Cooperative Computational Offloading in Mobile Edge Computing for Vehicles: A Model-Based DNN Approach

Suleman Munawar, Zaiwar Ali, Muhammad Waqas, *Senior Member, IEEE*, Shanshan Tu, *Member, IEEE*, Syed Ali Hassan, *Senior Member, IEEE*, and Ghulam Abbas, *Senior Member, IEEE*

***Abstract*—Many advancements are being made in vehicular networks, such as self-driving, dynamic route scheduling, real-time traffic condition monitoring, and on-board infotainment services. However, these services require high computation power and precision and can be met using mobile edge computing (MEC) mechanisms for vehicular networks. MEC operates through the edge servers available at the roadside, also known as roadside units (RSU). MEC is very useful for vehicular networks because it has extremely low latency and supports operations that require near-real-time access to rapidly changing data. This paper proposes an efficient computational offloading, smart division of tasks, and cooperation among RSUs to increase service performance and decrease the delay in a vehicular network via MEC. The computational delay is further reduced by parallel processing. In the division of tasks, each task is divided into two sub-components which are fed to a deep neural network (DNN) for training. Consequently, this reduces the overall time delay and overhead. We also adopt an efficient routing policy to deliver the results through the shortest path to improve service reliability. The offloading, computing, division, and routing policies are formulated, and a model-based DNN approach is used to obtain an optimal solution. Simulation results prove that our proposed approach is suitable in a dynamic environment. We also compare our results with the existing state-of-the-art, showing that our proposed approach outperforms the existing schemes.**

***Index Terms*—Deep neural network (DNN), computational offloading, vehicular networks, smart task division, routing policy, mobile edge computing.**

Manuscript received 27 November 2021; revised 27 April 2022 and 25 August 2022; accepted 18 October 2022. Date of publication 26 October 2022; date of current version 14 March 2023. This work was supported in part by Beijing Natural Science Foundation under Grant 4212015, in part by the China Ministry of Education - China Mobile Scientific Research Foundation under Grant MCM20200102, and in part by the Deanship of Scientific Research at King Khalid University through Large Groups Project under Grant RGP.2/201/43. The review of this article was coordinated by Dr. Feng Ye. *(Corresponding author: Muhammad Waqas.)*

Suleman Munawar and Ghulam Abbas are with the Faculty of Computer Science and Engineering, Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi 23460, Pakistan (e-mail: u2017446@giki.edu.pk; abbasg@giki.edu.pk).

Zaiwar Ali is with the Faculty of Electrical Engineering, Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi 23460, Pakistan (e-mail: zaiwar.ali@giki.edu.pk).

Muhammad Waqas is with the Department of Computer Engineering, College of Information Technology, University of Bahrain, Sakhir 32038, Bahrain, and also with the School of Engineering, Edith Cowan University, Perth, WA 6027, Australia (e-mail: engr.waqas2079@gmail.com).

Shanshan Tu is with the Engineering Research Center of Intelligent Perception and Autonomous Control, Faculty of Information Technology, Beijing University of Technology, Beijing 100021, China (e-mail: sstu@bjut.edu.cn).

Syed Ali Hassan is with the School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad 44000, Pakistan (e-mail: ali.hassan@seecs.edu.pk).

Digital Object Identifier 10.1109/TVT.2022.3217323

I. Introduction

CLOUD computing technologies have become increasingly popular over recent years. They help many businesses to access the application software and storage over high-speed Internet connections [1]. Everyday activities, such as banking, e-mails, media streaming, and e-commerce, use the cloud services [2]. In vehicular networks, cloud computing enables computation for various applications, such as collision prevention, safety, blind crossing, dynamic route scheduling, and real-time traffic condition monitoring [3]. To further reduce the time delay and the transmission cost of the computation offloading, cloud-based mobile edge computing (MEC) offloading frameworks are proposed in vehicular networks. They not only save time but also alleviate the computational burden of vehicles. In vehicle-to-infrastructure (V2I) communication, the vehicles can send their computational tasks to the edge servers stationed at the roadside, known as roadside units (RSUs), which execute these demanding computational tasks. However, by comparing MEC with conventional mobile cloud computing (MCC), the performance margin in time delay becomes substantial using MEC instead of MCC [4] because of the proximity of MEC servers to the end-users/vehicles.

A significant objective in vehicular networks is to provide processing services to guarantee low delay and high reliability. However, the main hurdle is to select an appropriate task offloading and computation mechanism. The decision is difficult for vehicles, particularly in those regions where the communication range of RSUs is overlapping. The service delay can be reduced by implementing the task partition technique, and cooperative computing in MEC [5]. Task partition and cooperative computing are techniques in which a task can be divided into multiple components, and multiple RSUs can process the components in parallel. A plethora of literature available for these systems takes help from conventional and artificial intelligence paradigms. We have discussed some of this literature in Section II of "Related Works".

In this paper, we propose an efficient computational offloading, smart division of tasks, cooperation and efficient routing

0018-9545 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

among RSUs to minimize the computational service delay, increase service performance and improve service reliability in a MEC-enabled vehicular network. In our proposed method, when an RSU receives the computational task sent by a vehicle, it partially or entirely computes the task load instead of executing the entire computational task. It delivers the processed results to another RSU. The smart task division strategy is utilized to reduce the total service time delay using parallel processing [19]. In parallel processing, we assume that the computational tasks offloaded by vehicles are from those applications whose tasks can be divided into subtasks. However, the divided components of a task load are computationally independent of each other. By utilizing cooperation and efficient routing among RSUs, vehicles acquire the computation results without interruption due to high mobility. In this regard, we propose a model-based DNN approach to overcome the problem of dynamic conditions [20]. The main contributions of our proposed work are given as follows.

1) We propose a cooperative computing method among RSUs for vehicular networks operating via MEC. This reduces the time delay and provides high-quality services. Our approach can further reduce the total service time delay by smart division of tasks and parallel processing.
2) We also propose efficient routing to improve service reliability and reduce the failure of service sessions due to the high mobility of the vehicles so that the results get delivered back to the vehicle successfully.
3) The computation task offloading, division of task, task processing, and routing are challenging problems having high computational overhead. Therefore, we propose to solve these challenges using a deep learning approach to minimize the overall cost of service time delay. A DNN is trained and tested using an optimal dataset generated from the proposed method, which includes solving task offloading, division of tasks, parallel processing, and routing problems. This new approach is referred to as the proposed method with DNN.

The rest of the paper is organized as follows. After the introduction in Section I, related works are discussed in Section II, Section III explains the network model. Section IV formulates the total service time delay and routing method. The proposed model-based DNN solution is illustrated in Section V. Section VI explains the steps of dataset generation. The complexity analysis of the proposed approach concerning existing state-of-the-art is illustrated in Section VII. The simulation results are discussed in Section VIII. Finally, Section IX concludes the paper.

II. Related Works

In [6], a deep neural network (DNN) based MEC framework is explored, considering multiple mobile devices and one MEC server. A processing delay prediction mechanism is developed for typical DNN tasks to facilitate task partitioning. In [7], a task partition and scheduling scheme is proposed to divide the computing workload among edge servers and coordinate the execution order for tasks offloaded to the servers. After this, deep

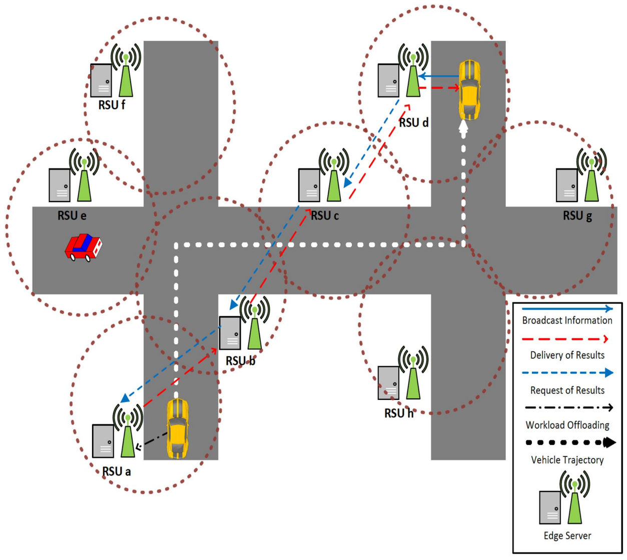


Fig. 1. System model.

reinforcement learning (DRL) is utilized to determine vehicle task offloading, computing, and result delivery policy.

In addition, due to the edge servers' high mobility and limited communication range, there is an in-service risk session [8]. The reason is that the vehicle changes its position rapidly and can be out of the communication range from the respective RSU, causing service failures [9]. The problem of deciding on offloading computational tasks in vehicular networks has been explored in [10], [11], and [12]. In these works, the fundamental goal is to reduce the total time delay. Nonetheless, the service dependability within sight of vehicle versatility is not considered. In [13], the problem of task offloading in sight of the vehicle's mobility is explored. The authors have proposed a mobility-aware and location-based task offloading scheme, where they jointly optimized the task offloading decisions, offloading time, and computation resource allocation.

To maintain service continuity using MCC schemes, dynamic service placement and migration have been examined in [14]. The service can be transferred to another cloud server as users move across different geological areas. Hence, the user may partner with another cloud server by the end of the service session [15]. Markov decision process (MDP) is used to decide on a dynamic service placement strategy [16], [17]. Nevertheless, the service interference and network overhead for virtual machine movement cannot be evaded in-service migration. Therefore, we can improve service reliability through cooperation between servers. The network overhead for service migration is further reduced by migrating the results of the processed task instead of migrating the entire task to be processed. Therefore, a learning-based methodology is proposed to make task offloading, and computing decisions [18]. The authors proposed vehicle-to-vehicle (V2V) correspondence to scatter the computing results if the edge servers cannot interface with the vehicle during the service session.

We compared our system model in Fig. 1 with the system models in [21] and [22]. In [21], the authors have utilized an

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

SDN-based load balancing task offloading scheme to minimize the processing delay of all the computation tasks. Moreover, the authors have considered two communication modes, V2V and V2I, and the vehicle will have to choose one. The vehicle will also have to choose one of the three offloading decisions for each vehicle to complete its computation task, i.e., executing on its CPU locally, offloading to the MEC server connected to RSU, and offloading to the remote cloud server. In both these decisions, the driver will input decisions, or the vehicle will compute something and then decide. In our paper, we proposed utilising DNN to minimize the overall service time delay, which does not need communication to make decisions. It will almost instantly provide the decision. In our paper, we want to reduce the load on vehicles as much as possible, which includes executing on its CPU locally and computing something to make offloading decisions.

