of the serving RSU's range for the handover process. The tasks are considered failures if the vehicle moves out of the coverage area during data uploading or downloading.

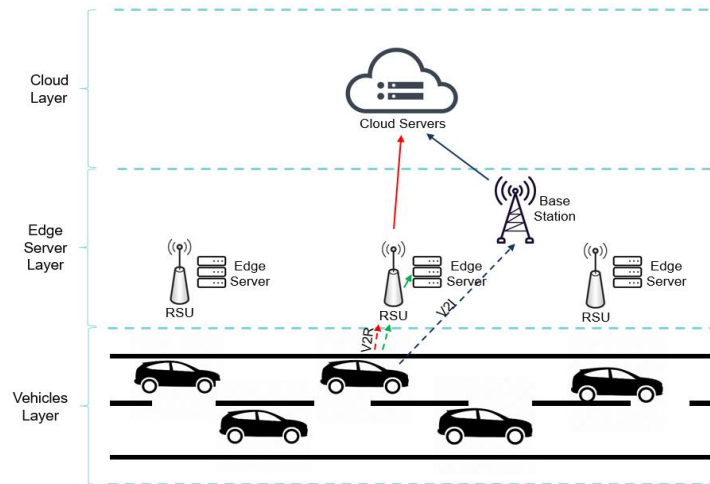


Figure 1. Vehicular Edge Computing Architecture

Problem Formulation

As previously mentioned, IoT devices work in resource-constrained environments in many aspects, such as communication resources, computation resources, and storage. These systems can leverage remote servers to handle their data processing. In this regard, a workload management system is considered to make allocation decisions between edge servers and the cloud since the onboard processing is neglected and not affected by the proposed system.

The proposed model manages task offloading at runtime. It uses different types of features to predict the location of execution. Task processing options include cloud-based cellular base stations (CBS), cloud-based roadside units (RSU), and edge servers.

Each vehicle generates task τ , which needs to be processed on server S .

$$S = \{sE1, sRSU1, sCBS1, \dots, sEF, sRSUF, sCBSF\}.$$

Notation sE represents the edge node, $sRSU$ represents the cloud node through RSU, and $sCBS$ represents the cloud node through CBS. Hence, the objective is to specify a neural network-based model that assigns task τ to n , which belongs to S , in order to minimize the service time R of autonomous vehicle V that made the task τ processing request. This is done by training the model based on minimum service time for executed tasks.

In light of this, this work suggests a simple, one-stage, one-model neural network task orchestrator to address the issue of limited onboard processing capability and reduce response times for intelligent vehicles. The proposed model relies on NN support to predict the location of execution for the autonomous agents' workloads to be completed at the edge in the cloud through RSU or in the cloud through CBS. The model selects the target device based on this prediction. Furthermore, the proposed model relies on stand-alone features and uses dependable features to customize the dataset. In this context, the stand-alone features refer to the features that can be known before executing the tasks, such as vehicle location. The dependable feature refers to the features that depend on the task execution to be known, such as service time.

AI Proposed Model

As presented earlier, the tasks generated by vehicles are sent to edge servers or the cloud through two options. The selection of task execution location is considered in this work. The proposed model optimizes the workload allocation process by considering multiple features related to the resources, communication, and application characteristics.

The model presented in **Figure 2**, consists of the input layer, hidden layer, and output layer. The input layer

consists of 14 neurons while one hidden layer is composed of 7 neurons. The output layer contains 3 neurons with a softmax activation function. The model output is represented as a vector of 3 values. Each value of the vector represents one execution location. The model uses a fully connected layer. The model parameter is presented in **Table 1**.

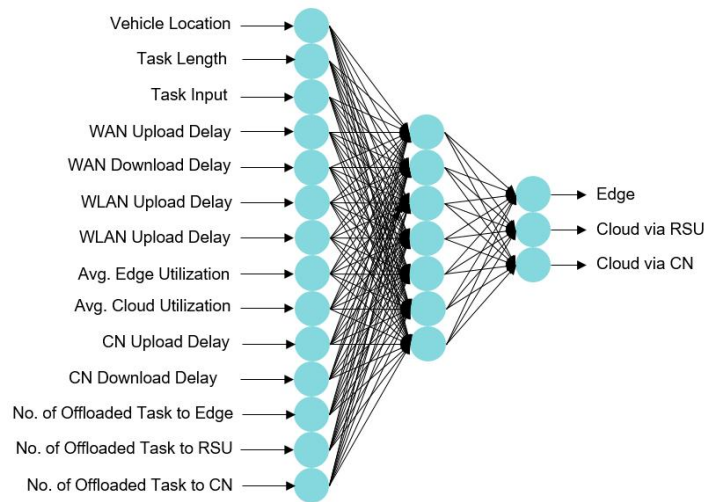


Figure 2. The Proposed Model Architecture

Table 1. Proposed Model Specification

Parameter	Specification or Value
Neural Network Layers	3 (1-Input, 1-Hidden, 1-Output)
Activation Function	Layer 1,2 -Relu, 3-Softmax
Optimizer	Adam
Dataset Size	1,306,812 Records (80% Training, 10% Validation, and 10% Testing)
Epoch	20
Patch Size	32
Learning Rate	0.001

The features selected as input parameters presented in **Figure 2** are all considered as standalone features. The standalone features are vehicle length, task length, task input, WAN upload delay, WAN download delay, WLAN upload delay, WLAN download delay, average edge utilization, average cloud utilization, cellular network (CN) upload delay, CN download delay, number of offloaded tasks to the edge, number of offloaded tasks to the cloud through RSU, and number of offloaded tasks through CN.

Dataset Creation

The model proposed has been trained using a pre-generated dataset. The dataset has been created by collecting the data from the EdgeCloudSim simulator. **Figure 3** presents the main steps required for dataset creation. Preprocessing the dataset is carried out by selecting the successfully executed task on a certain layer. In case the task is executed successfully on more than one option, the option with minimum service time is selected. Hence, each task should be recorded with minimum service time and successfully executed.

