mathematics

MDPI

Article

Task-Offloading Strategy Based on Performance Prediction in Vehicular Edge Computing

Feng Zeng [1,*], Jiangjunzhe Tang [1], Chengsheng Liu [1], Xiaoheng Deng [1,*] and Wenjia Li [2,*]

[1] School of Computer Science and Engineering, Central South University, Changsha 410083, China; 204712270@csu.edu.cn (J.T.); liucs@csu.edu.cn (C.L.)
[2] Department of Computer Science, New York Institute of Technology, New York, NY 10023, USA
* Correspondence: fengzeng@csu.edu.cn (F.Z.); dxh@csu.edu.cn (X.D.); wli20@nyit.edu (W.L.)

Abstract: In vehicular edge computing, network performance and computing resources dynamically change, and vehicles should find the optimal strategy for offloading their tasks to servers to achieve a rapid computing service. In this paper, we address the multi-layered vehicle edge-computing framework, where each vehicle can choose one of three strategies for task offloading. For the best offloading performance, we propose a prediction-based task-offloading scheme for the vehicles, in which a deep-learning model is designed to predict the task-offloading result (success/failure) and service delay, and then the predicted strategy with successful task offloading and minimum service delay is chosen as the final offloading strategy. In the proposed model, an automatic feature-generation model based on CNN is proposed to capture the intersection of features to generate new features, avoiding the performance instability caused by manually designed features. The simulation results demonstrate that each part of the proposed model has an important impact on the prediction accuracy, and the proposed scheme has the higher Area Under Curve (AUC) than other methods. Compared with SVM- and MLP-based methods, the proposed scheme has the average failure rate decreased by 21.2% and 6.3%, respectively. It can be seen that our prediction-based scheme can effectively deal with dynamic changes in network performance and computing resources.

Keywords: vehicular edge computing; task offloading; performance prediction; deep learning; service delay

MSC: 68T07

Citation: Zeng, F.; Tang, J.; Liu, C.; Deng, X.; Li, W. Task-Offloading Strategy Based on Performance Prediction in Vehicular Edge Computing. *Mathematics* **2022**, *10*, 1010. https://doi.org/10.3390/math10071010

Academic Editors: Xinchao Zhao, Xingquan Zuo, Yinan Guo and Kunpeng Kang

Received: 22 February 2022
Accepted: 18 March 2022
Published: 22 March 2022

1. Introduction

With the rapid development of communication technology, vehicles have begun to be equipped with many sensors and on-board computing units, and have gradually evolved from ordinary traffic tools to intelligent devices, which can achieve automatic driving, intelligent navigation and other advanced functions [1,2]. However, many on-board sensors may generate a massive amount of data in a short time, and limited vehicular computing resources cannot fully process these data in time [3]. Although remote cloud servers can provide computing services to requesting vehicles, they are far away from vehicles, meaning that network transmission and processing of a large amount of data would result in the high service latency that cannot be tolerated by delay-sensitive vehicular applications [4]. Facing the above challenge of cloud computing [5], some researchers have introduced multi-access edge computing (MEC) [6] to the Internet of vehicles, and formed a new network paradigm called vehicular edge computing (VEC) [7] that can effectively improve the data communication rate and computation capability of vehicles.

In the framework of VEC, vehicles can offload some computation-intensive tasks to VEC servers, so that they can obtain high-quality and rapid computing services. Generally speaking, the VEC framework usually consists of three layers [8]. The first is the data layer, including vehicles with massive data generated in real time. The second is the edge layer,

including some devices such as road-side units (RSUs), edge servers, cellular base stations, and so on. Since the distance between the edge network and the vehicles is relatively close, the edge layer can provide effective computation service for vehicles. The third is the cloud layer, which is usually able to provide strong computing power for remote users, but the service delay may be a little higher than that of the edge.

With the support of VEC, the performance of computation-intensive vehicular applications can be greatly improved [9]. However, the resources of edge servers are limited and expensive, and with increasing demand for VEC from vehicles, competition for computing resources is becoming increasingly tough. Consequently, we should find the optimal solution to schedule computation resources to satisfy the computation offloading requests from the huge number of vehicles, which becomes the key issue in VEC. Moreover, with consideration to the time-varying network status and computing resources, the effective prediction of task-offloading performance can bring great benefits to service requesters. It is conceivable that if the task-offloading performance can be effectively predicted, the requesting vehicles will choose the most effective offloading strategies from the outset.

With the development of VEC technology, more and more new intelligent applications have become popular, generating more and more data. On the other hand, deep learning is an important method to mine effective information from massive data. Compared with other methods, deep-learning methods can obtain more accurate prediction results. Consequently, massive amounts of data from vehicular applications can be processed by deep-learning-related techniques to achieve what we want. In this paper, we propose a deep-learning-based method to predict the task-offloading performance, including the offloading result (success/failure) and the service delay, then find the best offloading strategy according to the prediction results. The main contributions in this paper are as follows.

(1) We propose a task-offloading scheme based on deep-learning-based prediction, in which a binary classifier is responsible for predicting the offloading result (success/failure) and a regressor is responsible for service-delay prediction. The model is trained with history offloading data to predict the performance of task offloading, and to make task-offloading decisions based on the predicted results.
(2) We propose an automatic feature-generation model based on convolutional neural networks (CNN). Our approach uses the convolutional layer to capture intersections between raw features to generate new features, avoid the unstable performance caused by manually designed features, and improve the practicability and robustness of prediction models.

The remainder of this paper is organized as follows. In Section 2, we introduce related work. The system models and main problem are discussed in Section 3. Section 4 presents the performance-prediction-based task-offloading scheme and proposes the deep-learning models for performance prediction and new feature generation. Then, we train the models, carry out the simulation, and analyze the simulation results in Section 5. Finally, a conclusion is drawn in Section 6.

2. Related Work

In the VEC system, vehicles can offload all or part of their computing tasks to an edge server via wireless communication. However, the resources of edge servers are limited [10], and the resources should be effectively allocated to meet the requirements of the requesting vehicle as much as possible. To this end, scholars have conducted many studies [4,10–16] on the optimal strategy of computation offloading in VEC, including when to offload, how to offload, and where to offload the computation. In these works, advanced methods are used to find effective solutions, such as game theory [4,10,11], graph theory [12], reinforcement learning [13], blockchain [14–16] for security issues, and so on.

To reduce the impact of the mobility of vehicles and the complexity of dynamic networks on task offloading, Liu et al. [17] proposed a task-offloading method based on matching algorithms to minimize service delay. In their solution, a macro base station

(MBS) was set in the center of the road to match vehicles and RSUs, and the vehicles could send matching requests to the MBS for offloading decisions. However, the requests may generate extra network overheads, and the corresponding process would slow the speed of offloading decision making. In their model, the network status was supposed to be fixed, which is inconsistent with a real network environment.

Considering the limit of edge resources, Feng et al. [18] introduced a hybrid vehicular edge–cloud architecture, in which vehicles can not only offload tasks to edge servers, but also to neighboring vehicles and cloud servers. On this basis, the authors proposed a real-time task-offloading algorithm, which could increase the success rate of task offloading and make full use of transmission ability in a cellular network. When performing task offloading, the appropriate neighboring vehicle or RSU was selected as the offloading target by estimating service delay and execution time. If the neighboring vehicle and RSU cannot meet the service quality requirements, the task would be scheduled and offloaded to the remote server in the cloud center via the cellular network. Since this was a real-time algorithm, individual devices need to synchronize each other's state to work together. If the status information of the devices cannot be synchronized in real time, it would cause task offloading to fail.

Sonmez et al. [19] designed a machine-learning-based task-scheduling mechanism with two stages. In the first stage, the authors proposed a classification model based on multi-layer perceptron (MLP) for the prediction of the task-offloading result. With the success or failure of task offloading well predicted in the first stage, in the second stage, the offloading delay was predicted via a support vector machine (SVM). According to the previous prediction results, the task would be scheduled to the node that had the least processing delay to execute the task. The performance of this offloading mechanism depended on the performance of the classification and regression models used in both phases, and the performance of these two machine-learning models depended on the features selected, and bad features directly decreased the performance of the proposed mechanism. Consequently, the proposed mechanism may be sometimes unstable.