In [22], the authors have proposed that each vehicle has two options to either process a task locally on its vehicular terminal or remotely on a MEC server. However, the authors have utilized just a single server to process a task. Moreover, the authors have considered a unidirectional road in their system model. Hence, if the results need to be delivered through wireless backhaul, then the results are sent in a specific direction rather than a specific route or path. In our paper, we have utilized at most two servers to process a task in parallel and to reduce the processing time delay. We have considered a multi-directional road in our system model. If the results need to be delivered through wireless backhaul, then the results will be delivered via the shortest route from receiver RSU to deliver RSU. Moreover, we have utilized DNN in our paper to minimize the overall service time delay.

In [34], the authors have considered mobile devices (MD) rather than vehicles. They have considered a single MD and multiple MEC servers in their MEC network. The authors have proposed that the MD has two options to either process a task locally on the device or remotely on a MEC server. Moreover, they have proposed a parallel transmission and execution (PTE) scheme to reduce overall processing time. However, if an MD is transmitting its task to a MEC server, its transmission resource is busy, so it cannot transmit other tasks at that time. Furthermore, the tasks are classified into high priority tasks (HPTs), medium priority tasks (MPTs), and low priority tasks (LPTs) based on their task execution status and causal relationship. After completing the classification, task scheduling strategies are applied to determine the task execution order. These classification and task scheduling strategies are designed and computed locally on the MD. In our paper, we have considered multiple vehicles and multiple MEC servers in the MEC system. Due to low computing power, we want to reduce the load on vehicles as much as possible, which includes executing on its CPU locally and computing something to make offloading decisions. We have utilized at most two servers to process a task in parallel and to reduce the processing time delay. Moreover, we have utilized DNN in our paper to minimize the overall service time delay significantly.

In [35], the authors have utilized a distributed multi-hop task offloading decision model to make the task execution more efficient. The authors have utilized vehicles to make two offloading decisions: a candidate vehicle selection mechanism for screening the neighboring vehicles that can participate in offloading using the greedy method and a task offloading decision algorithm design part for obtaining the task offloading solution using the bat algorithm. Hence, the vehicles can either process a task locally on their vehicular terminal or remotely on a MEC server. In our paper, we proposed utilizing DNN to minimize the overall service time delay, which will instantly provide the decision. Since we want to reduce the load on vehicles as much as possible due to their low computation power, we did not utilize the local CPU of vehicles for either execution of tasks or to make offloading decisions.

III. Network Model

Fig. 1 depicts an area where several RSUs are located at the roadside. We denote the set of RSUs as $\mathbb{R}$. Each RSU acts as a MEC server for vehicles and communicates with other RSUs within its communication range via wireless links. We assume an epoch-based timeline in this work.

At epoch t, a vehicle interacts with an RSU from a certain distance. In this interaction, the vehicle offloads information about its trajectory and a task of a certain size. The data size of the computational task is denoted by $C(t)$. Moreover, the RSU gives an ID to the computational task offloaded by the vehicle and sends its ID and the allotted task ID back to the vehicle. It is assumed that the interaction of a vehicle at any epoch must be completed before the next epoch. We consider a global controller that makes the offloading and computing decisions for all the vehicles in a centralized manner. At epoch t, the offloading decision for vehicles is represented by a binary vector, $\epsilon_{a(t)}$, where $a \in \mathbb{R}$. If $\epsilon_{a(t)} = 1$, the computation tasks of a vehicle is offload to RSU "a" at epoch t. If $\epsilon_{a(t)} = 0$, the computation tasks of a vehicle is not offloaded to RSU "a" at epoch t. "a" is referred to as the receiver RSU for computation tasks generated at epoch t. The controller may select another RSU, which could jointly process a part of the computation task and transmits the results back to the vehicle.

An example is shown in Fig. 1. At epoch t, a vehicle offloads its computational task to RSU a. To reduce time delay, RSU $a \in \mathbb{R}$ needs cooperation from another RSU to complete a part of the task by computing in parallel. In addition to selecting RSUs to participate in the service, the controller also determines the smart task partition policy. The computation task is intelligently divided into two sub-components and computed in parallel. The RSU a keeps a portion and sends the rest of the task to the nearest RSU in the direction of the vehicle's trajectory, i.e., RSU $b \in \mathbb{R}$, as depicted in Fig. 1. Both these RSUs process the task in parallel. Upon completing each part of the task, RSU a sends its result to RSU b for combing. The complete result will be available at RSU b, referred to as *helper RSU* for computation tasks generated at epoch t. A helper RSU is the one that either computes the sub-component of the computational task in parallel with another RSU or computes the complete computational task by itself. Still, it may not deliver the result back to the vehicle.

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

The considered model limits the number of RSUs jointly processing the computation tasks to two servers. We denote the cooperative computing decision as a binary vector $\zeta_b(t)$, where $\zeta_b(t) = 1$, RSU b will join the computing session. However, if the vehicle is in the communication range of RSU b, the RSU b will deliver the result of a task offloaded by a vehicle at epoch t back to the vehicle. If $\zeta_b(t) = 0$, RSU b will not join the computing session, but it may deliver the result of a task offloaded by a vehicle to RSU a at epoch t back to the vehicle. The RSU, which transmits the result back to the vehicle, will be referred to as the *deliver RSU*.

Because of high mobility, a vehicle may travel out of the communication range of both RSU a and RSU b. Such that the result will not get delivered back to the vehicle. To improve the service reliability, the helper RSU b sends the combined result back to receiver RSU a such that the complete result will be available at receiver RSU a. The vehicle broadcasts information to all the servers it passes through the region, as shown in Fig. 1. This information is the receiver RSU's ID to which the vehicle offloaded its task and the task's ID. When any RSU other than the receiver RSU and helper RSU notices the ID of the receiver RSU, such as RSU d in Fig. 1, it creates the shortest route from itself to the receiver RSU a. Afterwards, RSU d will send a message to the receiver RSU a, requesting the complete result of the task with the ID given by the vehicle. If the receiver RSU a has the complete result, it will send the result via the shortest route from RSU a to RSU d. However, if RSU a could not complete the result, it will send a message via the route that the result is not found yet. When RSU d gets the complete result, it will forward it to the vehicle. In such a case, RSU d will be referred to as deliver RSU. Although, it might forward a message to notify that the result is not found.

The vehicle stops broadcasting the receiver RSU's and task IDs once the vehicle gets the complete result. Hence, the service session is terminated. On the other hand, the vehicle continues to broadcast the information until it receives the results. The vehicle gets the message that the result is not found. Our proposed model assumes a threshold value according to the service delay. Due to the threshold value, the vehicles do not have to wait long for the task to be processed. The vehicle stops broadcasting the receiver RSU and task IDs in such a case. Subsequently, the vehicle offloads its task to another RSU, which will then become the receiver RSU.

The controller also intelligently divides the task into two components, i.e., the computation task is not divided arbitrarily. Only those divided components are selected who would give the minimum time delay. For computation tasks offloaded at epoch t, the divided portions of the computational task to be processed by the receiver RSU and the helper RSU selected by the controller are denoted by $r_a(t)$ and $h_b(t)$, respectively, such that, $C(t) = r_a(t) + h_b(t)$. Hence, if $\epsilon_a(t) = 1$ and $\zeta_b(t) = 1$, we have the following three conditions and corresponding cases in our system model.

Condition # 1: $r_a(t) = C(t)$ and $h_b(t) = 0$.

1) **Case # 1:** The computing task offloaded at epoch t to RSU a is processed only at RSU a, which acts as both receiver and deliver RSU.
2) **Case # 2:** The computing task offloaded at epoch t to RSU a is processed at RSU a. However, the result is delivered by RSU b. It means that RSU b is delivered RSU and not the helper RSU.
3) **Case # 3:** The computing task offloaded at epoch t to RSU a is processed only at RSU a. However, the result is delivered by RSU a via the shortest route. This means that an RSU other than the receiver RSU will be the deliver RSU.

Condition # 2: $r_a(t) > 0$ and $h_b(t) > 0$

1) **Case # 1:** The computing task is offloaded to the receiver RSU a, it forwards $h_b(t)$ out of the computational task to helper RSU b. The RSU b and RSU a process the divided components of the computational task in parallel. The combined result is delivered from RSU a to the vehicle. In such a case, RSU a acts as the deliver RSU.
2) **Case # 2:** Once the computing task is offloaded to the receiver RSU a, it forwards $h_b(t)$ out of the computational task to helper RSU b. The RSU b and RSU a process the divided components of the computational task in parallel. The combined result is delivered from RSU b to the vehicle. Hence, RSU b will act as the helper and deliver RSU.
3) **Case # 3:** Once the computing task is offloaded to the receiver RSU a, it forwards $h_b(t)$ out of the computational task to helper RSU b. The RSU b and RSU a process the divided components of the computational task in parallel. However, the combined result is delivered by RSU a via the shortest route. It means that an RSU other than RSU a or RSU b will be the deliver RSU.

Condition # 3: $r_a(t) = 0$ and $h_b(t) = C(t)$

1) **Case # 1:** The computational task offloaded at epoch t to RSU a is processed only at RSU b, which acts as a helper and delivers RSU.
2) **Case # 2:** The computational task offloaded at epoch t to RSU a is processed only at helper RSU b. However, the result is delivered by RSU a via the shortest route. It means that an RSU other than RSU a and RSU b will be the deliver RSU.

A. Cost Model

We consider the cost model as the service delay, including propagation and computation delays.