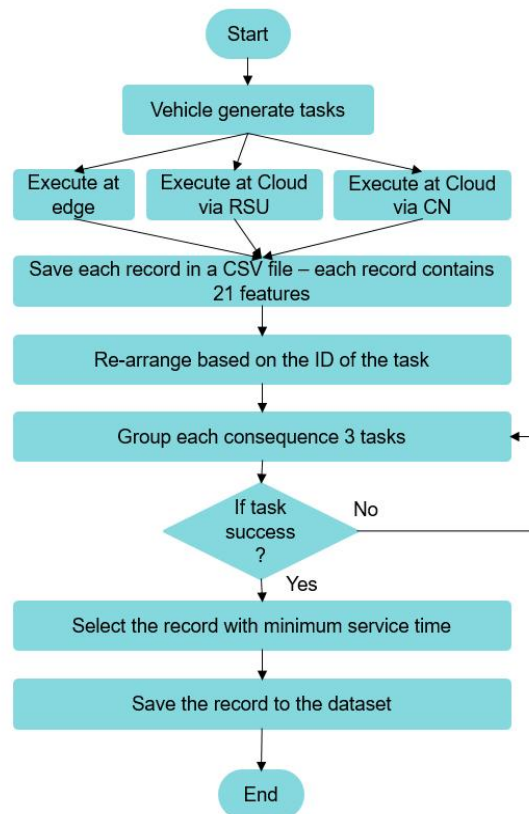


Figure 3. Flowchart for Generating the Dataset

Twenty-one features are recorded for each task. As presented in **Table 2**, the features recorded are Decision, Result, ID, ServiceTime, ProcessingTime, VehicleLocation, SelectedHostID, TaskLength, TaskInput, TaskOutput, WANUploadDelay, WANDownloadDelay, CNUplodDelay, CNDownloadDelay, WLANUploadDelay, WLANDownloadDelay, AvgEdgeUtilization, AvgCloudUtilization, NumOffloadedTaskE, NumOffloadedTaskRSU, NumOffloadedTaskCN.

Table 2. The Dataset Features with its Specification

Features	Specification
Decision	Location of Execution (Edge, Cloud through RSU, and Cloud through CN)
Result	Results of Execution (Success or Fail)
ID	Number of Tasks
ServiceTime	Time Required to Execute the Task and Get the Result Back to the Vehicle
ProcessingTime	Time for Executing the Task on Edge or Cloud through RSU or Cloud through CN
VehicleLocation	Location of the Vehicle when Generating the Task
SelectedHostID	The ID of the Server Selected for Execution
TaskLength	Length of Task
TaskInput	Input Size of Data
TaskOutput	Output Size of Data
WANUploadDelay	Delay Time of Upload Link for WAN Network
WANDownloadDelay	Delay Time of Download Link for WAN Network
CNUplodDelay	Delay Time of Upload Link for CN Network
CNDownloadDelay	Delay Time of Download Link for CN Network
WLANUploadDelay	Delay Time of Upload Link for WLAN Network
WLANDownloadDelay	Delay Time of Download Link for WLAN Network
AvgEdgeUtilization	Average Edge Utilization
AvgCloudUtilization	Average cloud Utilization
NumOffloadedTaskE	Number of Tasks Sent to the Edge for the Last 5 Seconds
NumOffloadedTaskRSU	Number of Tasks Sent to the Cloud through RSU for the Last 5 Seconds
NumOffloadedTaskCN	Number of Tasks Sent to the Cloud through CN for the Last 5 Seconds

The dependable features such as service time, results, and decision considered important features contribute to the prediction accuracy. However, these features can not be fed to the model as input because they can not be known before task execution. Hence the dependable features are used in the dataset generation process.

The usage of dependable features makes sure that the model selects the location of execution that provides less service time, ensures stability and generalization of the mode, and decreases the failure rate of tasks, hence maintaining the QoE required by the application.

Vehicles Application

The vehicle transmits tasks belonging to different applications with different requirements for each application. The applications considered are navigation, danger assessment, and infotainment.

All those applications belong to autonomous vehicle systems. Each application sends its task at different interarrival times, with different time sensitivity requirements and different task sizes. **Table 3** presents the application characteristics.

Table 3. Vehicle's Application Characteristics

Parameters	Danger Assessment	Infotainment	Navigation
Generating Task Percentage	35	35	30
Task Interarrival Time (sec)	5	15	3
Maximum Delay Requirement (sec)	1	1.5	0.5
Delay Sensitivity Requirement (sec)	0.8	0.25	0.5
Upload Data/ Download Data (KB)	40/20	20/80	20/20
Task Length (GI)	10	20	3
Edge/Cloud Utilization	20/4	40/8	6/1.6

PERFORMANCE EVALUATION

The effectiveness of the suggested workload allocation method is assessed by an extensive set of tests. The simulations are run on EdgeCloudSim (Sonmez, Ozgovde, & Ersoy, 2018), a program that can mimic mobile vehicles as well as computational and networking resources. Five well-known methods are used for the purpose of comparison. The game theory-based, SMA-based, MAB-based, ML-based, and random methods are considered. The specification and algorithm for all these approaches are presented in (Sonmez et al., 2021). The vehicles moved at different speeds. The density of the vehicles is presented in **Figure 4**. The x-axis represents the ID location of the vehicles. The location of the vehicle is identified by the location of RSU that the vehicle is in their coverage. The y-axis represents the number of vehicles on the road. The colored bar indicates the number of vehicles in each color. Each location covers around 10 to 20 vehicles in low number of vehicles. While in a very high number of vehicles, some RSU coverage around 90 vehicles and other location coverage around 30 vehicles. This diversity, varying speed segments, is achieved to emulate the actual road.