Yang et al. [20,21] proposed a multi-access edge-computing (MEC) task-allocation framework based on hierarchical machine learning. In contrast to previous studies, this framework not only used deep neural networks (DNN) on the MEC servers, but also deployed shallow neural networks (SNN) on the mobile devices. Once a computation task was going to offload, the mobile device first used the SNN model to make an offloading decision, and then uploaded the required data to the MEC servers for a more precise decision. The advantage of this architecture was that when there was a large amount of data, the computing load of the MEC server would be shared, significantly reducing data-processing time. However, good cooperation between the server and mobile device is needed, which may add complexity to the edge-computing framework.

Wu et al. [22] proposed a distributed task-offloading algorithm for the Internet of things based on DNN networks. The authors addressed a hybrid offloading model, in which the heterogeneity of edge and cloud servers was taken into account in the offloading destination selection. Meanwhile, they paid attention to the collaboration of cloud computing and edge computing, and considered the transmission delay of mobile services and the energy consumption of equipment.

Wang et al. [23] established a fog–cloud offloading system, and the whole system had two parts: the front end, including vehicles and RSUs, and the back end, including fog nodes and cloud servers. For the front end, a predictive combination method was proposed to reduce delay and consumption. For the back end, a deep-learning model was proposed to predict the consumption of each computing device and the computing node with the least consumption was selected in order to minimize the cost.

Wu et al. [24] proposed an SVM-based offloading algorithm to improve the offloading performance in vehicular edge computing. The algorithm segments a large task into subtasks according to network conditions, and uses an SVM to determine whether each subtask is executed locally or offloaded to the edge. In contrast to Wu [24], Zeng et al. [25]

proposed an effective task-scheduling mechanism, in which deep-learning methods were applied to predict the task-offloading results and the service delay, then the offloading strategy with the minimum delay was selected as the final decision. However, automatic feature generation is not considered, and the manual design of features cannot reflect the changing network performance and computing resource.

In summary, although the existing works have achieved some performance improvements for task offloading in VEC, there are still the following two points that should be given attention. First, task scheduling should be adaptive to the real-time changing network environment. Second, the machine-learning-based methods rely on the design of features, and different feature designs have great differences in performance. Considering the above two points, in this paper, we propose a deep-learning-based task-offloading method that can make optimal decisions regarding task offloading according to the current network performance and the workload of edge servers. With the help of the powerful expression ability of deep neural networks, the associations between task information and offloading performance in history data are mined to improve the efficiency of task offloading, and to reduce the uncertainty brought by manual feature design, we propose the automatic feature-generation model based on CNN to improve the practicability and robustness of the task-offloading decision method.

3. System Model

In this paper, we consider a typical three-tier architecture in VEC, as shown in Figure 1. The system consists of the vehicle layer, the edge layer, and the cloud layer. The vehicles can communicate with road-side units (RSUs) deployed at the roadside via wireless channels. Each RSU is wired connected to an edge server, and the RSUs are connected to the cloud center via optical cables. The vehicles can offload tasks to the edge server and to the cloud server with the use of vehicle-to-infrastructure (V2I) wireless communication via the RSUs. As another option, vehicular tasks can also be offloaded to the cloud server using cellular network communication between the vehicles and the cloud center. Therefore, there are three ways for vehicle users to offload tasks: first, to the edge server via RSUs; second, to the cloud server via RSUs; and third, to the cloud server via a cellular base station.

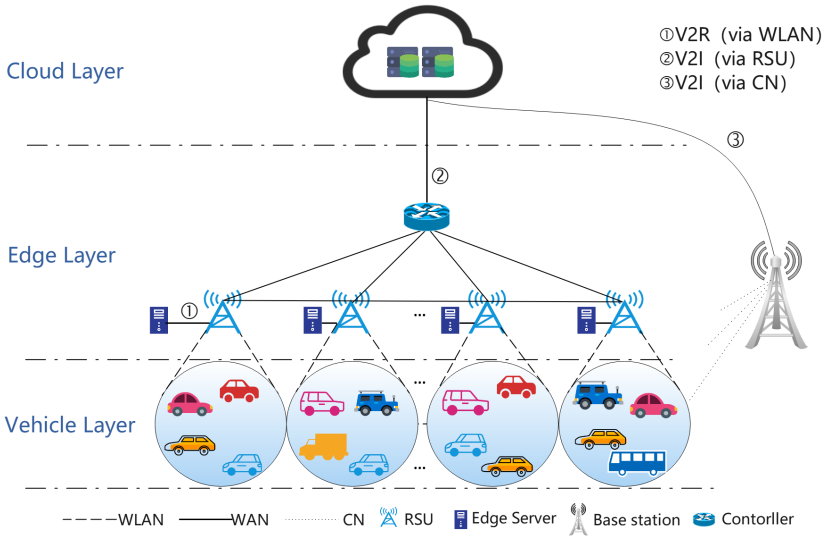

Figure 1. Multi-layered vehicular edge-computing system.

Each of the three offloading methods has its own advantage. From the perspective of wireless communication, short-range wireless communication technology is used between vehicles and RSUs, which has higher transmission bandwidth and lower data transmission latency compared to cellular communication. From the perspective of data-processing capability, the cloud center has stronger processing capability and lower task-execution

time compared to edge servers. However, edge servers are closer to vehicles, and the delay of data uploading and results returning can be much lower than that of the other two methods.

It is assumed that vehicles have n tasks that need to be offloaded, and these n tasks are represented as t_1, t_2, ..., t_n. As mentioned above, tasks can be offloaded in three ways. The service delay of task offloading is not only related to the offloading method, but also related to the current network status (such as network bandwidth, delay, throughput, load of edge servers, and so on) and task size. We suppose that if network status at a certain time is s, the service delays for task offloading via the three methods can be expressed as $D_s^1(t_i)$, $D_s^2(t_i)$, and $D_s^3(t_i)$, respectively, the real offloading delay of task t_i is represented as $D_s(t_i)$, and its maximum service delay for successful offloading is marked as T_i.

When $D_s(t_i) < T_i$, it is considered that the task can be successfully offloaded, otherwise, the offloading of task t_i has failed. Generally speaking, when the network and servers are only lightly loaded, we usually have $D_s^1(t_i) < D_s^2(t_i) < D_s^3(t_i)$, since the edge servers are close to the vehicles and the short-range wireless communication has more bandwidth than cellular network communication. However, when the task size is large, wireless communication congestion occurs, and the insufficient processing capacity of the edge servers increases the queuing delay. All these factors may cause the failure of task offloading. At this time, offloading the task to the cloud server may successfully offload the task and save the time of data transmission and processing. In this paper, we will find the offloading method with the minimum service delay as the optimal offloading strategy. We assume the binary integer variables $x_i \in \{0,1\}$, $y_i \in \{0,1\}$, and $z_i \in \{0,1\}$ are flags of the three offloading methods for task i, respectively. When the value of the flag variable is 1, it means that the corresponding task offloading method is adopted. For example, if x_i is 1, then the task i is offloaded by the first method (offloading to the edge server via RSUs). Similarly, $y_i = 1$ means that the task is offloaded with the second method, and $z_i = 1$ means that the task is offloaded with the third method. Formally, the problem can be modeled as follows.

$$\max \frac{1}{n}\sum_{i=1}^{n} f(D_s(t_i), T_i) \tag{1}$$

$$s.t.\ \ C1: \forall\, i \in [1,n],\ x_i + y_i + z_i = 1 \tag{2}$$

$$\text{C2: } D_s(t_i) = \ x_i D_s^1(t_i) + y_i D_s^2(t_i) + z_i D_s^3(t_i) \tag{3}$$

$$\text{C3: } f(D_s(t_i), T_i) = \begin{cases} 1 & D_s(t_i) \le T_i \\ 0 & D_s(t_i) > T_i \end{cases} \tag{4}$$

In the above problem description, condition C1 indicates that the vehicles can choose only one of the three ways to offload a task, C2 defines the service delay of the task, and C3 is the success function given the successful offloading, as 1 and the failure as 0. The problem is an integer linear programming problem that has high complexity, and due to the solving complexity, approximative solutions may be more effective in a real VEC system than deterministic ones.