1) Propagation Delay: The propagation model follows the 3GPP standards [23]. The path-loss between a transmitter and a receiver with distance, S can be computed as

$$P(S) = 40\left(1 - 4 \times 10^3 H\right)\log_{10} S - 18\log_{10} H + 21\log_{10} f + 80, \tag{1}$$

where H represents the antenna height and f is the carrier frequency. The distance between RSU a and RSU b as $S_{(a,b)}$. Hence, the data rate for the vehicle to RSU a is

$$\mathcal{R}_a = B_M \log_2\left(1 + \frac{T^M 10^{-P(S_a)/10}}{\delta_m^2}\right), \tag{2}$$

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

where T^M represents the vehicle transmit power, and B_M represents the bandwidth reserved for a vehicle. The data rate from RSU a to RSU b for forwarding the computation task offloaded is

$$\mathcal{R}_{a,b} = B_D \log_2 \left(1 + \frac{T_D 10^{-P(S_{a,b})/10}}{\delta_D^2}\right), \quad (3)$$

where T^D represents the transmit power of RSU, and B_D represents the bandwidth reserved for forwarding the task offloaded at epoch t. The parameters δ_m^2 and δ_D^2 denote the power of the Gaussian noise in the channel of vehicle-to-RSU and RSU-to-RSU, respectively. The network deployment is based on orthogonal frequency division multiple-access (OFDMA). We assume that the bandwidths, B_M, reserved for a vehicle and B_D reserved for RSU for forwarding the task offloaded at epoch t is divided into $\mathbb{K}_M$ and $\mathbb{K}_D$ subcarriers, respectively. Let $k_M \in \mathbb{K}_M = [1, 2, \ldots, M]$ and $k_D \in \mathbb{K}_D = [1, 2, \ldots, D]$ denote the available subcarriers to be allocated in each component execution period. OFDMA is a usual approach to consider in modelling of system model [33]. Related works of this paper have considered OFDMA [13], [27], and [33]. Therefore, this paper also considers OFDMA for comparing it with the related work. It is assumed that the channel condition is fixed for the duration of task offloading. Thus, the transmission delay for offloading the computation task at epoch t to RSU a is

$$\tau^a(t) = \frac{\epsilon_a(t) \times C(t)}{\mathcal{R}_a}, \quad (4)$$

and the forwarding delay at epoch t to RSU b is

$$\tau^b(t) = \frac{\epsilon_a \times \zeta_b(t) \times h_b(t)}{\mathcal{R}_{(a,b)}}. \quad (5)$$

Since the amount of output data is smaller than the amount of input data, we neglect the receiving delay for result delivering and sending messages [24], [25]. However, the receiving delay for result delivery can be considered as

$$\tau^R(t) = \frac{\sigma \times C(t)}{\mathcal{R}_{(d)}}, \quad (6)$$

where σ represents a value used to approximate the size of the result concerning the size of the computational task. Also, $\mathcal{R}_{(d)}$ is the data rate to send the result from an RSU to the vehicle as given by

$$\mathcal{R}_d = B_M \log_2 \left(1 + \frac{T_D 10^{-P(S_{L_m}(t))/10}}{\delta_M^2}\right), \quad (7)$$

where $S_{L_m}(t)$ is the distance between the deliver RSU and vehicle m at epoch t.

2) Computation Delay: Once computation task is offloaded by vehicle, the receiver RSU a, processes $r_{a(t)}$, a portion of computation load and forwards $h_{b(t)}$, a portion of computation load to the helper RSU b. The processing time delay for computing task offloaded to RSU a at epoch t is

$$\tau_a^E(t) = \frac{N\left[\epsilon_a(t) \times r_a(t)\right]}{F}, \quad (8)$$

TABLE I
DIVISIONS OF COMPUTATION TASKS

Index of divs [3_{Z2}][2]	Server 1 Load	Index of divs [3_{Z2}][2]	Server 2 Load
divs [0][0]	3 MB	divs [0][1]	0 MB
divs [1][0]	2 MB	divs [1][1]	1 MB

where $\epsilon_a(t)$ is a binary vector representing the offloading decision for a vehicle at epoch t for RSU a. The processing time delay for computing task offloaded to RSU b at epoch t is

$$\tau_b^E(t) = \frac{N\left[\epsilon_a(t) \times \zeta_b(t) \times h_b(t)\right]}{F}, \quad (9)$$

where F denotes the computing capability (CPU-cycle frequency) of an RSU, and N denotes the number of computation cycles needed to execute one bit.

We will get the processing time delay only if both the binary vectors $\epsilon_a(t)$ and $\zeta_b(t)$ are equal to one, which means that the vehicle has decided to offload its task to RSU a and the controller has selected the RSU b to process either a sub-component of the task or the whole task. Furthermore, the offloaded computation task is processed only if the computing tasks offloaded in previous epochs are completed. For the task offloaded to RSU a, the total computation delay (queuing delay and processing time delay) is formulated as

$$\tau_a^D(t) = \tau_a^Q(t) + \tau_a^E(t) = \max\{\tau_a^D(t-1) + 2k, 0\} + \tau_a^E(t), \quad (10)$$

where the value of k is the time length of an epoch. For the tasks offloaded to RSU b, the total computation delay is

$$\tau_b^D(t) = \tau_b^Q(t) + \tau_b^E(t) = \max\{\tau_b^D(t-1) + k, 0\} + \tau_b^E(t), \quad (11)$$

where $\tau_a^Q(t)$ and $\tau_b^Q(t)$ represent the queuing delay for the tasks offloaded to RSU a and RSU b at epoch t, respectively. In (10) and (11), the first term of max[$\cdot$] is the total computation delay of the previous task, which was offloaded to an RSU, and the second term is the value zero. If a task gets offloaded to an RSU with no incomplete tasks in its queue, then the first term may have a random negative value. Hence, for the next task, the max[$\cdot$] will choose the first term as the value for queueing delay. The division of computational tasks offloaded by vehicles is performed in the following manner.

Each computational task is divided into two sub-components computationally independent of each other. Both the divided portions are jointly called a combination, such that the size of each combination is the size of the total computational task. The number of combinations is $C[i]_{Z_g}$, where $C[i]$ is the size of the total computational task in MBs, and g is the number of components in which a task is to be divided into. The difference between the respective portions of each consecutive combination is 1 MB. Hence, the combinations are all the unique divisions of the computational task in two sub-components. For example, Table I shows the divisions of a computational task of size 3 MB, where the number of combinations is 3_{Z2}. We store these combinations in a 2D array called "divs". Each combination

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

has multiple offloading policies according to the number of RSUs used in the parallel processing. For instance, we are using two RSUs for parallel processing in our paper and in the first combination, there can be two offloading policies, [3, 0] meaning that server 1 will have 3 MB load and server two will have 0 MB load, and [0, 3] meaning that server 1 will have 0 MB load and server two will have 3 MB load.

IV. Problem Formulation

Our main focus is to reduce the total service delay and increase vehicle reliability. Hence, for a vehicle offloading the computation task at epoch t, the total service delay is

$$\tau^{\circ}(t) = \tau^{a}(t) + \max\{\epsilon_a(t)\tau_a^D(t), \tau^b(t) + \zeta_b(t)\tau_b^D(t)\}, \quad (12)$$

where the decision variables of (12) are $\epsilon_a(t)$ and $\zeta_b(t)$ whose values will decide the total service delay. The variables $\tau^a(t)$, $\tau_a^D(t)$, $\tau^b(t)$, and $\tau_b^D(t)$ are the delay variables of (12) because their values are affected by the decisions made by decision variables.

The use of the maximum function in (12) is explained below. In terms of the division of computational tasks into sub-components, there are three scenarios.

- Receiver RSU a will compute the total computational task, and helper RSU b does not compute anything. In this situation, the binary vector, $\zeta_b(t)$, will equal zero. It means that both $\tau^b(t)$ and $\tau_b^D(t)$ will also be zero. Hence, the maximum function selects $[\epsilon_a(t)\tau_a^D(t)]$.
- The helper RSU b will compute the total computational task, and receiver RSU a does not compute anything. In this situation, $\tau_a^D(t)$ will be zero. Hence, the max function will select $[\tau^b(t) + \zeta_b(t)\tau_b^D(t)]$.
- Both receiver RSU a and helper RSU b will compute a sub-component of the total computational task. In this situation, neither $[\epsilon_a(t)\tau_a^D(t)]$ nor $[\tau^b(t) + \zeta_b(t)\tau_b^D(t)]$ will be zero. The maximum function selects the greater value of between $[\epsilon_a(t)\tau_a^D(t)]$ and $[\tau^b(t) + \zeta_b(t)\tau_b^D(t)]$. This is because both RSUs are computing in parallel.

Note that we have not included the receiving delay for result delivery (6) in the total service delay (12). The reason for this is because we are assuming that the size of output data is smaller as compared to the size of input data [24], [25]. However, we can include the receiving delay for result delivery in the total service delay. Still, it will have a minimal effect because the size of output data is much smaller as compared to that of input data.

A. Optimization Constraints

We have designed some constraints for our optimization problem that should be considered when calculating the total service delay.

1) Time Threshold for Total Service Delay: We have placed a threshold for the maximum time delay a computational task can take to be processed. If a computational task takes more time than the threshold to be processed, then the whole computational task is discarded, even if it is partially processed. We have kept a scalable value threshold of 15 minutes. Therefore, the value of total service delay should be in the range of 0 to 15 minutes.

2) SNR Threshold for V2I: A computational task can be offloaded to an RSU when it satisfies a signal-to-noise ratio (SNR) threshold, ω°. Therefore,

$$\epsilon_a(t) = \begin{cases} 1, & \text{if } \frac{T^M 10^{P(S_a)/10}}{\delta_m^2} \leq \omega^{\circ} \\ 0. & \text{Otherwise} \end{cases} \quad (13)$$

3) SNR Threshold for I2I: A computational task or sub-component of the task can be forwarded from a receiver RSU to a helper RSU only when it satisfies an SNR threshold, ω°. Therefore,

$$\zeta_b(t) = \begin{cases} 1, & \text{if } \frac{T^D 10^{P(S_{a,b})/10}}{\delta_D^2} \geq \omega^{\circ} \\ 0. & \text{Otherwise} \end{cases} \quad (14)$$

In addition, if a vehicle moves out of the communication range of RSU b at the end of the service session fails when $\zeta_b(t) = 1$. The SNR threshold for result delivery is denoted as ω^D. A variable $\Phi_m(t)$ is kept indicating failure of the service session for vehicle m at epoch t. Hence, we have

$$\Phi_m(t) = s.t. \begin{cases} 1, & \text{if } \frac{T^D 10^{-P(S_{L_m}(t+\tau^{\circ}))/10}}{\delta_D^2} \leq \zeta_a(t)\omega^D \\ 0. & \text{Otherwise} \end{cases} \quad (15)$$

The distance between the deliver RSU and vehicle m at epoch t is denoted as $S_{(L_m)}(t + \tau^{\circ})$. We consider the vehicle's distance when the offloaded task has been computed, and the results are obtained at the delivered RSU. Therefore, we add the time of total service delay at epoch t. For the routing policy, we use an efficient routing algorithm. Note that a routing policy is adopted only for migrating results to increase the service reliability. The algorithm finds the shortest path between an RSU and all the nearest RSUs. Therefore, each RSU knows the shortest path to every nearest RSU. The time complexity of these policies is $O(C.C_{Z_g}.2^g.r^2)$, which increases the time delay significantly. Therefore, we need an intelligent approach to reduce the time delay. The C in $O(C.C_{Z_g}.2^g.r^2)$ represents the data size of the total computational task. The rest of the values are explained in Table V.