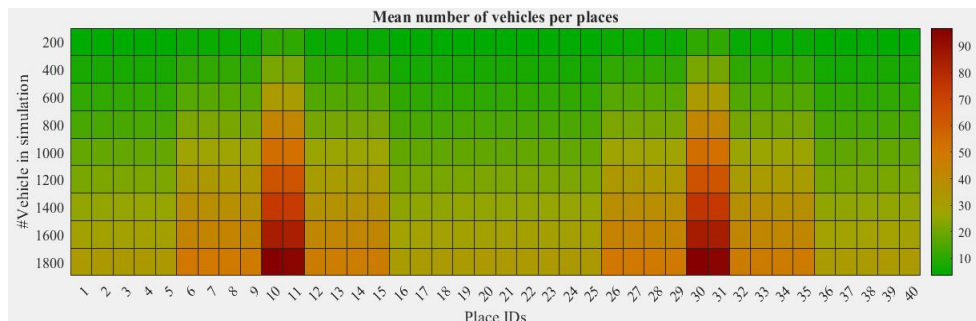


Figure 4. Vehicles Density

Performance Evaluation of the Proposed Model

Figure 5 presents the model's accuracy. The model converges fast and is stable at around 94% accuracy after 20 epochs. The validation accuracy is nearly the same as the training accuracy curve, indicating that the model

converges well and there is no overfitting.

The loss of the model is presented in **Figure 6**. The model converged fast until it reached 0.16% loss after 15 epochs. The loss of validation is nearly the same as the training loss.

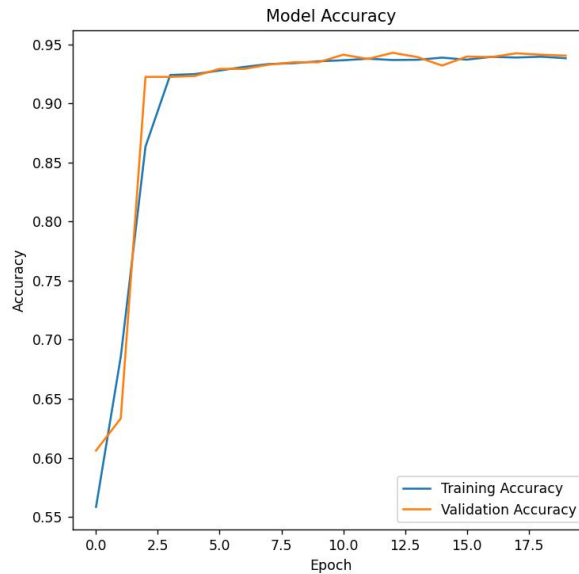


Figure 5. Average Model Accuracy

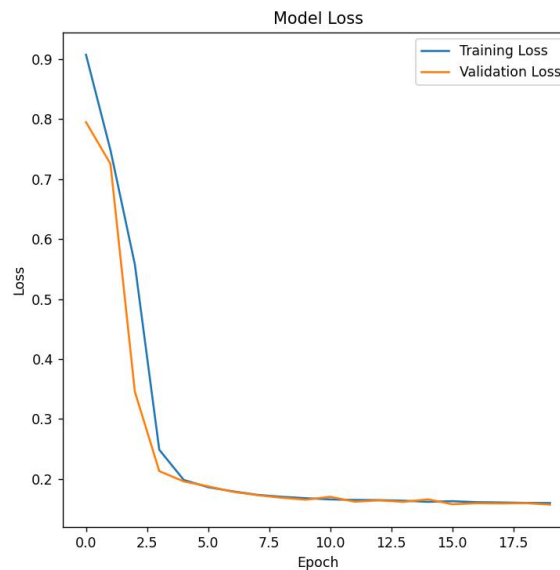


Figure 6. Average Model Loss

The confusion matrix of the model presented in **Figure 7** predicts a categorical label for each input. The confusion matrix presents the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) produced by the model on the test data. In multi-class classification, the rows of the confusion matrix demonstrate the data points' accurate classifications while the columns indicate the class's model predicted. The percentage of tasks that were accurately identified as being a part of the first class is 9.2%. The percentage of tasks that belong to the second class but were mistakenly assigned to the first class is 1%. The percentage of that should have been in the third class but was mistakenly assigned to the first class is 0%. 2.24% of tasks in the test dataset should have been in the first class but were mistakenly assigned to the second class. The percentage of data points that were accurately categorized as being a part of the second class is 41.23%, while 1% of tasks were mistakenly assigned to the second class when they should belong to the third class. 0.09% represents the percentage of tasks that belong to the first class but were mistakenly assigned to the third class. 1.4%, although adequately assigned to the second class, were mistakenly classified as belonging to the third class. Finally, the percentage of tasks accurately identified as part of the third class is 43.7%. Generally, high diagonal

values (12032, 53887, and 57055) reflect the model's ability to predict the location of execution correctly. Hence, the confusion matrix indicates that the classification model is predicting the correct classes with a high degree of accuracy. The evaluation metric for the proposed model includes average accuracy, average loss, precision, recall, F-score, Macro avg., and weighted average. They are presented in **Table 4**. The average accuracy is 94.2%, the average loss is 0.16. The model size after training and saving is 17.5 KB, while the training time is 7.08 minutes.

12032	1392	0
9.2%	1%	0%
2935	53887	1430
2.24%	41.23%	1%
13	1938	57055
0.09%	1.48	43.7

Figure 7. Confusion Matrix of the Proposed Model

Table 4. Evaluation Metric for Proposed Model

	Precision	Recall	F1-Score	Support
Class-0	0.80	0.90	0.85	13424
Class-1	0.94	0.93	0.93	58252
Class-2	0.98	0.97	0.97	59006
Macro avg.	0.91	0.93	0.92	130682
Weighted avg.	0.94	0.94	0.94	130682
Accuracy			94.2%	
Loss			0.16%	
Model Size			17.5 KB	
Training Time			7.08 min	

The proposed neural network model is compared to other well-known algorithms. The algorithms considered are the Random Forest, Naive Bayes, and Support Vector Machine (SVM). The Random Forest is an ensemble learning that enhances accuracy and decreases overfitting by combining several decision trees with tagging and random feature selection. Naive Bayes is a Probabilistic approach that provides effective classification based on probabilities. SVM is a supervised Learning method that uses support vectors to find the best hyperplane that maximizes the margin between classes. The evaluation metrics for Random Forest, Naive Bayes, and SVM are presented in **Table 5**, **Table 6**, and **Table 7**, respectively.