We want to maximize the success rate of task offloading. From the opposite perspective, a failed offloading may result in a waste of network and computing resources, and we should avoid the failure of task offloading as much as possible. Consequently, if the success rate of task offloading in a certain offloading method can be predicted before offloading, it will be effective for the vehicles to choose the right method of task offloading, therefore improving the performance of VEC system. Then, the key to problem solving lies in the accuracy of the offloading delay estimation. In this paper, based on historical data generated by EdgeCloudSim [26], we apply deep learning to predict the success rate and service delay of task offloading.

4. Task-Offloading Strategy Based on Performance Prediction

We propose a task-offloading strategy with deep-learning-based prediction of offloading performance, and the service delay is the key performance metric that determines the success or failure of task-offloading. Referring to the generation of user-click features in [27], we make the offloading features automatically generated to avoid the instability of prediction accuracy caused by manual feature combination. Based on the prediction of offloading success rate and service delay, the offloading method with high success rate and minimum service delay will be selected as the optimal strategy.

4.1. Prediction Model with Automatic Feature Generation

The proposed model is shown in Figure 2, which is mainly composed of two modules, namely the new feature-generation module and the performance prediction module. Similar to user-click feature generation in [27] which aims to predict the probability of a user to click on a given item, the generation of task-offloading features is based on the CNN network and can achieve feature cross-generation by convolution, pooling, recombination, and other operations. The prediction module uses DNN to learn raw features and newly generated features, and predicts the result (success or failure) of task offloading and the service delay. In addition to these two main modules, there is also a feature-preprocessing part. Each part of this model is described in detail as follows.

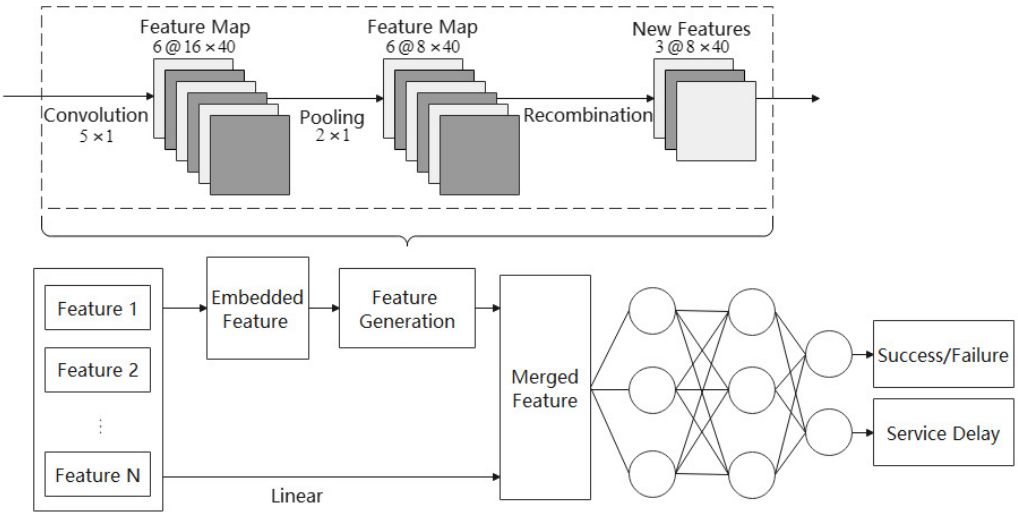

Figure 2. Prediction model with automatic feature generation.

4.1.1. Feature-Generation Module

CNN can learn spatial-related local patterns of samples. Taking this advantage, we propose a CNN-based model for automatic feature generation. As the input, the features need to be preprocessed, including the discretization and embedding.

Discretization is also called a binning operation. First, the features are sorted according to some rules, then each feature is divided according to the appropriate segmentation point, and finally the segmentation result is evaluated. Evaluation indicators include entropy, independence, and accuracy. If the result of segmentation cannot meet the requirements, a new segmentation point is selected for segmentation until the segmentation result meets the requirements.

The embedding operation can convert the discretized features and the classification features in the raw features into vectors with suitable dimensionality, so that the CNN can perform convolution operations. In the model, each input sample changes from 14×1 to $14 \times n$, where n is the given dimensionality. For features with high dimensionality after discretization, directly converting them to one-hot codes, which use binary numbers to represent different variables, will cause the vector to be too sparse, which is not conducive to the training of neural networks. In this situation, it is necessary to add an embedding

layer to compress it and reduce dimensionality. By contrast, if a feature has relatively low dimensionality, it should have its dimensionality increased to ensure that the neural network can handle it.

In this paper, neural networks are designed to automatically generate features that will be combined with raw features to improve the model's ability to express the task-offloading problem. In DNN, lots of parameters are generated in feature interaction, which will slow the calculation speed. Compared with DNN, CNN can reduce the number of parameters and better identify local feature interactions, but it may lose important global information, resulting in poor performance. To take advantage of both DNN and CNN, we combine DNN and CNN, and propose a CNN-based network to generate new features. The CNN model is used to automatically generate new features that can be combined with raw features to reduce the optimization pressure of the DNN model. As shown in the upper part of Figure 2, this is mainly composed of three components, i.e., convolution, pooling and recombination. The convolution layer captures local information and makes the features cross. The recombination layer generates new features while reducing the number of features. The new features generated from the raw features expand the input feature space and strengthens the model's ability for feature expression. Each component in the structure can be described as follows.

(1) Convolution layer

After feature preprocessing, each input sample is transformed from a 14×1 matrix to a 16×40 matrix, where 40 is the embedding dimension specified during the embedding operation, and then two more rows from 14 to 16 are added to the raw features. As shown in Figure 2, the 16×40 matrix is inputted to the first convolutional layer, and the output is the feature map of $(16, 40, 6)$.

(2) Pooling layer

The pooling layer compresses the output of the convolutional layer. The proposed model adopts max-pooling in the pooling layer, which retains the maximum value in the observation window. Generally speaking, the maximum value can often provide more information than the average value. Mean-pooling usually causes the information of the feature map to be diluted.

This model adopts an $h \times 1$ pooling window to deal with the output of the convolution. For the convenience of feature combination, the width is set to 1 to ensure that the feature map cannot be subsampled in the width dimension. It is supposed that the output of the first convolution is C^1, and then the output of the first pooling layer is P^1 shown in (5):

$$P^1_{x,y,i} = \max\left(C^1_{x*h,y,i}, \ldots, C^1_{x*2h-1,y,i}\right), \tag{5}$$

where x and y are the row and column indexes of the feature map. It can be seen from (5) that the height of the output of pooling is the same as the number of features divided by the height of the pooling window. The output of the first pooling layer is used as the input of the second convolutional layer, and so on:

$$E^{i+1} = P^i, \tag{6}$$

where P^i represents the output of the i pooling layer, and E^{i+1} represents the input of the $i+1$ convolutional layer.

(3) Recombination layer

The recombination is scheduled after the convolution and the pooling. The feature maps passing through the convolutional layer and the pooling layer already have local feature intersections. However, if the feature maps as the outputs of the pooling layer are directly sent to the densely connected layer, it will cause a loss of global features and low information density. To this end, the recombination layer uses a fully connected neural

network to recombine the output of the pooling layer, and the output features of the recombination layer are new features generated after feature crossover.

With the use of the convolutional layer, pooling layer, recombination layer and other network structures, the model can generate new cross features, as shown in Figure 3. In the figure, f1, f2, f3, and f4 are the raw features after discretization and embedding. It can be seen that the 4×3 input matrix is convolved by the 2×1 convolution kernel, and the output is the 3×3 response feature map. The feature f1 in the input matrix has disappeared from the output response feature matrix. The convolutional layer outputs a 3×3 response feature matrix, and the pooling layer outputs a 2×3 response feature matrix. After the convolution and pooling, only features f2 and f4 are retained. Finally, the interaction of f2 and f4 is obtained through a reorganization operation.

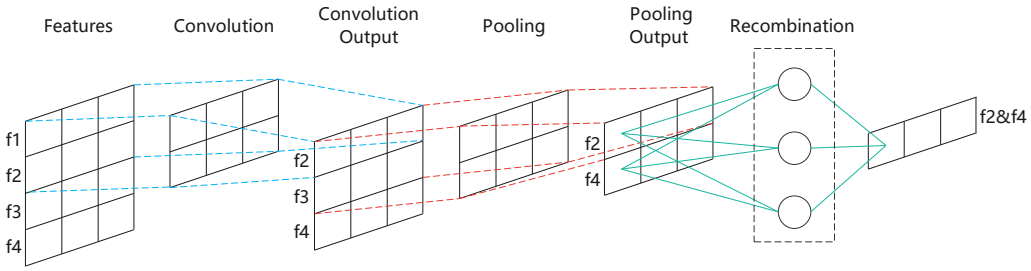

Figure 3. CNN-based feature generation.