The potential solutions include ANN, deep learning, and federated learning. Implementing federated learning costs more than collecting and processing the information centrally [26]. Federated learning requires frequent communication between nodes during the learning process. Thus, it requires not only enough local computing power and memory but also high bandwidth connections to be able to exchange parameters of the machine learning model. The federated learning technology avoids data communication, but it can require significant resources before starting centralised machine learning [26]. In our paper, we want to reduce the load on vehicles as much as possible, which includes avoiding huge storage devices. Moreover, even if the vehicles integrate high computing power capabilities, they can never be powerful enough to compute tasks in real-time. Therefore, we preferred the ANN approach over federated learning. ANN is widely used in MEC for vehicular networks, as also shown in [27], [28], and [29].

We designed an ANN model with a single hidden layer and tested it several times for different numbers of neurons in the

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

TABLE II
DISTANCES BETWEEN EACH SET OF RSUS

RSU	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1	0	200	0	0	0	0	0	0	0	0	0	0	0	0
2	200	0	0	220	0	0	300	0	0	0	0	0	0	0
3	0	0	0	250	0	0	0	0	0	0	0	0	0	0
4	0	220	250	0	280	0	0	0	0	0	0	0	0	0
5	0	0	0	280	0	220	310	0	0	0	0	0	0	0
6	0	0	0	0	220	0	0	0	0	0	0	0	0	0
7	0	300	0	0	310	0	0	300	0	0	0	0	0	0
8	0	0	0	0	0	0	300	0	300	0	260	0	0	0
9	0	0	0	0	0	0	0	300	0	280	0	0	250	0
10	0	0	0	0	0	0	0	0	280	0	0	0	0	0
11	0	0	0	0	0	0	0	260	0	0	0	200	250	0
12	0	0	0	0	0	0	0	0	0	0	200	0	0	0
13	0	0	0	0	0	0	0	0	250	0	250	0	0	310
14	0	0	0	0	0	0	0	0	0	0	0	0	310	0

TABLE III
VALUES USED ONLY IN TABLE V AND THE ALGORITHM 1

Value	**Description**
n	Rows of input array
k	Columns of input array
z	z is workload
a	Lower limit of data range of S_{L_m}
b	Upper limit of data range of S_{L_m}
c	Lower limit of data range of $S_(a, b)$
d	Upper limit of data range of $S_(a, b)$
e	Lower limit of data range of C
f	Upper limit of data range of C
g	Number of components a task can be divided into
r	Total number of RSUs

hidden layer each time. Then we redesigned the ANN model and tested it several times for different numbers of hidden layers and neurons in each hidden layer. We observe a 10%–15% increase in accuracy rate for an ANN model with 5 hidden layers having $[10, 10, 10, 10, 11]$ neurons. Therefore, we preferred the deep learning approach over the ANN approach with a single hidden layer. A multi-layer neural network that contains two or more hidden layers is called a deep neural network [30]. Our ANN model has 5 hidden layers. In other words, we are using deep learning in our paper. We use a model-based DNN approach to obtain efficient computational offloading, innovative division of tasks, cooperation among RSUs, and efficient routing policy.

In our paper, we have targeted applications whose output data size is smaller than the input data size. For example, consider two types of vehicular applications: "Finding the nearest gas station" and "road safety applications".

- **Finding nearest gas station:** Consider a user in a vehicle who wants to find a gas station for refuelling. The user will connect to an RSU and ask it to find a gas station. Along with this, other information from many sensors would also be required. These will include the current fuel level, the vehicle's speed, the vehicle, the precise location of the vehicle, the direction of the vehicle's trajectory, and the type of fuel the vehicle runs on. All this data is offloaded to RSU. The RSU will process it and produce a result. This result will include the shortest route to an appropriate gas station. Sometimes a vehicle's fuel will not be enough to reach the gas station. Therefore, sometimes the result will also suggest adjusting the vehicle's speed. In this application, the size of the result is much less than the input data.
- **Road safety applications:** Road safety applications take a lot of input data through various sensors from multiple vehicles, including information from weather sensors, temperature sensors, brake oil sensors, the precise location of a vehicle, pictorial view of the road, etc. After the task is processed, the result will include only a safety message informing a vehicle user that the road ahead is either safe to travel or not and the reason why it is not safe for travelling. In these vehicular applications, the output's data size is much less than the input's.

V. MODEL-BASED DNN APPROACH

We considered two intersections of roads joined together with 14 distributed RSUs and generated a dataset of 2,744 values. The distance between a vehicle and a receiver RSU, i.e., S_a, is considered a uniform random variable, i.e., $S_a \sim [25–318]$ m. The distance between the receiver RSU and vehicle, $S_{(L_m)}$, when the computational task is processed completely, and the result is to be delivered to the vehicle, is considered as a uniform random variable in the range [100–1000] m. The data size of the total computational task, $C(t)$, is also considered a uniform random variable with a range of [5–570] MB. The distances between each set of RSUs are shown in Table II in meters, where a 0 value of distance indicates the absence of connection between those two RSUs.

In the DNN model, we used a single input layer, 5 hidden layers and 1 output layer. The input layer has 3 neurons, hidden layers have $[10, 10, 10, 10, 11]$ neurons, respectively, and the

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

Algorithm 1: Dataset Generation.

```
Input: Table II, Table VI, $S_{L_m}$ $[n] \in \{a-b\}$m, $S_a[n] \in \{c-d\}$m, $C[n] \in \{e-f\}$MB, divs$[C+1]$ for each $C$,
Output: $s_1[n]$, $s_2[n]$, $d_s[n]$, $\Phi_m$ $[n]$, path$[n][r]$
Combine $S_{L_m}$, $S_a$ and $C$ to make a cartesian product of $[n][3]$ rows and columns.
Check from the cartesian product of $S_a$ follows the SNR constraint from (15).
for $i$ = 0 : (n-1) do
    Set delay1 $[2^g + 1]$;
    Set delay2 $[C[i]_{Z_g} + 1]$;
    for $j$ = 0 : ($C[i]_{Z_g}$ - 1) do
        Divide: task $C[i]$ into g components.
        for $j$ = 0 : ($2^g$ - 1) do
            Calculate: Transmission delay for offloading computational task to the receiver RSU for $i$ using (4);
            Calculate: Computation delay for divs[k][0] using (10);
            Calculate: Transmission delay for forwarding the computational task to RSU b using (5);
            Calculate: Computation delay for divs[k][1] using (11);
            delay1[k] = $\left[(4) + \max(10), ((5) + (11))\right]$
        end
        Calculate: minimum from delay1$[2^g + 1]$ array and store it in delay2$[C[i]_{Z_g} + 1]$ array.
    end
    Calculate: minimum from delay2$[C[i]_{Z_g} + 1]$ array and store the index in $j$.
    $s_1[i]$ = load on server 1
    $s_2[i]$ = load on server 2
    if $delay2[j] \geq 900$ then
        $S_{L_m}$[i] = 0;
        $\Phi_m$[i] = 1;
        else
        $\Phi_m$[i] = 0;
    end
    Use Dijkstra's algorithm to find the shortest path to send results through routing between servers.
    Save the path in path$[i][r]$ array and the information of delivery RSU in $dS[i]$.
end
```

Algorithm 2: Network Training.

```
Divide the dataset into training, validating and testing sets.
Network ⟵ DNN[10, 10, 10, 10, 11]
while $i \leq dataset$ do
    trained = train (network, inputs, labels)
end
Test performance on the test set.
Accuracy ⟵ Number of correctly predicted policies / Total number of policies.
```

output layer has 18 neurons. The activation function of the first four hidden layers is $tanh$, and that of the fifth layer is linear/identity activation function [31]. The design of the DNN model was obtained after rigorous testing as we noticed a $10\% - 15\%$ decrease in the accuracy rate for other values of hidden layers and the number of neurons in each hidden layer.

We trained the network by using the generated dataset. The values of our generated dataset are stored in a 2D array, with several rows equal to the size of the dataset, as illustrated in Table IV. In each row, there are 21 values, which are explained further. At index 0, the distance between receiver RSU and vehicle before offloading the computational task. At index 1, the distance between receiver RSU and vehicle after the computational task has been processed completely. At index 2, the size of the total computational task. Hence, these three values are given as input to DNN; therefore, 3 neurons for the input layer. Indices from 3 to 6 are sub-component of the computational task for receiver RSU, sub-component of the computational task for helper RSU, ID of delivering RSU, and value of variable $\Phi_m(t)$, respectively. Indices from 7 to 20 contain the shortest path from the receiver RSU to the deliver RSU, where index 7 always includes the ID of the receiver RSU, index 8 includes the ID of the RSU nearest to the receiver RSU in the shortest path, and so on, till the ID of the deliver RSU, for the rest of the values. If the ID of the deliver RSU comes at a value before 20, then all the values after it are kept to zero. All these values from 3 to 20 are the output values, and these are 18 in total. Hence 18 neurons for the output layer. In this paper, we have designed and trained our DNN model for a maximum of 14 RSUs in the network. In our scenario, this model is scalable and can work for any possible number of RSUs in the network between minimum and maximum, i.e., from 1 to 14 RSUs.

A common problem in machine learning and data science is overfitting [32]. Overfitting occurs when a neural network model does not generalise well from the training data to the testing data. Therefore, we split our dataset into training, validation, and testing subsets. Consequently, our proposed generated dataset has three subsets, i.e., training, validation, and testing. This process is a robust measure against overfitting. Hence, our method can detect overfitting using the train-validate-test process for the proposed DNN method. It decides if over-fitting has occurred and the learning process has been saturated. The model sees that the error on the training set decreases while the error on the validation set remains constant or begins to decrease. At this point, the model stops the training and moves toward the testing process, or it might lead to overfitting. The validation set and the testing set differ because the validation set is not as challenging as the testing set. The validation set contains unseen data, but the outputs' variations are insignificant. We divide the dataset as 70% for training, 15% for validation and 15% for testing.