Table 5. Evaluation Metric for Random Forest

Metric	Values
Correctly Classified Instances (Accuracy)	94%
Incorrectly Classified Instances	5.3122%
Mean Absolute Error	0.0544
Root Mean Squared Error	0.1622
Model Size	542.6 MB
Training Time	25 min and 14 sec

Table 6. Evaluation Metric for NaiveBayes

Metric	Values
Correctly Classified Instances (Accuracy)	90.0686%
Incorrectly Classified Instances	9.9314%
Mean Absolute Error	0.0665
Root Mean Squared Error	0.2538
Model Size	5 KB
Training Time	13 sec

Table 7. Evaluation Metric for SVM

Metric	Values
Correctly Classified Instances (Accuracy)	93.6716%
Incorrectly Classified Instances	6.3284%
Mean Absolute Error	0.2364
Root Mean Squared Error	0.2971
Model Size	8 KB
Training Time	1 day, 7 hours, 24 min, 58 sec

The highest accuracy achieved with the neural network model reached 94.2%, followed by Random Forest, SVM, and NaiveBayes, achieving 94%, 93.6716 %, and 90.06%, respectively—the worst case in accuracy presented by NaiveBayes. The lowest training time achieved with NaiveBayes reached 13 seconds, while the others took 14 sec, 7.08 min, and 31.4 hours for Random Forest, neural network model, and SVM. The lowest model size achieved with NaiveBayes is 5KB, while the SVM, neural network model, and Random Forest model sizes are 8KB, 17.5KB, and 542.6MB, respectively. Hence, the proposed neural network model achieved the highest performance in terms of accuracy while achieving the lowest loss compared to the Root Mean Squared Error of another method. The Random Forest algorithm achieved performance close to the proposed model in terms of accuracy and loss, but the model size of Random Forest is significantly high compared to the proposed model. The increase in model size is due to the fact that the Random Forest algorithm is a tree-based algorithm that produces large complexity with a large dataset. The neural network model only saved the weight and structure, so the model size did not increase as the dataset increased.

Performance Evaluation of the Model Inference

The proposed neural network model is trained using Python programming language, TensorFlow library, and Keras. The proposed model is used for inference in the EdgeCloudSim simulator. The proposed model is compared to several approaches, which are the random, simple moving average (SMA) based, the multi-armed bandit (MAB) theory-based, game theory-based, and machine learning-based vehicular edge allocation decision. The random approach selects the execution location with equal probability for each execution layer. The SMA is a time series forecasting technique that predicts future results using recent, short-term historical data. The time series data is recorded by this technique using a fixed-length list, where each element of the list includes the data for a particular time period. By giving the data, which has been gathered more recently, a higher weight, the average success rate of each alternative is determined. Next, it chooses the target machine with the best success rate. MAB is a reinforcement learning approach that aims to maximize the expected gain. The game theory-based technique modifies the offloading probability of vehicles to maximize utility by using a multi-user, noncooperative computation offloading game. The machine learning (ML) based strategy proposed a two-stage machine learning model. The first stage is predicting whether the tasks will be successfully executed or fail. Then, if the tasks are executed successfully, then the service time is estimated for each location of execution to select the location with minimum service time. The features used for each model are also different.

The performance evaluation measures the average QoE, the average QoE for each application, the average failed task, and the reasons for failed tasks. The simulation settings are presented in **Table 8**.

The QoE is measured by considering the service time and failure tasks as in Eq. (1).

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

Where T_i represents the service time, R_i represents the application's delay requirement, and S_i represents delay sensitivity.

Table 8. Simulation Parameters

Parameter	Values
Simulation Time / Warm-up Period	60 / 3 Minutes
Network Delay Model	MMPP/M/1 Queue Model
No. of VMs per Cloud/Edge	20 / 40 (2 Per Edge)
Capacity of Cloud/Edge VM	150 / 20 GIPS
Range of RSU (WLAN)	200 m
MAN / WLAN Bandwidth	1000 / 10 Mbps
Bandwidth of WAN/WAN over LTE	50 / 20 Mbps

Parameter	Values
Propagation Delay of WAN/WAN over LTE	150 / 160 ms

Figure 8 presents the average QoE. The proposed method achieved the highest QoE compared to the other approaches considered with a high number of vehicles and a small number of vehicles, while the MAB-based approach achieved better performance in a medium number of vehicles. The MAB-based approach achieved better performance for a specific number of vehicles. However, its performance dropped quickly in a high number of vehicles since the complexity of the algorithm increased.

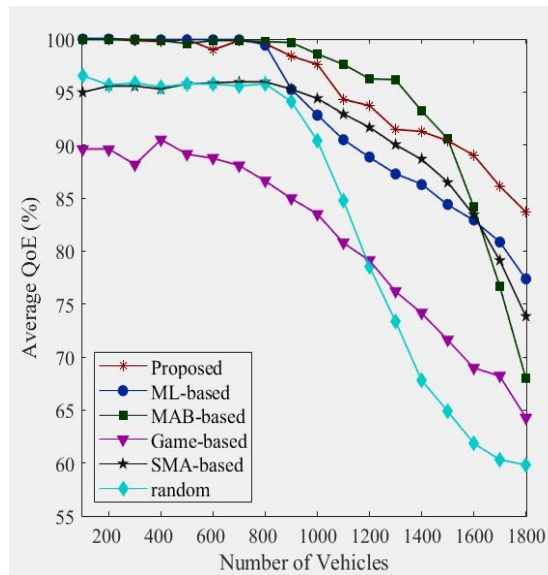


Figure 8. Average QoE

The proposed approach increased the average QoE by 28.57%, 11.9%, 23.8%, 19.04%, and 8.33%, compared to random, SMA-based, Game-based, MAB-based, and ML-based approaches, respectively.

Figure 9 represents the average QoE for the danger assessment application. The proposed methods achieved the highest performance in a relatively high number of vehicles compared to its competitors. The characteristics of this application reflect the model behaviour for medium inter-arrival time and the highest delay sensitivity application. The lowest QoE reached 80% and presented as the highest value achieved with 1800 vehicles.