4.1.2. Task-Offloading Result Prediction Module

The new features and raw features need to be merged and put into the task-offloading prediction model. It is assumed that new features are $R = (R_1, R_2, ..., R_i)$, in which i is the number of times for convolution, pooling and recombination. Then, the new features can be combined with the raw features according to (7):

$$E = (E^{'T}, R^T)^T, \tag{7}$$

where E' is the raw features, and E is the new feature matrix including raw features and the new features generated by the feature-generation module. In our proposed model, E is the input, and the offloading result and service delay are the outputs. Moreover, there are some hidden layers in the prediction model, and the first hidden layer input of the fully connected network can be denoted as I_1:

$$I_1 = Flatten(E), \tag{8}$$

where $Flatten$ is to flatten the original two-dimensional feature matrix E into a vector, since the two-dimensional matrix cannot be the input into the fully connected network. Given I_i is the input of the i hidden layer, the output of the i hidden layer can be expressed as O_i:

$$O_i = Relu\left(I_i W^i + B^i\right), \tag{9}$$

where W^i is weight matrix of the i hidden layer, B^i is the bias value of the i hidden layer, and $Relu$ is the activation function. By adding a nonlinear activation function $Relu$ to the hidden layer, nonlinear transformation can be achieved. In addition, superimposing multiple nonlinear change layers can obtain a rich hypothesis space, therefore improving the expression ability of the neural network.

The task-offloading prediction model uses a multi-input network structure. The network has two processing units. One is the task-offloading result classifier, and the other is the service-delay regressor. Predicting the offloading result of a task is a binary classification task, so that we choose binary cross-entropy as the loss function of the classifier, and predicting the service delay of the task offloading is a regression task. Thus, we take the mean square error as the loss function of the regressor.

The prediction result of the classifier is denoted as y_1:

$$y_1 = sigmoid\left(I_{nh}W^{nh+1} + B^{nh+1}\right), \tag{10}$$

where I_{nh} is the activation output of the last hidden layer. Since the task-offloading result prediction is a two-class classification task, *sigmoid* is used as the activation function of the classifier, which has a success rate of task offloading as the output.

The output of the regressor is denoted y_2:

$$y_2 = I_{nh}W^{nh+1} + B^{nh+1}, \tag{11}$$

where W^{nh+1} is the weight value of the regressor, and B^{nh+1} is the bias value. The service-delay prediction is a regression task, and the regression task generally does not require an activation function, so that the regressor directly takes the service delay as its output.

4.2. Task-Offloading Strategy Based on Performance Prediction

Based on the above prediction method, we can use historical data to train the models that are, respectively, suitable for the three offloading methods, and then three effective prediction models can be obtained. When a task is requested for offloading, according to the characteristics of the task and network environment, three well-trained models are used to predict the offloading results and service delay for decision making, as shown in Figure 4. The three trained models are labeled as V2R (via WLAN), V2R (via RSU) and V2R (via CN) in Figure 4, and each model outputs two values. One is a binary value to represent the success or failure of task offloading, and the other is the estimated service delay for task offloading.

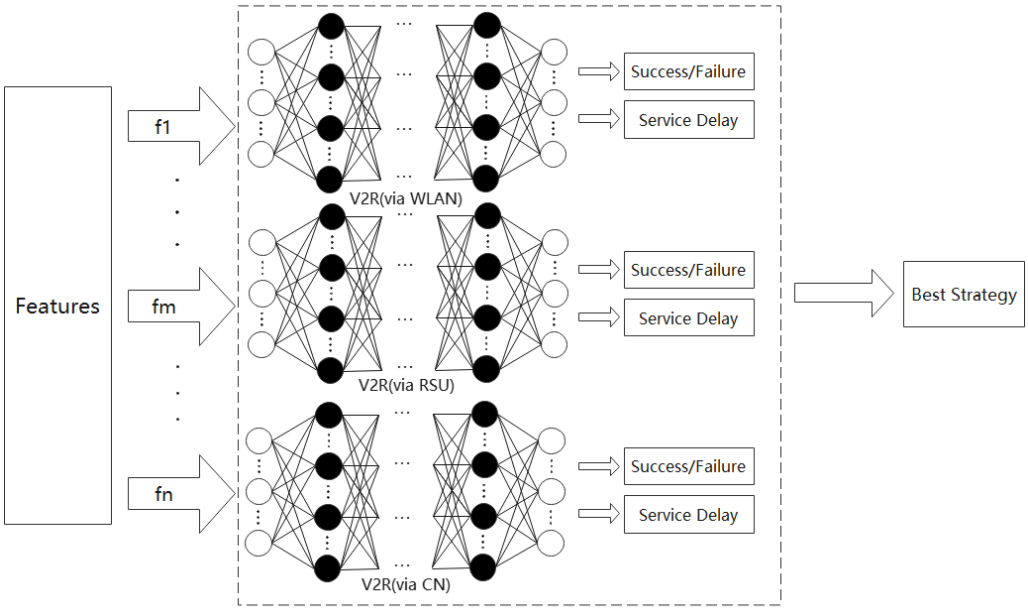

Figure 4. Task-offloading strategy based on performance prediction.

It is assumed that the offloading of task n is predicted by the above method, the results of three offloading ways for task offloading are c_n^1, c_n^2 and c_n^3, and the predicted service delays are s_n^1, s_n^2 and s_n^3, respectively. Then, the offloading method with the most successful result and the minimum service delay is the final decision. Let i be the optimal method to offload the task. Then, we have:

$$i = \min_{1 \le i \le 3} c_n^i s_n^i, \tag{12}$$

where $i \in [1,3]$ and $c_n^i \in \{0,1\}$.

The decision-making process is shown in Algorithm 1. From lines 4 to 6, the task features are put into the prediction model to achieve the predicted offloading results and service delays. Then, the predicted results using the three offloading methods will be compared to select the best one, shown in lines 7 to 14. In line 8, it is judged whether the task is offloaded successfully. If c^i is equal to 0, the task offloading fails, and the unsuccessful results are not considered.

Algorithm 1 Task-offloading strategy based on performance prediction

Require: $X = 14 \times 1$: 14 selected features of a task; $f(x)$: the proposed model; $f_{WLAN}(x)$: vehicles communicate with RSUs via WLAN; $f_{RSU}(x)$: vehicle-to-infrastructure via RSUs; $f_{CN}(x)$: vehicle-to-infrastructure via CN;
Ensure: i: selected the i-th offloading strategy;

```
1:  function BestStrategy(X, f_WLAN(x), f_RSU(x), f_CN(x))
2:      result ← 0
3:      min ← 0
4:      c^1, s^1 ← f_WLAN(X)
5:      c^2, s^2 ← f_RSU(X)
6:      c^3, s^3 ← f_CN(X)
7:      for each i ∈ [1,3] do
8:          if c^i ≠ 0 then
9:              if min == 0 or min > c^i × s^i then
10:                 result ← i
11:                 min ← c^i × s^i
12:             end if
13:         end if
14:     end for
15:     return result
16: end function
```

5. Performance Evaluation

5.1. Feature Selection

After selecting the appropriate dataset, based on domain expertise and the output of the EdgeCloudSim [26] simulation platform, we select the 12 features shown in Table 1.

Table 1. Selected features.

Name	Description	Data type
TaskLength	Task size	Integer
TaskInput	Input data size	Integer
TaskOutput	Output data size	Integer
WLANUpDelay	WLAN uplink latency	Float
WLANDownDelay	WLAN download latency	Float
GSMUpDelay	GSM uplink delay	Float
GSMDownDelay	GSM download delay	Float
WANUpDelay	WAN uplink delay	Float
WANDownDelay	WAN download delay	Float
AvgEdgeUtilization	Edge server utilization	Float
NumoffloadTask	Number of recent uninstalls	Integer
VehicleCount	Number of vehicles	Integer

The 12 selected features are all numeric types (integer or float). Before putting all data into the model to start training, the features need to be standardized via (13):

$$x = \frac{x - mean}{std}, \tag{13}$$

where *mean* and *std* represent the average value and the standard deviation of the features, respectively. Standardizing the input data can avoid large gradient updating during training, avoid the failure of the neural network to converge, and make learning of the model easier.