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

TABLE IV
DATASET

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0	25	100	11	11	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	25	100	30	28	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2742	318	1000	446	388	58	10	0	1	2	7	8	9	10	0	0	0	0	0	0	0	0
2743	318	1000	570	523	47	14	0	1	2	7	8	11	13	14	0	0	0	0	0	0	0

As a result, our proposed model offers an intelligent division of tasks and the shortest path to deliver to the vehicle user. We used MATLAB to produce, train, validate, and test the DNN model. Note that we did not use any platform to simulate traffic and moving vehicles. However, all the simulation parameters used in this manuscript were kept very close to the values in a practical scenario. The proposed approach is summarized in Algorithm 1 and Algorithm 2.

VI. Dataset Generation

In the first step, each of the 2744 distances between receiver RSU a and the vehicle before offloading the computational task are checked through (15) to ensure that the SNR constraint is followed. If the constraint is not followed, it implies that the vehicle is far from the receiver RSU a and cannot offload its task to the receiver RSU a. If the constraint is followed correctly, the vehicle is at an acceptable distance from the receiver RSU a and can quickly offload its task to process.

The time delay is calculated for each computational task in the next step. Each computational task is divided into two sub-components processed by receiver RSU a and helper RSU b in parallel. Both the divided portions are jointly called a combination. Each computational task has multiple combinations of all the unique divisions of the computational task in two sub-components. Each combination has multiple offloading policies to decide which portions of the task will be processed by receiver RSU and helper RSU. This is explained with an example in Table I in Section III. Time delay is calculated for all unique combinations for each task, which includes transmission delay for offloading computational task to the receiver RSU a, using (4), computation delay for the first sub-component using (10), transmission delay for forwarding the second sub-component to helper RSU b, using (5), and computation delay for the second sub-component using (11). Each delay is added using (12). We now have multiple time delays for a single combination of a single computational task. The minimum time delay is calculated for a single combination and stored in an array from each time delay. From this array, the minimum time delay is calculated for a computational task, and the combination with the minimum time delay is stored in the dataset. This minimum time delay is compared to a threshold. If the time delay is less than the threshold, the task has been processed successfully. But suppose the time delay is greater than or equal to the threshold. In that case, the binary value in the dataset, which denotes the failure in processing a computational task, is set to one, which means that the task could not be processed entirely in the allocated time.

In the next step, among the receiver RSU a, and helper RSU b, the delivery RSU is determined by checking the SNR constraint from (15). After this, the information of delivery RSU is also stored in the dataset for each value. In this step, there are three cases in which the delivery RSU is determined.

- First, when the sub-component of the receiver RSU a is equal to zero, it implies that the whole computational task will be processed by helper RSU b, which will also act as the delivery RSU. Hence, the distance between receiver RSU a and delivered RSU b is subtracted from the above distance to get the distance between helper RSU b and vehicle. Furthermore, this subtracted distance is checked through (15) to ensure that the SNR constraint is followed. If the constraint is followed properly, then the binary variable, Φ_m is set to zero, which means that the vehicle was at an acceptable distance from the delivery RSU b and received the result successfully. However, if the constraint is not followed properly, then Φ_m is set to one, which means that the vehicle is far from the delivery RSU b, and the result could not be received by the vehicle.
- Second, when the sub-component of the computational task of receiver RSU a and the sub-component of the computational task of helper RSU b are both greater than zero, the distance between receiver RSU a and vehicle before offloading the computational task, is checked through (15) to make sure that SNR constraint is followed. If the constraint is followed correctly, Φ_m is set to zero, the vehicle received the result successfully, and the receiver RSU a is also the delivery RSU. However, if the constraint is not followed correctly, Φ_m is set to one, which means that the vehicle is far from the receiver RSU a, and the result could not be received by the vehicle. In this case, the helper RSU b is checked to determine if helper RSU b is the delivery RSU or not. Since the distance which is taken after the processing of a computational task is completed is between receiver RSU a and vehicle. Hence, the subtracted distance is checked through (15) to ensure that the SNR constraint is followed. If the constraint is followed correctly, Φ_m is set to zero, which means that the vehicle was at an acceptable distance from the helper RSU b. The vehicle received the result successfully, and the helper RSU b is also the delivery RSU. However, if the constraint is not followed correctly, Φ_m is set to one, and the vehicle cannot receive the result.
- Third, when the sub-component of the helper RSU b is equal to zero, it implies that the whole computational task will be processed by receiver RSU a. The distance between receiver RSU a and the vehicle before offloading

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

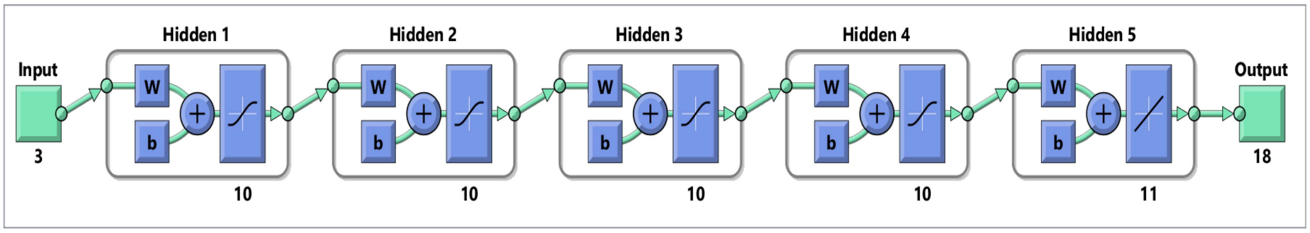


Fig. 2. Diagram of the DNN Model.

the computational task is checked through (15) to ensure that the SNR constraint is followed. If the constraint is followed correctly, Φ_m is set to zero, and the receiver RSU a becomes the delivery RSU. However, if the constraint is not followed correctly, Φ_m is set to one. In this case, the helper RSU b is checked to determine if helper RSU b is the delivery RSU or not. Hence, the subtracted distance is checked through (15) to ensure that the SNR constraint is followed. If the constraint is followed correctly, Φ_m is set to zero, and the helper RSU b becomes the delivery RSU. Otherwise, Φ_m is set to one, which means that the vehicle is far from the helper RSU b, and the result could not be received by the vehicle.

In the next step, we determine the delivery RSU among all the RSUs in the network except for the receiver RSU a and the helper RSU b and store the information on delivery RSU in the dataset. In this step, we also determine the shortest path among RSUs to deliver the result to the vehicle; this path is stored in the dataset. From a distance between receiver RSU and vehicle after the processing of the computational task is completed, we determine the most suitable RSU for delivering the result of each vehicle by keeping the SNR constraint. Suppose the constraint is not met for a vehicle. In that case, the delivery RSU in the dataset is kept null, and the binary variable in the dataset, which denotes the failure in the processing of a computational task, is set to one, which means that the result could not be received by vehicle and the service session has failed. If the constraint is met and the vehicle is at an acceptable distance from a delivery RSU, then the information of that delivery RSU is stored in the dataset. Dijkstra's algorithm is applied to determine the shortest path from the receiver RSU a to the delivery RSU. This shortest path is stored in the dataset. After completing every index of the dataset, the whole generated dataset is given to DNN for training, validation, and testing. Fig. 3 is also shown below to explain the algorithm graphically.

VII. Complexity Analysis

In this paper, we use three algorithms compared to our proposed method. These are the Greedy method, Always-migrate method, and Random method [9]. In this section, n is the size of the dataset. The time complexities of the greedy method, always-migrate method, and random method are $O(nk)$, $O(nk)$, and $O(nk)$, respectively. The time complexity of our proposed method without DNN is $O(C.C_{Z_g}.2^g.r^2)$. Consequently, the time complexity of the proposed method without DNN is worse than those of the other methods. Moreover, the proposed method increases the time delay significantly. However, we will see from the simulation results in Section VIII that the proposed method without DNN is better, and it outperforms the greedy, always-migrate, and random methods. Hence, we must utilize the proposed method, but its time complexity makes it challenging to use. Therefore, instead of using the proposed algorithm without DNN for every RSU, we trained DNN from the dataset obtained from the proposed method. The time complexity of the proposed method with DNN is $O(nlog(n))$ while training. Once the DNN is trained, its complexity becomes constant and $O(1)$. This reduces the service time delay to a minimum, and the time complexity also becomes lower than those of greedy, always-migrate, and random methods. Table V compares asymptotic complexities of the proposed method without DNN, proposed method with DNN, and other methods.

VIII. Simulation Results

For the simulation results, we use three algorithms compared to our proposed method. These are the Greedy method, Always-migrate and random methods [9]. The simulation parameters are shown in Table VI. In the Greedy method, a vehicle sends its task to the RSU with the highest SNR. However, the division of tasks and cooperation of different RSU is not considered. The whole task will be processed at the same RSU and considered to deliver RSU. In the Always-Migrate method, a vehicle sends the task to an RSU, but the whole task is migrated to another RSU the vehicle is approaching. The entire task is processed at the second RSU, which acts as a helper RSU and delivers RSU.

Furthermore, in the Random method, a vehicle randomly sends the task to an RSU, divided into two components. The component is kept at the RSU, and the other component is forwarded to another RSU as the vehicle approaches. Note that there is no routing implementation in each of these methods. Hence, if the vehicle travels out of the communication range of the deliver RSU, the result will not be able to reach the vehicle, and the service session will fail.