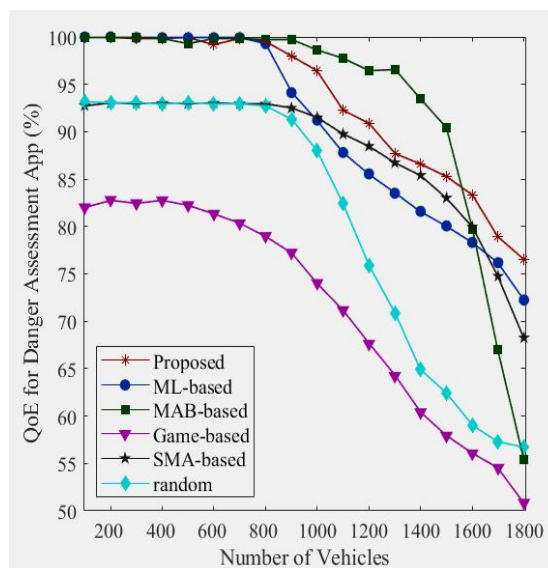


Figure 9. The QoE of Danger Assessment Application

Figure 10 presents the QoE for the Navigation application. The navigation application generates a task every 3 seconds, hence the lowest inter-arrival time, while the maximum delay requirement is the smallest. The proposed methods achieved the highest QoE for all vehicle numbers, reaching 94% for a large number of vehicles.

Figure 11 presents the QoE for the infotainment application. The app's best performance was achieved using an MAB-based algorithm for all vehicle numbers except the 1800 vehicles. The app's requirement allows the highest delay while generating the tasks every 15 seconds. The app generates the tasks with the highest interarrival time compared to the other application's interarrival time.

The average failed tasks are presented in **Figure 12**. The lowest failure tasks were achieved with the proposed methods and game-based approach. While the game-based approach achieved the lowest average failure rate, it exhibited the worst case in QoE for all applications at longest with the random approach. According to this, the comparison will be against the ML-based approach for average task failure.

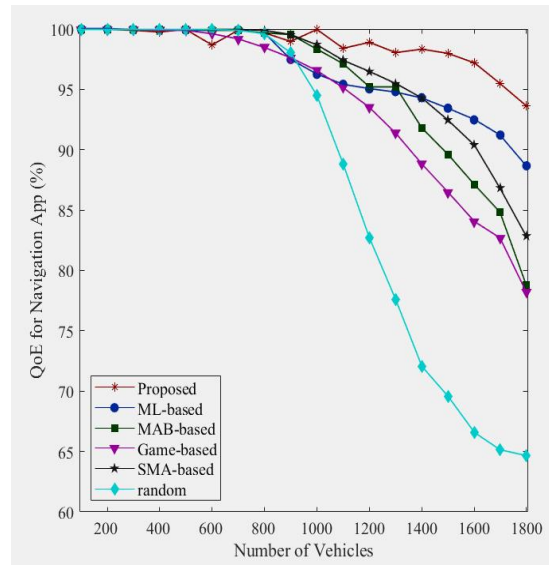


Figure 10. The QoE of the Navigation Application

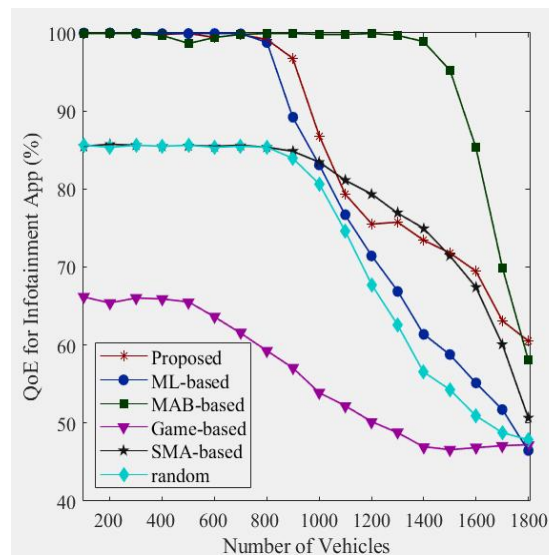


Figure 11. QoE for Infotainment Application

Only the ML-based approach with the proposed approach balances the QoE and failure rate. The proposed methods achieved nearly stable failure tasks among the other approaches, reaching 50% enhancement compared to the ML-based approach.

Failure tasks for infotainment, navigation, and danger assessment are presented in **Figures 13, 14, and 15**, respectively. The lowest failed task was achieved with a navigation application, reaching 7%. The lowest failure

rate has been achieved by the proposed method. The danger assessment has achieved the second lowest failure tasks, reaching 9% as a maximum value. In the small number of vehicles, lower than 700 vehicles, achieved no failure rate for all approaches considered. The highest failure rate has been achieved with infotainment applications.