5.2. Simulation Setting

The simulation is carried out in the EdgeCloudSim platform [26], and the simulation scene is shown in Figure 5. In the movement mode, the road is an elliptical section of 16 km, RSUs are deployed for wireless coverage of the entire road, and each RSU has the wireless signal coverage of a 400 m section in the road. Therefore, the deployment of 40 RSUs can meet the system requirements. Consequently, the elliptical section of 16 km is divided into 40 subsections of 400 m, and in the 40 subsections, vehicle moving speeds are set as 20 km/h, 40 km/h, and 60 km/h. In the initial stage of the system, vehicles are evenly distributed on the 16 km circular road, and the total number of vehicles in the system remains unchanged during operation. Since the sections have vehicles moving at different speeds, the density of vehicles in low-speed sections is higher than that in high-speed sections, indicating that some areas are hot zones and traffic congestion may exist, which is consistent with real situations.

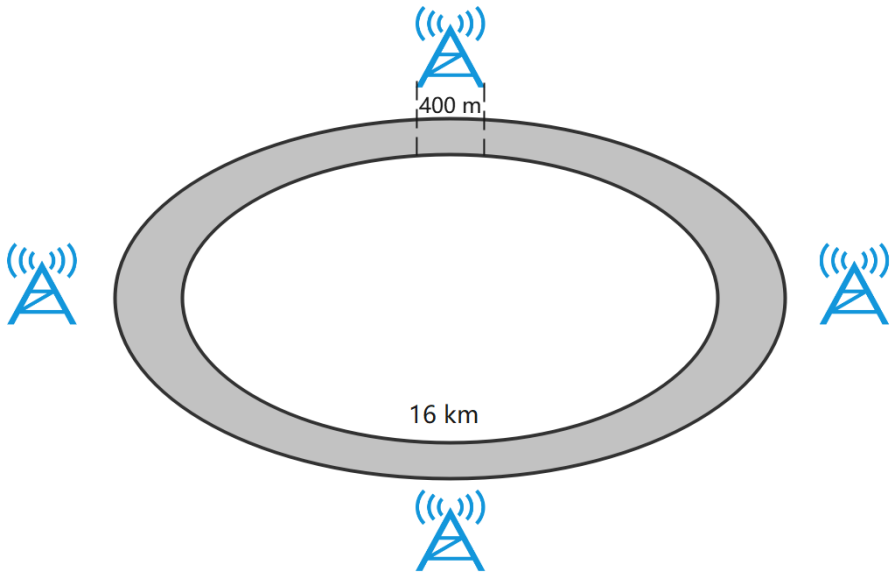

Figure 5. Vehicle movement mode.

The task and related parameters are shown in Table 2 and 3. In the simulation, the tasks are divided into three types, which are navigation tasks, driving assistance tasks, and infotainment tasks, and, respectively, account for 35%, 35%, and 30% of all tasks. The three types of task represent three typical tasks for the Internet of vehicles. Navigation tasks represent tasks that are sensitive to time delay, driving assistance tasks require a large amount of uploaded data, and infotainment tasks represent tasks with a large amount of downloaded data. The time intervals for generating three types of task are set to 3 s, 5 s, and 40 s, respectively. Navigation task requests are the most intensive, driving assistance tasks are the second-most, and infotainment task requests are the least. The lengths of the three types of tasks are set to 3 GI, 10 GI, and 5 GI, respectively, which represents the task demand for computing resources. Navigation tasks have the least demand for computing resources, while driving assistance tasks require the most computing resources.

Table 2. Task parameters.

	Navigation Tasks	Assisted Driving Tasks	Infotainment Tasks
Specific gravity (%)	35	35	30
Interval (s)	3	4	40
Upload/download volume (KB)	20/20	40/20	20/80
Maximum time delay (s)	0.5	1	2
Length of assignment (GI)	3	10	5

Table 3. Other parameters.

Parameters	Value
WAN network bandwidth	1000 Mbs
WLAN network bandwidth	100 Mbs
CN network bandwidth	10 Mbs
RSU coverage radius	200 m
VEC execution speed	800 GIPS
Cloud execution speed	3000 GIPS
Number of vehicles	100–2200
Number of simulation rounds	100

5.3. Simulation Results and Analysis

We first study the impact of some factors on the performance of the model, and these factors include the height of convolution kernel, the number of convolution layers, the number of recombination layers and the number of hidden layers. In the experiment evaluation, AUC (area under curve) and loss are selected as the performance indicators. AUC is the area enclosed by the ROC (receiver operating characteristic) curve and the two coordinate axes, which can be better measure the performance of the two-class model. If the test dataset is given, the larger AUC and the smaller loss represent the better performance of the model.

5.3.1. The Impact of the Height of Convolution Kernel on Model Performance

In this model, the width of the convolution kernel is fixed to 1. The impact of different heights in the performance of the model is shown in Figure 6. When the height of the convolution kernel increases from 2 to 14 (i.e., the number of input features), the performance of the model shows a trend of first increasing and then decreasing. When the height of convolution kernel is 8, the performance is optimal. The experimental results show that the more feature interactions learned by the convolution kernel, the better the performance of the model. However, as the height of convolution kernel increases, the model learns high-level feature interactions, ignoring low-level feature interactions, and model parameters increase, which increases the difficulty of model training and leads to the decline of model performance.

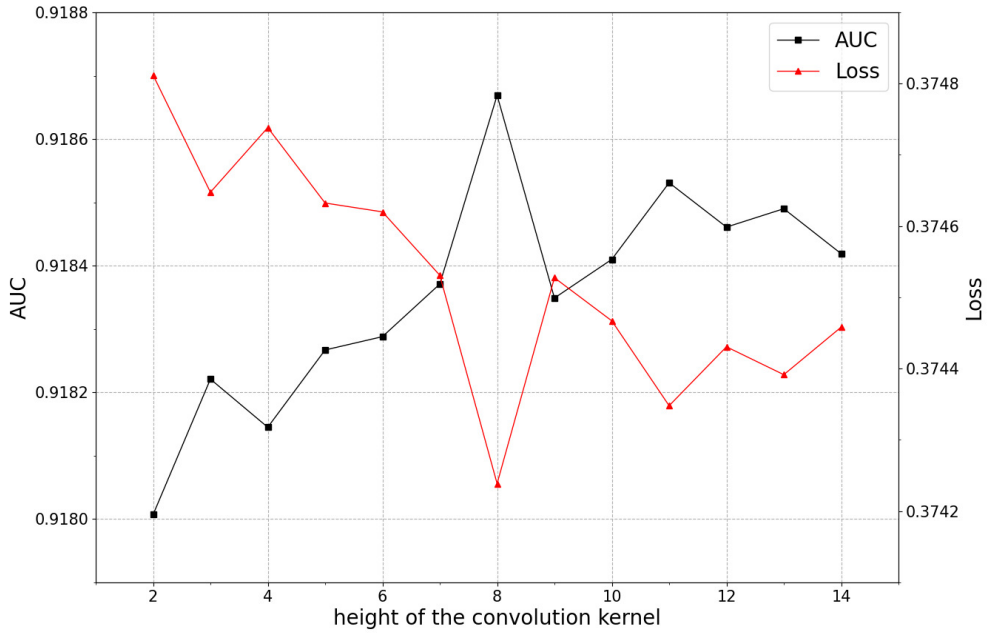

Figure 6. The impact of the height of convolution kernel on model performance.

5.3.2. The Impact of the Number of Convolution Layers on Model Performance

The impact of the number of convolutional layers on model performance is shown in Figure 7. With the increasing number of convolutional layers, the model performance increases. In our opinion, with more convolutional layers, the model has better ability to extract valuable information from both low-level features and high-level features. The shallow convolutional layer learns the intersection of low-level features, and the deep convolutional layer learns the intersection of high-level features. The increasing number of convolutional layers can improve model performance, indicating that the intersection of high-level features is helpful for performance improvement of the model.