The success and failure rates of the entire service session computed from our dataset are compared with the Greedy, Always-migrate and random methods, as summarized in Table VII. It can be observed from the result that our proposed method outperforms the existing schemes. This is because, in our proposed method, the task offloaded by a vehicle is intelligently divided among two RSUs to process in parallel. Hence, the overall service delay is minimum. However, due to our proposed routing algorithm, if the vehicle moves out of the communication

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

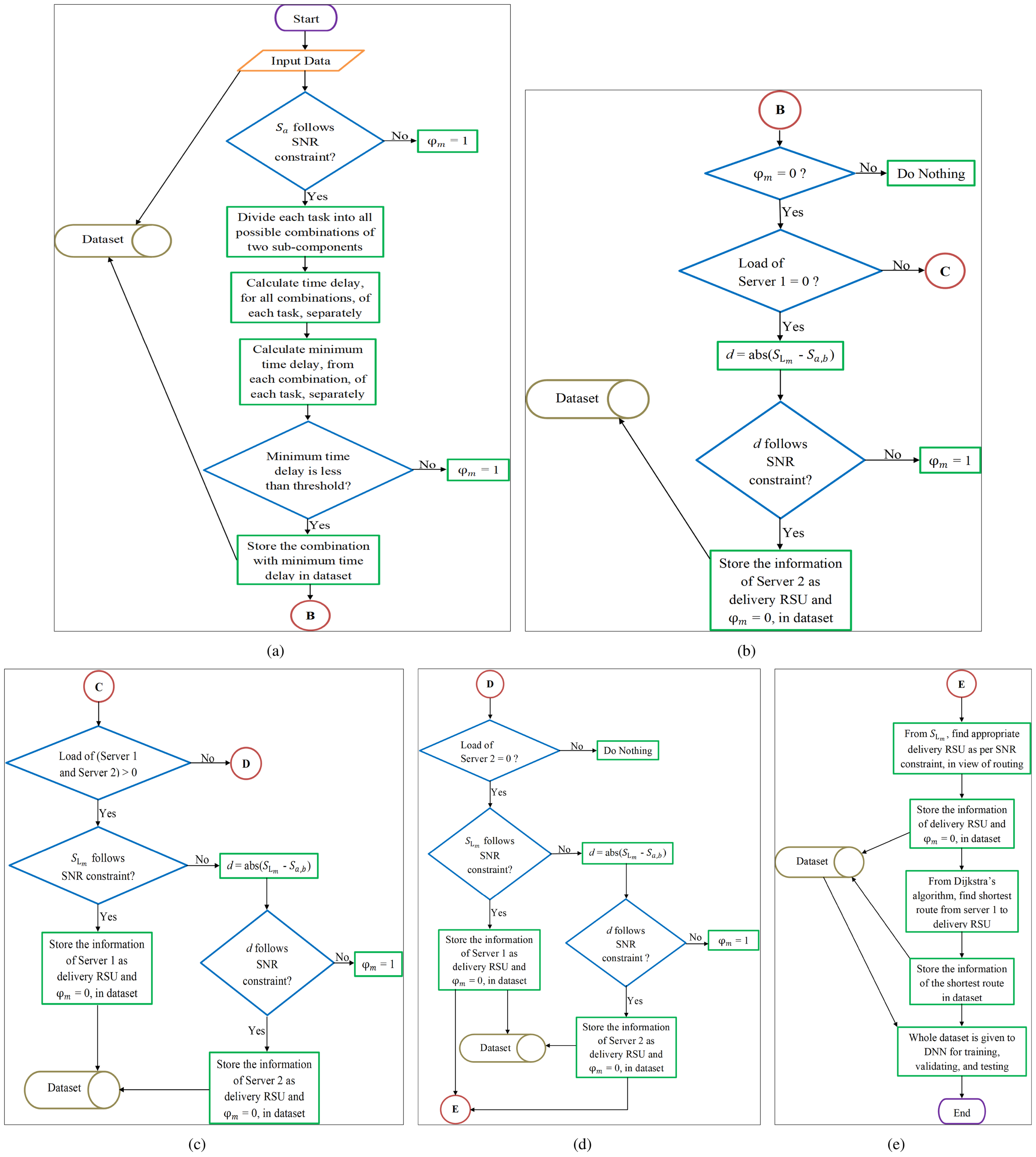

Start
Input Data
S_a follows SNR constraint?
No
$\varphi_m = 1$
Yes
Dataset
Divide each task into all possible combinations of two sub-components
Calculate time delay, for all combinations, of each task, separately
Calculate minimum time delay, from each combination, of each task, separately
Minimum time delay is less than threshold?
No
$\varphi_m = 1$
Yes
Store the combination with minimum time delay in dataset
B
(a)
B
$\varphi_m = 0$?
No
Do Nothing
Yes
Load of Server 1 = 0 ?
No
C
Yes
$d = \text{abs}(S_{L_m} - S_{a,b})$
Dataset
d follows SNR constraint?
No
$\varphi_m = 1$
Yes
Store the information of Server 2 as delivery RSU and $\varphi_m = 0$, in dataset
(b)
C
Load of (Server 1 and Server 2) > 0
No
D
Yes
S_{L_m} follows SNR constraint?
No
$d = \text{abs}(S_{L_m} - S_{a,b})$
Yes
Store the information of Server 1 as delivery RSU and $\varphi_m = 0$, in dataset
d follows SNR constraint?
No
$\varphi_m = 1$
Yes
Dataset
Store the information of Server 2 as delivery RSU and $\varphi_m = 0$, in dataset
(c)
D
Load of Server 2 = 0 ?
No
Do Nothing
Yes
S_{L_m} follows SNR constraint?
No
$d = \text{abs}(S_{L_m} - S_{a,b})$
Yes
Store the information of Server 1 as delivery RSU and $\varphi_m = 0$, in dataset
d follows SNR constraint ?
No
$\varphi_m = 1$
Yes
Dataset
Store the information of Server 2 as delivery RSU and $\varphi_m = 0$, in dataset
E
(d)
E
From S_{L_m}, find appropriate delivery RSU as per SNR constraint, in view of routing
Store the information of delivery RSU and $\varphi_m = 0$, in dataset
Dataset
From Dijkstra's algorithm, find shortest route from server 1 to delivery RSU
Store the information of the shortest route in dataset
Whole dataset is given to DNN for training, validating, and testing
End
(e)

Fig. 3. Graphical explanation of Algorithms 1 - 2 by using flowchart.

range of both RSUs, the result can be delivered back to the vehicle.

Fig. 4 compares the relation between total service delay and total computational task size, which is offloaded by a vehicle and processed by RSU(s). The total service delay includes overall computation delay and result delivery delay. We do not consider the routing algorithm in this experiment. The reason is that there is no routing policy implementation in the existing three approaches. If we consider the routing portion of the proposed method, the other three methods will have some values in their graphs indicating that the service session failed due to the high mobility of the vehicle. Hence, it will be out of the communication range of the two RSUs and therefore making the total service delay infinite. We considered the size of the

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

TABLE V
ASYMPTOTIC COMPLEXITY

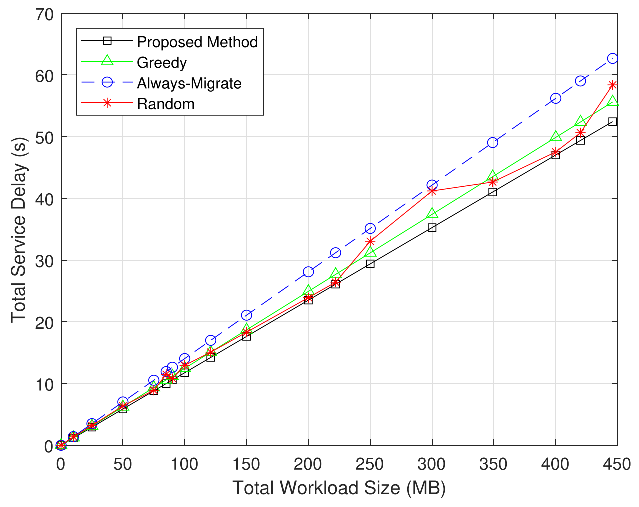
Method	Asymptotic Complexity	Reference
Proposed Method with DNN	$O(1)$	Proposed by this paper
Greedy Method	$O(nk)$	[9]
Always-migrate Method	$O(nk)$	[9]
Random Method	$O(nk)$	[9]
Proposed Method without DNN	$O(C.C_{Z_g}.2^g.r^2)$	Proposed by this paper

TABLE VI
SIMULATION PARAMETERS

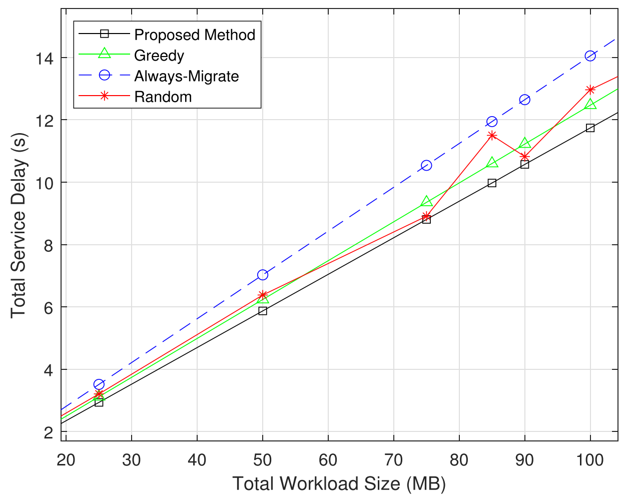
Notation	Description	Value
H	Antenna's Height	6 m
f	Carrier frequency	2400 MHz
T^D	Transmit power of RSU	28 dBm
T^M	Transmit power of vehicle	20 dBm
B^M	Bandwidth reserved for vehicle	1 bps
B^D	Bandwidth reserved for forwarding the task offloaded to RSU	9 bps
δ_D	Power of the Gaussian noise in the channel of RSU-to-RSU	93 dBm
δ_M	Power of the Gaussian noise in the channel of vehicle-to-RSU	93 dBm
ω°	Signal-to-noise ratio (SNR) threshold	8 dBm
ω^D	SNR threshold for result delivering	6 dBm
$S_{(a,b)}$	Distance between RSU a and RSU b	200 m
k	The time length of an epoch	1
N	Number of computation cycles needed to execute one bit	1300 C / bits
F	Computing capability (CPU-cycle frequency) of an RSU	8.2×10^4 C / s
σ	Value used to approximate the size of result concerning the size of computational task	0.015

TABLE VII
SUCCESS RATE AND FAILURE RATE OF THE TOTAL SERVICE SESSION

Algorithms	Success rate	Failure rate
Proposed Method	90.12 %	9.88 %
Greedy Method	60 %	40 %
Always-migrate Method	21.9 %	78.10 %
Random Method	12.66 %	82.34 %



(a)



(b) Zoomed version of Fig. 4(a)

Fig. 4. Total service delay vs total task size offloaded by a vehicle.

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

computational task in MBs as a variable parameter by taking 18 values from (0 – 446 MB). The distance between a vehicle and receiver RSU before offloading the task is kept constant, i.e., 100 m. After the offloaded task is processed, the distance between receiver RSU and the vehicle is also constant at 300 m. The reason for taking these values is to explore the relation between the total service delay and total computational task offloaded by a vehicle to RSU for processing.