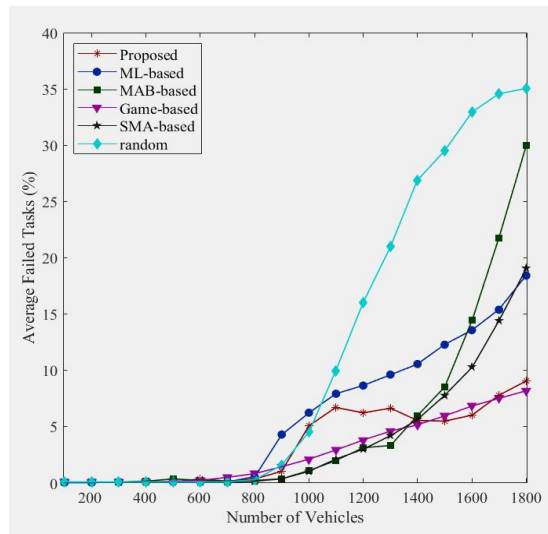


Figure 12. Average Failed Tasks

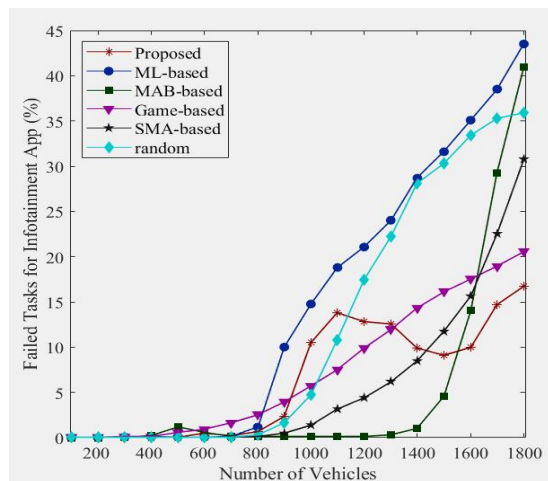


Figure 13. Failed Tasks for Infotainment Application

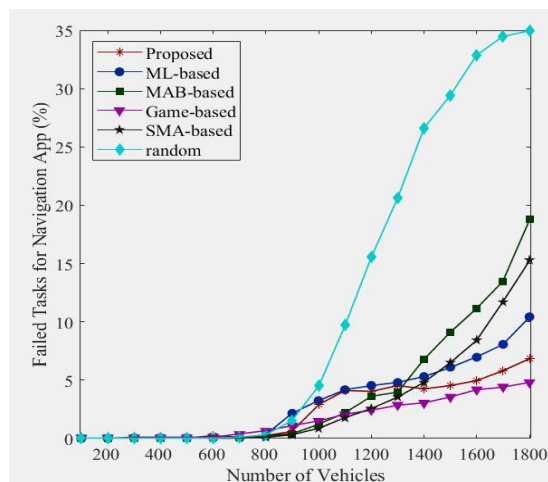


Figure 14. Failed Tasks of the Navigation Application

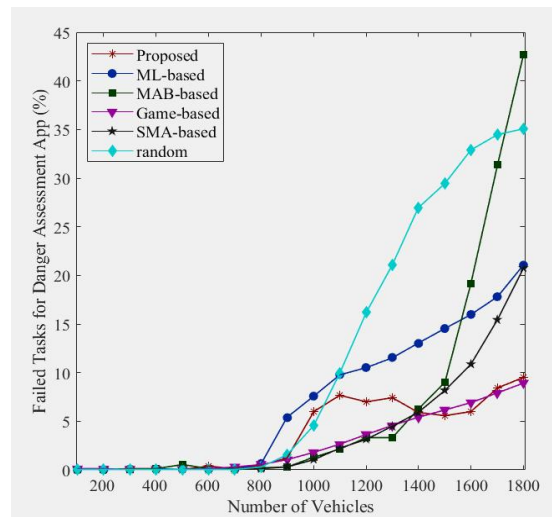


Figure 15. Failed Tasks for Danger Assessment Application

The failed task percentage caused by the VM Capacity is presented in **Figure 16**. The tasks that are assigned to the edge, cloud through RSU, or cloud through cellular network could fail in case there are no resources to process the tasks in that server.

As the number of vehicles increases, the Game-based method performs noticeably worse than the other algorithms, with a high increase in the percentage of unsuccessful tasks. The proposed methods achieved the lowest failed tasks for small and medium numbers of vehicles and slightly increased for a high number of vehicles indicating its superior performance. While the ML-based, MAB-based, SMA-based, and random algorithms exhibit comparable performance patterns, the proposed approach has a little greater percentage of failed assignments in around 1300 vehicles.

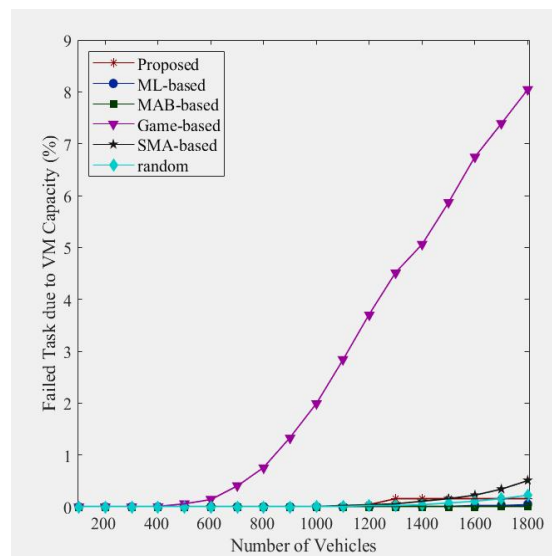


Figure 16. Failed Tasks Due to VM Capacity

The tasks failed because of the high mobility of vehicles presented in **Figure 17**. The tasks were considered as failed in case the vehicle moved out of the RSU range while uploading or downloading tasks. In case the vehicle moves out of RSU range, the server will send the results to the next RSU in the vehicle's direction of movement to finally get the results of tasks. The proposed model managed to achieve no failure due to mobility in a small number of vehicles, while the failure rate increased in high vehicle numbers. The proposed methods achieved the lowest failed rate compared to other methods. In the high number of vehicles, 1400 to 1800 vehicles, the ML-based methods achieved better performance than the proposed methods with a small difference.

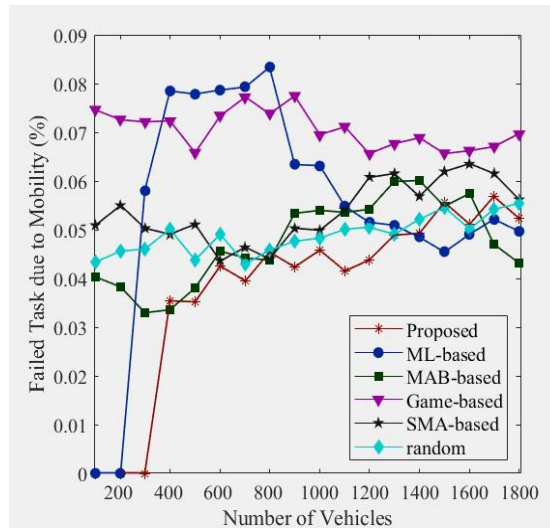


Figure 17. Failed Tasks Due to Mobility

The percentage of tasks that failed due to CN is presented in Figure 18. The tasks that are allocated to the cloud through the CN, but there are not enough resources, are considered failed. The proposed approach and game-based approach perform better than any other technique in terms of stability in all numbers of vehicles. It shows a relatively low percentage of tasks that fail even when the number of vehicles rises dramatically compared to the random method, which achieved the worst failure rate, reaching 35%.