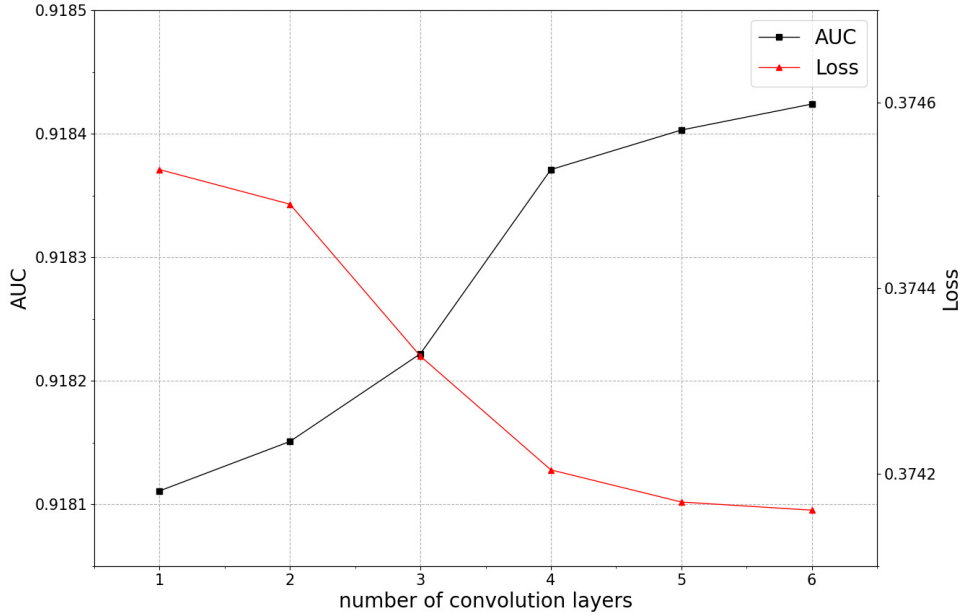

Figure 7. The impact of the number of convolution layers on model performance.

5.3.3. The Impact of the Number of Recombination Layers on Model Performance

In the model, the convolution and pooling feature maps need to be recombined by the recombination layer to generate new features, and the number of newly generated features is determined by the number of recombination layers. The larger the number of recombination layers, the more new features. The impact of the number of recombination layers on model performance is shown in Figure 8. With the number of recombination layers increasing from 0 to 1, the model performance has a certain improvement, indicating that the new generated features can improve the model performance. However, when the number of recombination layers increases from 1 to 4, the model performance decreases, since the number of features increases with an increase in the number of recombination layers. Too many features can bring noise and increase the difficulty of learning. Moreover, more features can reduce the generalization ability of the model, which may easily result in overfit.

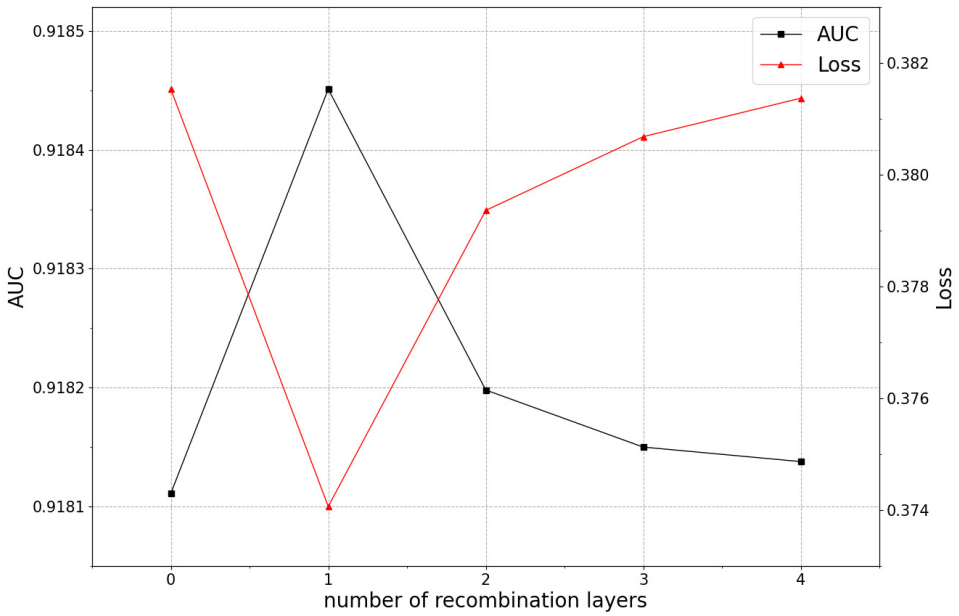

Figure 8. The impact of the number of recombination layers on model performance.

5.3.4. The Impact of the Number of Hidden Layers on Model Performance

The proposed model is composed of a fully connected network, and the main factors affecting the performance of the model are the number and the width of hidden layers. According to the previous practices of deep learning, a deeper model is better than a wider model at the same situation. In this experiment, the width of the hidden layer is set in a decreasing manner. It is supposed that the width of the i hidden layer is L_i, as shown in (14):

$$L_i = \begin{cases} 2^{11-i} & i \leq 4 \\ 2^7 & i > 4 \end{cases} \quad i \in \mathbb{N}. \tag{14}$$

When the number of hidden layers is 6, the widths of the hidden layers are 1024, 512, 256, 128, 128, 128, in order. The impact of the number of different hidden layers on model performance is shown in Figure 9.

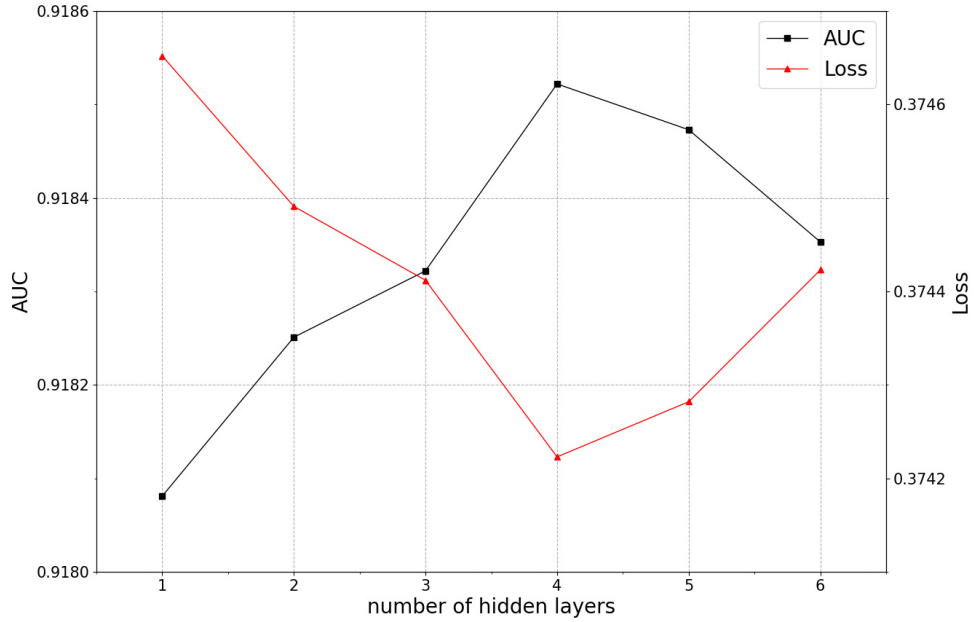

Figure 9. The impact of the number of hidden layers on model performance.

With the increasing number of hidden layers, the model performance shows a trend of first increasing and then decreasing. When the number increases from 1 to 4, the performance has positive correlation with it. When the number is 4, the model has the best performance with the highest AUC and the lowest loss. When the number increases from 4 to 6, the performance shows a negative correlation. At this critical point, the number increasing does not effectively improve the performance, but brings more complexity to the model and needs to process more parameters, resulting in more difficult model training and longer training time. On the other hand, the increasing hidden layers makes the generalization ability of the model worse, which makes the model easy to overfit during the training. To prevent overfit, a regularization process must be added in the training, but this approach may be not helpful. To choose an appropriate regularization process method, more experiments are needed, and training error may be too large, which may affect the judgment of the model performance. It is better to set an appropriate number of hidden layers in the model design stage. The experimental results show that reducing the hidden layer width and deepening the hidden layer depth, and setting the number of hidden layers reasonably, can give full play to the performance of the model.