The service delay of the random method fluctuates between the always-migrate method and the proposed method. At some points, the delay of the random method gets close to or equal to that of the proposed method. However, it never achieves a lower delay than our proposed method, which means that our proposed method reduces the total service delay to its minimum attainable level. The proposed method performs well by dividing a computational task into several combinations of two subtasks. It then calculates the service delay of each combination, and from all those combinations, it chooses the combination of divided subtasks whose service delay is minimum. This fluctuation in the random method divides the computational task randomly, and sometimes it selects a combination with a low service delay. Still, at times it selects a combination that has high service delay.

Furthermore, the overlapping between graphs of the methods is high at first. Nevertheless, as the size of a computational task increases, the overlap between graphs of the methods decreases, and we can differentiate between them. If we increase the task size further, the overlapping between graphs of the methods is expected to decrease further. Note that we use 2 RSUs for cooperative computing and parallel processing. Other RSUs are used for routing and reducing the failure to deliver results back to the vehicle. If the number of RSUs used in cooperative computing and parallel processing increases, the total service delay will be reduced further. Moreover, the size of the result is estimated to be 1.5% of the size of the computational task. Hence, the value of σ is 0.015.

The relation between total service delay and the number of vehicles that wish to have their tasks processed by the same RSU is shown in Fig. 5. In this comparison, we consider the three variable parameters, i.e., the data size of the computational task, the distance between vehicle and receiver RSU before offloading the task, and the distance between the vehicle and the receiver RSU after the offloaded task is processed. The data size of the computational task in MBs is 500 MB. The distance between a vehicle and receiver RSU before offloading the task is 100 m. After the offloaded task is processed, the distance between receiver RSU and the vehicle is 300 m. Another significant variable is the number of vehicles from 1 to 14 who wish to have their tasks processed by the same RSU.

The values are taken in this fashion because we want our simulation graph to explore the relation between total service delay and the number of vehicles that wish to have their tasks processed by the same RSU. To achieve this, we must keep all other variables constant. All the tasks are combined. Hence, the total data size of the task to be processed is the summation of the tasks from each vehicle. The resources of the RSU are divided

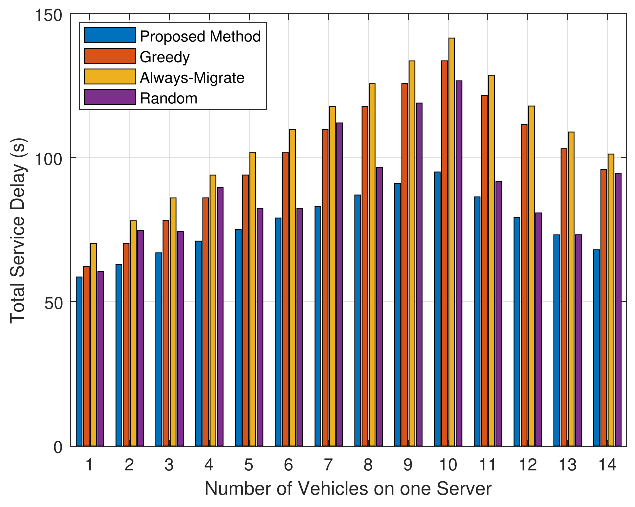


Fig. 5. Total service delay vs number of vehicles who wish to have their tasks processed by the same RSU.

among the vehicles as the traffic on RSU increases. Hence, the parameter F gets divided. When this happens, the total service delay for combined tasks of all the vehicles is escalated. We have placed a threshold for the maximum time delay a computational task can take to be processed. If a computational task takes time to be processed, which exceeds the threshold, then the whole computational task is discarded, even if it is partially processed. The threshold value is considered as 15 minutes. A computational task will usually take much less time to process, but it will be discarded if it takes more than 15 minutes. Note that we have taken this threshold value for our simulations. It is not fixed. It is scalable, and we can change it for our model.

Furthermore, each graph starts to decline after 10 vehicles because a threshold limits the RSU to entertain only 10 vehicles at a time, and the total data size of the tasks is constant. The component tasks of different vehicles are neglected, and the total workload size decreases, decreasing the total service delay. The proposed method reduces the delay considerably. This is because the proposed method divides a computational task into several combinations of two subtasks. It calculates the service delay of each combination, and from all those combinations, it chooses the combination of divided subtasks whose service delay is minimum. Thus, we achieve the minimum total service delay. The service delay of the random method fluctuates, and at some points, it is better than the always-migrate method and random method. However, the proposed method performs better than the existing state-of-the-art.

We use the train-validate-test process for the proposed DNN method. This process helps to detect the overfitting of the network model so that training can be stopped. After training and validating, the testing process takes place. We gradually increased the size of the dataset to see the fluctuation in the accuracy rate of the trained network model caused by it. Table VIII shows the accuracy rate of the proposed DNN method for the testing process concerning increasing dataset size.

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

TABLE VIII
SIZE OF DATASET V.S. ACCURACY RATE OF NETWORK MODEL

Dataset Size	Training	Validation	Testing	Overall
2744	85.4%	84.2%	81.3%	84.6%
3375	86.0%	82.2%	81.0%	84.7%
4096	85.2%	84.9%	83.9%	84.9%

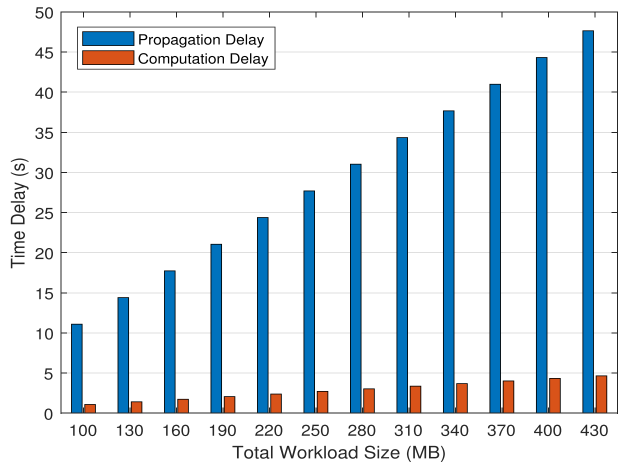


Fig. 6. Comparison of Propagation delay and Computation delay vs total task size offloaded by a vehicle.

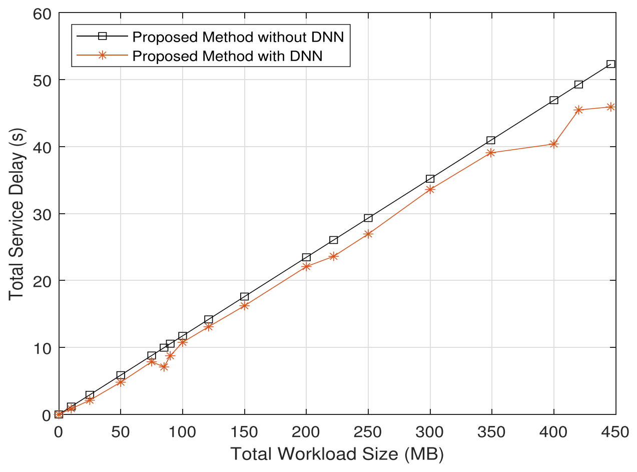


Fig. 7. Comparison of delay of Proposed Method with and without DNN vs total task size offloaded by a vehicle.

Fig. 6 compares propagation time delay and computation time delay concerning increasing total computational task size, which is offloaded by a vehicle and processed by RSU(s). We use 2 RSUs for cooperative computing and parallel processing. Therefore, the computation delay includes the time taken by either both RSUs or one of them to process the computational task. The propagation delay includes the time taken for offloading the computational task from the vehicle to the receiver RSU, forwarding a sub-component or the whole computational task from receiver RSU to helper RSU, and delivering the result back to the vehicle. We considered the size of the computational task in MBs as a variable parameter by taking 12 values from (100–446 MB). The distance between a vehicle and receiver RSU before offloading the task is kept constant, i.e., 100 m. After the offloaded task is processed, the distance between receiver RSU and the vehicle remains constant at 300 m. The reason for taking these values is to explore the relation of propagation delay and computation delay with the increasing computational task size offloaded by a vehicle to receiver RSU for processing. We see that the computation time delay is much less than the propagation time delay because the task is processed in parallel by two RSUs. The result is estimated to be 1.5% of the size of the computational task.

In Fig. 7, we have compared the Proposed Method with DNN and the Proposed Method without DNN. We have calculated the time delay concerning the total task size offloaded by a vehicle. Since once the DNN is trained, its complexity is constant and $O(1)$. Therefore, the trained DNN model's time delay is minimal, as shown in Fig. 7.

IX. CONCLUSION

This paper explored the computation task offloading, task processing and routing problem in vehicular networks. We introduced a smart task division policy to decrease service delay and increase service reliability. In this regard, an efficient and smart routing policy is adopted to ensure the delivery of tasks results to vehicles. We obtained an optimal solution by using an efficient routing algorithm. The algorithm's complexity is high and causes a significant increase in service time delay. Therefore, we proposed a model-based DNN approach and obtained low time complexity to solve the complexity problem. Hence, we obtained optimal solutions from the trained DNN model without calculating it. Therefore, the offloading, computing, division, and routing policies are formulated, and a model-based DNN approach is used to get an optimal solution. Simulation results proved that our proposed approach is suitable in a dynamic environment. We also compared our results with the existing state-of-the-art, showing that our proposed approach outperforms the existing schemes. In our future work, we will utilize reinforcement learning to develop a model that will work even in a scenario where many RSUs are present.

REFERENCES

[1] Y. Lee, H. Choi, Y. Nam, S. Park, and E. Lee, "RSU-driven cloud construction and management mechanism in VANETs," in *Proc. IEEE 8th Int. Conf. Cloud Netw.*, Coimbra, Portugal, 2019, pp. 1–4.
[2] H. Zhang, Z. Wang, and K. Liu, "V2X offloading and resource allocation in SDN-assisted MEC-based vehicular networks," *China Commun.*, vol. 17, no. 5, pp. 266–283, May 2020.
[3] N. Slamnik-Krijestorac, M. Peeters, S. Latre, and J. M. Marquez-Barja, "Analyzing the impact of VIM systems over the MEC management and orchestration in vehicular communications," in *Proc. 29th Int. Conf. Comput. Commun. Netw.*, Honolulu, HI, USA, 2020, pp. 1–6.
[4] C. M. Huang and C. F. Lai, "The delay-constrained and network-situation-aware V2V2I VANET data offloading based on the multi-access edge computing (MEC) architecture," *IEEE Open J. Veh. Technol.*, vol. 1, pp. 331–347, 2020.
[5] J. Ji, K. Zhu, C. Yi, R. Wang, and D. Niyato, "Joint resource allocation and trajectory design for UAV-assisted mobile edge computing systems," in *Proc. IEEE Glob. Commun. Conf.*, Taipei, Taiwan, 2020, pp. 1–6.