The failed tasks due to WLAN limitation (bandwidth and congestion) are presented in Figure 19. The proposed model achieved a worse case primarily on the high number of vehicles reaching 0.15%. The lowest failure rate have been achieved by the random approach, which reached 0.01. Other approaches considered varied between the proposed method and the random method with small differences.

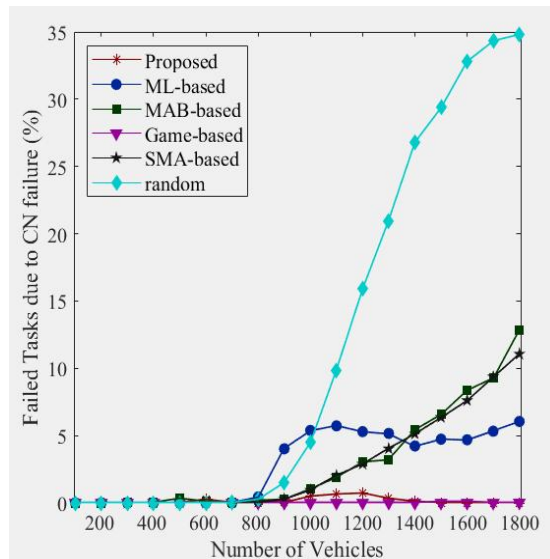


Figure 18. Failed Tasks Due to CN

A failed task caused by the WAN channel is presented in Figure 20. The proposed model achieved no failure task for vehicle numbers as the other methods in a small number of vehicles, less than 900. From 900 to 1300 vehicles, the proposed model achieves the highest failure rate while achieving a nearly stable failure rate in a high number of vehicles. The MAB-based approach failure rate rises fast – from 1400 to 1800 vehicles - to reach 17.9% as the maximum value. The proposed model achieves 6.1%, which is the highest failure rate in 1800 vehicles.

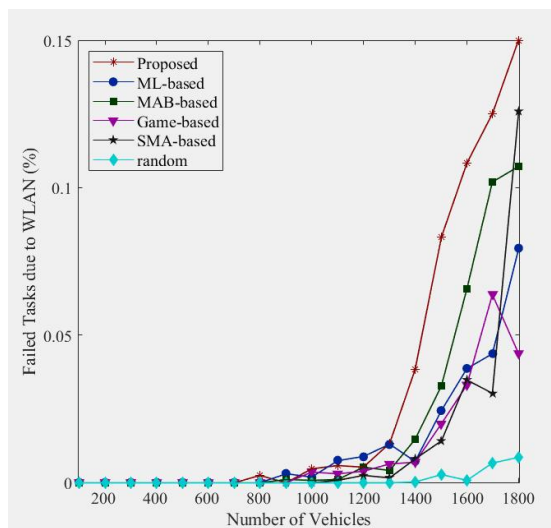


Figure 19. Failed Tasks Due to WLAN

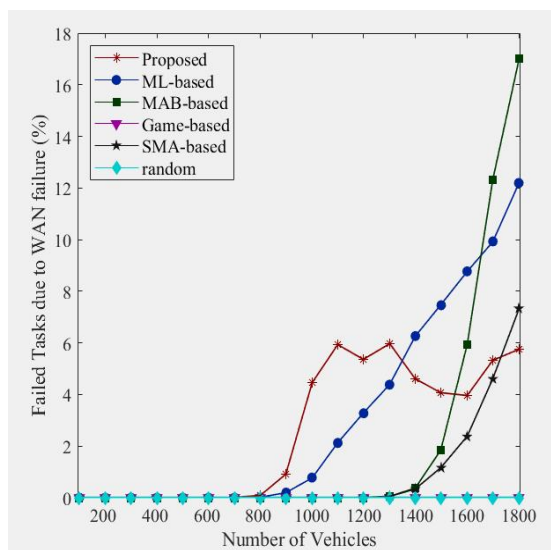


Figure 20. Failed Tasks Due to WAN

The proposed method achieved the highest performance in navigation applications in terms of average failure rate and QoE compared to other applications. This is because of the application characteristics and requirements. The smallest task length, and upload/download data minimize the burden of the network. On the other hand, navigation requires a strict maximum delay requirement of 0.5 seconds. The proposed model achieves low latency and fast data processing, which makes it appropriate for real-time navigation requirements. The navigation application is considered a lightweight application with rapid task generation.

The infotainment application has achieved the lowest performance. The highest task size and upload/download data requirements may experience increased failure rates and decreased QoE. Potential bandwidth and resource limitations cause degraded performance. Regardless of the flexibility and tolerance in the delay sensitivity requirement (0.25) and maximum delay requirement (1.5), the high upload and download data (20/80 KB) could lead to cognition when the number of vehicles increases. The infotainment application is considered the most tolerant compared to other applications with heavy data transferred. Also characterized by the longer task interval time (15 seconds).

Applications with stricter requirements (like navigation and danger assessment) benefit more from the proposed method, which is designed to minimize delays and handle rapid data processing. The lower failure rates in these applications indicate that the method is well-suited to meet their critical performance needs. The proposed model tends to achieve no failure rate and maximum QoE for a small number of vehicles while performing better than its competitors, especially in high vehicle numbers. The effectiveness and robustness of the neural network model are achieved by several metrics such as its architecture, parameters used, and, most

importantly, the dataset. The model is trained using a big dataset that contains only successfully executed tasks on certain layers. The success of the task couples with minimum service time. Hence, the model tends to send the task to the location that generally achieves successfully executed tasks with minimum service time, taking into account 14 input features.