5.3.5. The Impact of Different Parts of Feature-Generation Module on Model Performance

This section discusses the impact of raw features, new features, and recombination layers on model performance. The experimental results are shown in Figure 10. Four experimental groups are considered: one as the control group, and the other three as the reference groups in which, except for the removal of the specified components, the rest of the model structure is the same as the control group.

(1) The benchmark model is a complete model for task offloading based on CNN automatic feature generation, which serves as a control group for the entire experiment.
(2) The model with the removal of the raw features means that after features are generated, only new features are the input into the prediction model, and raw features are discarded. This is mainly used to examine the impact of the raw features on the performance.
(3) The model with the removal of new features means that newly generated features are not input into the model, and the model is equivalent to the ordinary DNN neural network, which is mainly used to examine the impact of newly generated feature on the performance.

(4) The model with the removal of recombination layer means that newly generated features are the input to the prediction model together with the raw features. However, in this model, the raw features are not recombined after the convolutional layer and the pooling layer, and the output is directly input to the predictive model to evaluate the impact of the recombination layers on the performance.

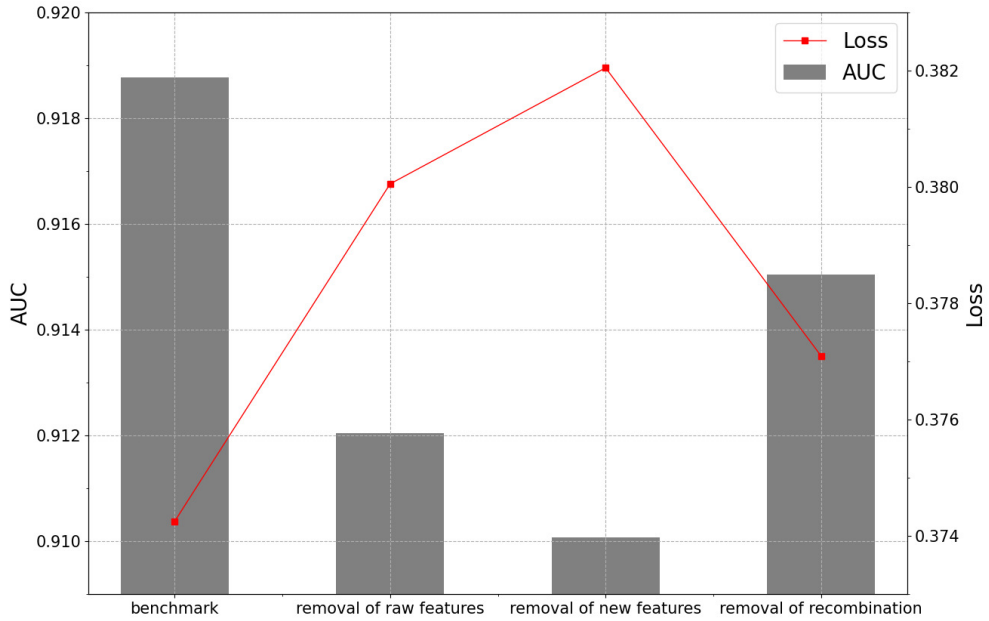

Figure 10. The impact of different parts of the feature-generation module on model performance.

The simulation results in Figure 10 show that any part of the model has an important impact on performance. Referring to the benchmark model, the removal of newly generated features has the most performance degradation. The model with the removal of new features has the worst performance among the three groups, indicating that only relying on raw features cannot optimize model performance. It is also shown in Figure 10 that the removal of the raw features or the new features will both cause the performance to decrease, which shows that the raw features and generated features are both critical to the performance.

Referring to the benchmark model, the removal of the recombination layer leads to the performance decreasing. However, the model with the removal of the recombination layer has better performance than the models removing the raw features or new features. Of the three experimental groups, the model with the removal of the recombination layer has the highest AUC and the lowest loss. In fact, removing the recombination layer does not remove the feature-generation network structure, but reduces the feature density of new features. Therefore, the performance degradation is not significant. The purpose of the recombination layer is to recombine the intersections of adjacent features extracted by CNN to generate a global feature intersection. If the recombination layer is removed, when generating new features, the intersections of features are limited to adjacent features. Consequently, when performing task-offloading results and service-delay prediction, the model without the recombination layer may have features of no practical meaning, which can cause loss in performance.

5.3.6. Performance Comparison with Other Models

To objectively evaluate the superiority of the model proposed in this paper, random, SVM, MLP [19] and DMNet [25] models are compared with our proposed model under the same conditions, and the failure rate of offloading is selected as the performance indicator. It can be observed from Figure 11 that as the number of vehicles in the system increases,

our proposed model always maintains a minimum average offloading failure rate and has the best performance.

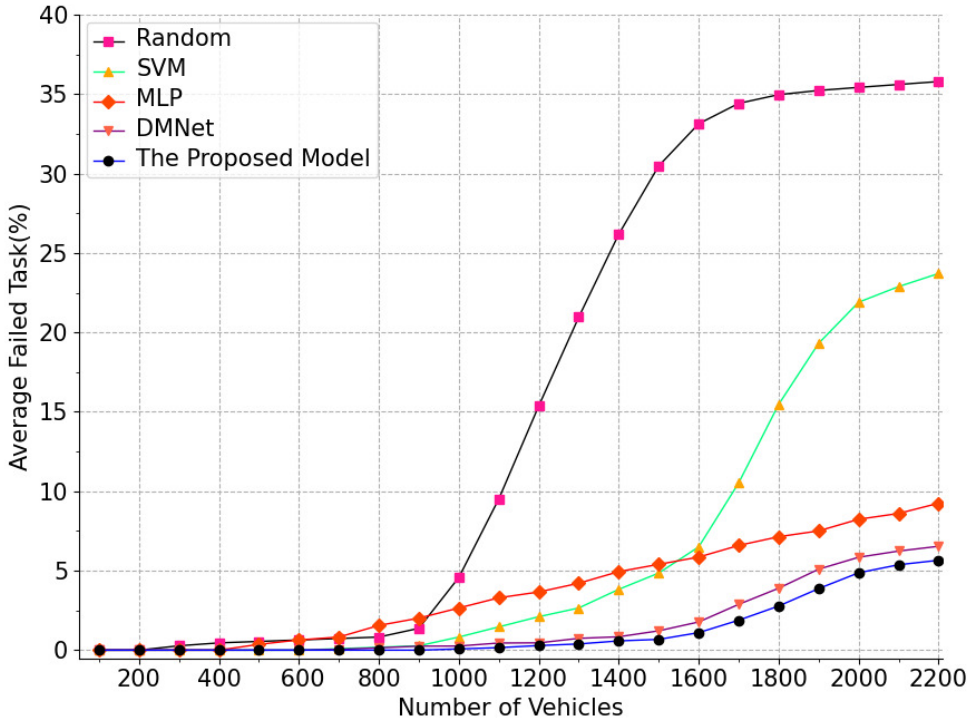

Figure 11. The performance comparison of different models.

As shown in Figure 11, with the increase in the number of vehicles, the average failure rate of task offloading increases, no matter which models are applied in task offloading. As the number of vehicles increases, the number of tasks to be offloaded increases too. When the number of offloaded tasks is close to the maximum load of the system, the service delay of offloading may increase greatly; thus, the failure of task offloading is inevitable. From Figure 11, when the number of vehicles increases from 100 to 700, both DMNet and the proposed model can ensure all tasks are offloaded successfully, since the system load is low. When the number of vehicles increases from 800 to 2200, the task-offloading failure rate of DMNet model is higher than that of the proposed model. The failure rate gap between two different models first increases and then decreases. When the number of vehicles increases from 800 to 1800, the gap gradually increases and reaches the maximum value at 1800. Then, the gap decreases as the number of vehicles increases. When the number of vehicles is 2200, the average failure rate of the two models is relatively close, since the system is close to full load. The average failure rate of the DMNet model increases rapidly from 1300 vehicles, while the proposed model accelerates from 1500 vehicles. The experimental results show that the proposed model can maximize the success rate of task offloading for VEC system. Compared with SVM, MLP and DMNet, the proposed model has a failure rate of task offloading decreased by 21.2%, 6.3% and 2.1%, respectively.