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

[6] M. Gao, W. Cui, D. Gao, R. Shen, J. Li, and Y. Zhou, "Deep neural network task partitioning and offloading for mobile edge computing," in *Proc. IEEE Glob. Commun. Conf.*, 2019, pp. 1–6.

[7] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1122–1135, Dec. 2020.

[8] M. Waqas , "A comprehensive survey on mobility-aware D2D communications: Principles, practice and challenges," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1863–1886, Jul.–Sep. 2020.

[9] M. Li, J. Gao, N. Zhang, L. Zhao, and X. Shen, "Collaborative computing in vehicular networks: A deep reinforcement learning approach," in *Proc. IEEE Int. Conf. Commun.*, Dublin, Ireland, 2020, pp. 1–6.

[10] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[11] Q. Qi et al., "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192–4203, May 2019.

[12] X. Wang, Z. Ning, and L. Wang, "Offloading in internet of vehicles: A fog-enabled real-time traffic management system," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4568–4578, Oct. 2018.

[13] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26652–26664, 2019.

[14] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Trans. Cloud Comput.* vol. 7, no. 2, pp. 369–382, Apr. 2019.

[15] Y. Zhao, X. Liu, L. Tu, C. Tian, and C. Qiao, "Dynamic service entity placement for latency sensitive applications in transportation systems," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 460–472, Feb. 2021.

[16] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.

[17] H. Liang, X. Zhang, J. Zhang, Q. Li, S. Zhou, and L. Zhao, "A novel adaptive resource allocation model based on SMDP and reinforcement learning algorithm in vehicular cloud system," *IEEE Trans. Veh. Technol.*, vol. 68, no. 10, pp. 10018–10029, Oct. 2019.

[18] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019.

[19] J. Shen, T. Zhou, J. Lai, P. Li, and S. Moh, "Secure and efficient data sharing in dynamic vehicular networks," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8208–8217, Sep. 2020.

[20] Y. Cui, Y. Liang, and R. Wang, "Intelligent task offloading algorithm for mobile edge computing in vehicular networks," in *Proc. 91st Veh. Technol. Conf.*, Antwerp, Belgium, 2020, pp. 1–5.

[21] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020.

[22] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Veh. Technol. Mag.*, vol. 12, no. 2, pp. 36–44, Jun. 2017.

[23] J. Chen, W. Xu, N. Cheng, H. Wu, S. Zhang, and X. Shen, "Reinforcement learning policy for adaptive edge caching in the heterogeneous vehicular network," in *Proc. IEEE Glob. Commun. Conf.*, 2018, pp. 1–6.

[24] M. Waqas, M. Zeng, and Y. Li, "Mobility-assisted device to device communications for content transmission," in *Proc. 13th Int. Wireless Commun. Mobile Comput. Conf.*, Valencia, Spain, 2017, pp. 206–211.

[25] M. Waqas, M. Zeng, Y. Li, D. Jin, and Z. Han, "Mobility assisted content transmission for device-to-device communication underlaying cellular networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6410–6423, Jul. 2018.

[26] T. Li, A.K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37 no. 3, pp. 50–60, May 2020.

[27] I. Nurcahyani and J. W. Lee, "Role of machine learning in resource allocation strategy over vehicular networks: A survey," *Sensors*, vol. 21, no. 19, 2021, Art. no. 6542.

[28] K. M. A. Alheeti, A. Gruebler, and K. D. McDonald-Maier, "On the detection of grey hole and rushing attacks in self-driving vehicular networks," in *Proc. 7th Comput. Sci. Electron. Eng. Conf.*, 2015, pp. 231–236.

[29] Z. Chen, Q. He, L. Liu, D. Lan, H.-M. Chung, and Z. Mao, "An artificial intelligence perspective on mobile edge computing," in *Proc. IEEE Int. Conf. Smart Internet Things*, 2019, pp. 100–106.

[30] P. Kim, "*MATLAB Deep Learning*," New York City, NY, USA: Apress, 2017.

[31] T. He, H. Mao, and Z. Yi, "Subtraction gates: Another way to learn long-term dependencies in recurrent neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 4, pp. 1740–1751, Apr. 2022.

[32] S. Tu et al., "Reinforcement learning assisted impersonation attack detection in device-to-device communications," *IEEE Trans. Veh. Technol.*, vol. 70, no. 2, pp. 1474–1479, Feb. 2021.

[33] Z. Ali, Z. H. Abbas, G. Abbas, A. Numani, and M. Bilal, "Smart computational offloading for mobile edge computing in next-generation Internet of Things networks," *Comput. Netw.*, vol. 198, 2021, Art. no. 108356.

[34] R. Chai, M. Li, T. Yang, and Q. Chen, "Dynamic priority-based computation scheduling and offloading for interdependent tasks: Leveraging parallel transmission and execution," *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, pp. 10970–10985, Oct. 2021.

[35] C. Chen, Y. Zeng, H. Li, Y. Liu, and S. Wan, "A multi-hop task offloading decision model in MEC-enabled internet of vehicles," *IEEE Internet Things J.*, early access, Jan. 18, 2022, doi: 10.1109/JIOT.2022.3143529.

Suleman Munawar received the B.S. degree in computer engineering from the Faculty of Computer Science and Engineering, Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, in 2021. In 2021, he joined ProntoDigital LLC, Fort Wayne, IN, USA, as a Data Analytics Consultant. His research interests interests include the areas of vehicular networks, mobile edge computing, the Internet of Things, deep learning, and reinforcement learning.

Zaiwar Ali received the B.S. degree in electronics engineering from the COMSATS Institute of Information Technology Abbottabad, Pakistan, in 2012, the M.S. degree in electronics engineering from the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology (GIK Institute), Topi, Pakistan, in 2015, and the Ph.D. degree in electronics engineering from the Telecommunications and Networking Research Lab, GIK Institute, in 2021. He is currently with the Faculty of Electrical Engineering, GIK Institute, as a Assistant Professor. From 2016 to 2018, he was a Lecturer with the Faculty of Electrical Engineering, GIK Institute. From 2013 to 2015, he was a Graduate Assistant with GIK Institute and was the recipient of the highest level of merit Scholarship. His research interests include multi-access edge computing, cloud computing, stochastic processes, IoT, machine learning, and wireless sensor networks.

Muhammad Waqas (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees from the Department of Electrical Engineering, University of Engineering and Technology, Peshawar, Pakistan, in 2009 and 2014, respectively, and the Ph.D. degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, during September 2015–June 2019. From 2012 to 2015, he was with the Sarhad University of Science and Information Technology, Peshawar, Pakistan, as a Lecturer and Program Coordinator. From August 2019 to March 2022, he was with the Faculty of Computer Science and Engineering, GIK Insitute of Engineering Sciences and Technology, Topi, Pakistan, as an Assistant Professor. From October 2019 to September 2022, he was also associated with the Faculty of Information Technology, Beijing University of Technology, Beijing, China, as a Visiting Research Associate. He is currently an Assistant Professor with the Computer Engineering Department, College of Information Technology, University of Bahrain, Zallaq, Bahrain. He is also an Adjunct Senior Lecturer with the School of Engineering, Edith Cowan University, Joondalup, WA, Australia. His research interests include the areas of physical layer security, vehicular networks, mobile edge computing and the Internet of Things. He has several research publications in reputed Journals and Conferences. He is the Co-Chair, TPC Member and Reviewer of several international conferences and journals. He is also an Associate Editor for the International Journal of Computing and Digital Systems. He was the recipient of the Best Paper Award at ASSP in 2021.

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl

Shanshan Tu (Member, IEEE) received the Ph.D. degree from Computer Science Department, Beijing University of Posts and Telecommunications, in 2014. From 2013 to 2014, he visited the University of Essex, Colchester, U.K., for National Joint Doctoral Training. From 2014 to 2016, he was with the Department of Electronic Engineering, Tsinghua University, Beijing, China, as a Postdoctoral Researcher. He is currently an Associate Professor and Deputy Dean with Faculty of Information Technology, Beijing University of Technology, Beijing, China. His research interests include the areas of cloud computing, MEC, and information security techniques.

Syed Ali Hassan (Senior Member, IEEE) received the M.S. degree in mathematics and the Ph.D. degree in electrical engineering from Georgia Institute of Technology, Atlanta, GA, USA, and the M.S. degree in electrical engineering from the University of Stuttgart, Stuttgart, Germany. He was a Research Associate with Cisco Systems, Inc., San Jose, CA, USA. He is currently a Professor with the School of Electrical Engineering and Computer Science, NUST, Islamabad, Pakistan, where he is also the Director of the Information Processing and Transmission Research Group, which focuses on various aspects of theoretical communications. He has (co)authored more than 250 publications in international conferences and journals and has organized several special issues/sessions as editor/chair in leading journals/conferences. His research interest focuses on the signal processing for communications.

Ghulam Abbas (Senior Member, IEEE) received the B.S. degree in computer science from the University of Peshawar, Peshawar, Pakistan, in 2003, and the M.S. degree in distributed systems and the Ph.D. degree in computer networks from the University of Liverpool, Liverpool, U.K., in 2005 and 2010, respectively. From 2006 to 2010, he was Research Associate with Liverpool Hope University, Liverpool, U.K., where he was associated with the Intelligent and Distributed Systems Laboratory. Since 2011, he has been with the Faculty of Computer Sciences and Engineering, Ghulam Ishaq Khan (GIK) Institute of Engineering Sciences and Technology, Topi, Pakistan. He is currently with Huawei Network Academy, as an Associate Professor and the Director. He is a Co-Founding Member of the Telecommunications and Networking Research Laboratory, GIK Institute of Engineering Sciences and Technology. His research interests include computer networks and wireless and mobile communications. He is a Fellow of The Institute of Science and Technology, U.K., and the British Computer Society.

thorized licensed use limited to: NUST School of Electrical Engineering and Computer Science (SEECS). Downloaded on August 10,2025 at 01:03:28 UTC from IEEE Xplore. Restrictions appl