CONCLUSION

Workload allocation is an online problem, and because its inputs are changing so quickly, formal optimization techniques are unable to address it effectively. To enhance system performance, workload allocation methods offer to transfer incoming tasks to the best processing unit. The location of execution must guarantee that the QoE meets the application requirements. Especially in highly dynamic systems, such as vehicular systems, safety on the road is inevitable. A neural network-based workload allocation method for multi-access, multi-layer VEC architectures is presented in this paper. The proposed method is capable of managing and efficiently handling unpredictable nonlinear systems by taking into account several factors. The effectiveness and robustness of the neural network model are achieved by the dataset used and the high number of input features that affect the decision accuracy. Hence, the model sends the task to the location that generally achieves successfully executed tasks with minimum service time. In order to compare the suggested model, five methods are considered which are random, SMA, MAB theory, ML, and game theory. The proposed methods achieved better results regarding QoE and average task failure rate. The proposed methods achieved 8.33% to 28.57% enhancement compared to other methods considered in terms of QoE and 50% enhancement in average failure rate compared to the best earlier methods, the ML-based approach. While the proposed model achieves stability and scalability by handling a high number of vehicles and providing efficient real-time decisions, several issues, such as data privacy and security, need to be addressed for real implementation. In future work, optimization methods will be considered for optimizing the features used as input to the model. The vehicle on-board processing tasks will be considered, and vehicle-to-vehicle communication will also be investigated to enable sharing of resources of other vehicles and reduce transmission costs.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

REFERENCES

- Ammar, S., & Dawood, A. (2024). AI workload allocation methods for edge-cloud computing: A review. *Al-Iraqia Journal of Scientific Engineering Research*, 2(4), 115-132. doi:10.58564/ijser.2.4.2023.125
- Biswas, A., & Wang, H. C. (2023). Autonomous vehicles enabled by the integration of IoT, edge intelligence, 5G, and blockchain. *Sensors*, 23(4), 1963. doi:10.3390/s23041963
- Chen, R., Fan, Y., Yuan, S., & Hao, Y. (2024). Vehicle collaborative partial offloading strategy in vehicular edge computing. *Mathematics*, 12(10), 1466. doi:10.3390/math12101466
- Dai, F., Liu, G., Mo, Q., Xu, W. H., & Huang, B. (2022). Task offloading for vehicular edge computing with edge-cloud cooperation. *World Wide Web*, 25(5), 1999-2017. doi:10.1007/s11280-022-01011-8
- Dos Anjos, J. C. S., Gross, J. L. G., Matteussi, K. J., González, G. V., Leithardt, V. R. Q., & Geyer, C. F. R. (2021). An algorithm to minimize energy consumption and elapsed time for IoT workloads in a hybrid architecture. *Sensors*, 21(9), 2914. doi:10.3390/s21092914
- Hao, T., Zhan, J., Hwang, K., Gao, W., & Wen, X. (2021). AI-oriented workload allocation for cloud-edge computing. *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 555-564. doi:10.1109/CCGrid51090.2021.00065
- Koulamas, C., & Lazarescu, M. T. (2018). Real-time embedded systems: Present and future. *Electronics (Switzerland)*, 7(9), 205. doi:10.3390/electronics7090205
- Liu, L., Chen, C., Pei, Q., Maharjan, S., & Zhang, Y. (2021). Vehicular edge computing and networking: A survey. *Mobile Networks and Applications*, 26(3), 1145-1168. doi:10.1007/s11036-020-01624-1
- Liu, Z., Jia, Z., & Pang, X. (2023). DRL-based hybrid task offloading and resource allocation in vehicular networks. *Electronics*, 12(21), 4392. doi:10.3390/electronics12214392
- Long, J., Luo, Y., Zhu, X., Luo, E., & Huang, M. (2020). Computation offloading through mobile vehicles in IoT-edge-cloud network. *Eurasip Journal on Wireless Communications and Networking*, 2020. doi:10.1186/s13638-020-01848-5
- Naha, R. K., Garg, S., Georgakopoulos, D., Jayaraman, P. P., Gao, L., Xiang, Y., & Ranjan, R. (2018). Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access*, 6, 47980-48009.
- Nguyen, V., Khanh, T. T., Oo, T. Z., Tran, N. H., Huh, E. N., & Hong, C. S. (2021). Latency minimization in a fuzzy-based mobile edge orchestrator for IoT applications. *IEEE Communications Letters*, 25(1), 84-88.
- Peixoto, M. J. P., & Azim, A. (2023). Design and development of a machine learning-based task orchestrator for intelligent systems on edge networks. *IEEE Access*, 11, 33049-33060. doi:10.1109/ACCESS.2023.3263483
- Sonmez, C., Ozgovde, A., & Ersoy, C. (2018). EdgeCloudSim: An environment for performance evaluation of edge computing systems. *Transactions on Emerging Telecommunications Technologies*, 29(11), e3493. doi:10.1002/ett.3493
- Sonmez, C., Tunca, C., Ozgovde, A., & Ersoy, C. (2021). Machine learning-based workload orchestrator for vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 22(4), 2239-2251. doi:10.1109/TITS.2020.3024233
- Sun, D., Chen, Y., & Li, H. (2024). Intelligent vehicle computation offloading in vehicular Ad Hoc networks: A multi-agent LSTM approach with deep reinforcement learning. *Mathematics*, 12(3), 424. doi:10.3390/math12030424
- Ullah, I., Lim, H. K., Seok, Y. J., & Han, Y. H. (2023). Optimizing task offloading and resource allocation in edge-cloud networks: A DRL approach. *Journal of Cloud Computing*, 12(1), 1-28. doi:10.21203/rs.3.rs-2522525/v1
- Wang, H., Peng, Z., & Pei, Y. (2020). Offloading schemes in mobile edge computing with an assisted mechanism. *IEEE Access*, 8, 50721-50732. doi:10.1109/ACCESS.2020.2979770
- Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X., & Chen, M. (2018). In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5), 156-165. doi:10.1109/MNET.2019.1800286
- Wu, Z., Jia, Z., Pang, X., & Zhao, S. (2024). Deep reinforcement learning-based task offloading and load balancing for vehicular edge computing. *Electronics*, 13(8), 1511. doi:10.3390/electronics13081511
- Yang, S., Lee, G., & Huang, L. (2022). Deep learning-based dynamic computation task offloading for mobile edge computing networks. *Sensors*, 22(11). doi:10.3390/s22114088
- Zhang, X., Cao, Z., & Dong, W. (2020). Overview of edge computing in the agricultural Internet of Things: Key technologies, applications, challenges. *IEEE Access*, 8, 141748-141761. doi:10.1109/ACCESS.2020.3013005