6. Conclusions

In this paper, we have addressed multi-layered vehicle edge-computing architecture in which vehicles can offload tasks via three offloading methods. Since the three offloading methods have their own advantages, the offloading strategies should be optimally chosen for the vehicles according to network status and server load. Our solution in this paper is to predict offloading performance based on historical data, so that the requesting vehicles can find the most effective offloading strategy from the beginning. Therefore, we have designed a deep-learning model suitable for task-offloading result (success/failure) and service-delay prediction, and an automatic feature-generation model based on a CNN has

been proposed to capture the intersection of features to generate new features, avoiding the performance instability caused by manually designed features. Based on performance prediction, the strategy with successful task offloading and the minimum service delay is selected as the final decision. The experimental results have demonstrated that the performance of our proposed model is better than that of other models.

In future work, energy consumption should be considered as an indicator to evaluate model performance, and an offloading strategy for decomposable tasks should also be studied, which needs to take account of the time order of subtask groups, and the relationship between subtasks. With the rapid development of VEC technology, some computation-intensive applications are deployed in vehicles, and huge data may be generated, which places higher demands on the efficiency of deep-learning models. This is also our focus in the future.

Author Contributions: F.Z. and J.T. conceived and designed the experiments; C.L. performed the experiments; F.Z. and X.D. analyzed the data; W.L. contributed reagents/materials/analysis tools; F.Z. and W.L. wrote the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Science Foundation of China (Grant No. 62172450), the Key R&D Plan of Hunan Province (Grant No. 2022*GK*2008) and the Nature Science Foundation of Hunan Province (Grant No. 2020*JJ*4756).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Olariu, S. A Survey of Vehicular Cloud Research: Trends, Applications and Challenges. *IEEE Trans. Intell. Transp. Syst.* **2020**, *21*, 2648–2663. [CrossRef]
2. Foh, C.H.; Kantarci, B.; Chatzimisios, P.; Wu, J.; Gao, D. IEEE Access Special Section Editorial: Advances in Vehicular Clouds. *IEEE Access* **2016**, *4*, 10315–10317. [CrossRef]
3. Liu, L.; Chen, C.; Pei, Q.; Maharjan, S.; Zhang, Y. Vehicular Edge Computing and Networking: A Survey. *Mob. Netw. Appl.* **2021**, *26*, 1145–1168. [CrossRef]
4. Wang, R.; Zeng, F.; Yao, L.; Wu, J. Game-Theoretic Algorithm Designs and Analysis for Interactions among Contributors in Mobile Crowdsourcing with Word of Mouth. *IEEE Internet Things J.* **2020**, *7*, 8271–8286. [CrossRef]
5. Feng, J.; Liu, Z.; Wu, C.; Ji, Y. AVE: Autonomous Vehicular Edge Computing Framework with ACO-Based Scheduling. *IEEE Trans. Veh. Technol.* **2017**, *66*, 10660–10675. [CrossRef]
6. Ahmed, A.; Ahmed, E. A Survey on Mobile Edge Computing. In Proceedings of the 2016 10th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, 7–8 January 2016; pp. 1–8. [CrossRef]
7. Abbas, N.; Zhang, Y.; Taherkordi, A.; Skeie, T. Mobile Edge Computing: A Survey. *IEEE Internet Things J.* **2018**, *5*, 450–465. [CrossRef]
8. Rasheed, A.; Chong, P.H.J.; Ho, I.W.H.; Li, X.J.; Liu, W. An Overview of Mobile Edge Computing: Architecture, Technology and Direction. *KSII Trans. Internet Inf. Syst. (TIIS)* **2019**, *13*, 4849–4864. [CrossRef]
9. Raza, S.; Wang, S.; Ahmed, M.; Anwar, M. A Survey on Vehicular Edge Computing: Architecture, Applications, Technical Issues, and Future Directions. *Wirel. Commun. Mob. Comput.* **2019**, *2019*, 3159762. [CrossRef]
10. Zeng, F.; Chen, Q.; Meng, L.; Wu, J. Volunteer Assisted Collaborative Offloading and Resource Allocation in Vehicular Edge Computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 3247–3257. [CrossRef]
11. Wu, G.; Li, Z. Task Offloading Strategy and Simulation Platform Construction in Multi-User Edge Computing Scenario. *Electronics* **2021**, *10*, 3038. [CrossRef]
12. Chen, J.; Yang, Y.; Wang, C.; Zhang, H.; Qiu, C.; Wang, X. Multi-Task Offloading Strategy Optimization based on Directed Acyclic Graphs for Edge Computing. *IEEE Internet Things J. (Early Access).* **2021**. [CrossRef]
13. Wang, K.; Wang, X.; Liu, X.; Jolfaei, A. Task Offloading Strategy Based on Reinforcement Learning Computing in Edge Computing Architecture of Internet of Vehicles. *IEEE Access* **2020**, *8*, 173779–173789. [CrossRef]
14. Kang, J.; Yu, R.; Huang, X.; Wu, M.; Maharjan, S.; Xie, S.; Zhang, Y. Blockchain for secure and efficient data sharing in vehicular edge computing and networks. *IEEE Internet Things J.* **2019**, *6*, 4660–4670. [CrossRef]

15. Astarita, V.; Giofrè, V.P.; Mirabelli, G.; Solina, V. A Review of Blockchain-Based Systems in Transportation. *Information* **2020**, *11*, 21. [CrossRef]
16. Tanwar, S.; Bhatia, Q.; Patel, P.; Kumari, A.; Singh, P.K.; Hong, W.-C. Machine Learning Adoption in Blockchain-Based Smart Applications: The Challenges, and a Way Forward. *IEEE Access* **2020**, *8*, 474–488. [CrossRef]
17. Liu, P.; Li, J.; Sun, Z. Matching-Based Task Offloading for Vehicular Edge Computing. *IEEE Access* **2019**, *7*, 27628–27640. [CrossRef]
18. Feng, J.; Liu, Z.; Wu, C.; Ji, Y. Mobile Edge Computing for the Internet of Vehicles: Offloading Framework and Job Scheduling. *IEEE Veh. Technol. Mag.* **2019**, *14*, 28–36. [CrossRef]
19. Sonmez, C.; Tunca, C.; Ozgovde, A.; Ersoy, C. Machine Learning-Based Workload Orchestrator for Vehicular Edge Computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 2239–2251. [CrossRef]
20. Yang, B.; Cao, X.; Li, X.; Zhang, Q.; Qian, L. Mobile-Edge-Computing-Based Hierarchical Machine Learning Tasks Distribution for IIoT. *IEEE Internet Things J.* **2020**, *7*, 2169–2180. [CrossRef]
21. Yang, B.; Cao, X.; Li, X.; Kroecker, T.; Qian, L. Joint Communication and Computing Optimization for Hierarchical Machine Learning Tasks Distribution. In Proceedings of the 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 29 June–3 July 2019; pp. 1–6. [CrossRef]
22. Wu, H.; Zhang, Z.; Guan, C.; Wolter, K.; Xu, M. Collaborate Edge and Cloud Computing With Distributed Deep Learning for Smart City Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 8099–8110. [CrossRef]
23. Wang, X.; Wei, X.; Wang, L. A deep learning based energy-efficient computational offloading method in Internet of vehicles. *China Commun.* **2019**, *16*, 81–91. [CrossRef]
24. Wu, S.; Xia, W.; Cui, W.; Chao, Q.; Lan, Z.; Yan, F.; Shen, L. An Efficient Offloading Algorithm Based on Support Vector Machine for Mobile Edge Computing in Vehicular Networks. In Proceedings of the 2018 10th International Conference on Wireless Communications and Signal Processing (WCSP), Hangzhou, China, 18–20 October 2018; pp. 1–6. [CrossRef]
25. Zeng, F.; Liu, C.; Tang, J.; Li, W. Deep Learning-Based Task Offloading for Vehicular Edge Computing. In Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications (WASA 2021), Nanjing, China, 25–27 June 2021; Volume 12939, pp. 291–298. [CrossRef]
26. Sonmez, C.; Ozgovde, A.; Ersoy, C. EdgeCloudSim: An environment for performance evaluation of Edge Computing systems. In Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, Spain, 8–11 May 2017; pp. 39–44. [CrossRef]
27. Liu, B.; Tang, R.; Chen, Y.; Yu, J.; Guo, H.; Zhang, Y. Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction. In Proceedings of the World Wide Web Conference on—WWW '19, San Francisco, CA, USA, 13–17 May 2019. [CrossRef]