applied sciences

MDPI

Article

LTransformer: A Transformer-Based Framework for Task Offloading in Vehicular Edge Computing

Yichi Yang †, Ruibin Yan † and Yijun Gu *

College of Information and Cyber Security, People's Public Security University of China, Beijing 102600, China; 2020212200043@stu.ppsuc.edu.cn (Y.Y.); yanruibin@stu.ppsuc.edu.cn (R.Y.)
* Correspondence: guyijun@ppsuc.edu.cn
† These authors contributed equally to this work.

Abstract: Vehicular edge computing (VEC) is essential in vehicle applications such as traffic control and in-vehicle services. In the task offloading process of VEC, predictive-mode transmission based on deep learning is constrained by limited computational resources. Furthermore, the accuracy of deep learning algorithms in VEC is compromised due to the lack of edge computing features in algorithms. To solve these problems, this paper proposes a task offloading optimization approach that enables edge servers to store deep learning models. Moreover, this paper proposes the LTransformer, a transformer-based framework that incorporates edge computing features. The framework consists of pre-training, an input module, an encoding–decoding module, and an output module. Compared with four sequential deep learning methods, namely a Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), a Gated Recurrent Unit (GRU), and the Transformer, the LTransformer achieves the highest accuracy, reaching 80.1% on the real dataset. In addition, the LTransformer achieves 0.008 s when predicting a single trajectory, fully satisfying the fundamental requirements of real-time prediction and enabling task offloading optimization.

Keywords: edge computing; task offloading; trajectory prediction; deep learning

Citation: Yang, Y.; Yan, R.; Gu, Y. LTransformer: A Transformer-Based Framework for Task Offloading in Vehicular Edge Computing. *Appl. Sci.* **2023**, *13*, 10232. https://doi.org/10.3390/app131810232

Academic Editor: Andreas Sumper

Received: 9 August 2023
Revised: 8 September 2023
Accepted: 11 September 2023
Published: 12 September 2023

1. Introduction

In recent years, research related to edge computing has gradually received extensive attention from researchers [1,2]. Vehicular edge computing (VEC), as a part of edge computing, provides real-time service to vehicular users. It has excellent prospects in the fields of intelligent transportation systems, smart city applications, and vehicular applications.

As the infrastructure becomes well established, edge servers extend their service coverage to a wider scope. In traffic control, edge computing servers can acquire and regulate real-time traffic. In in-vehicle tasks, edge computing servers can provide high-quality services to users. However, the quality of service (QoS) in VEC still cannot be significantly improved, and one of its bottlenecks is the inefficient task offloading. Traditional task offloading methods are plagued by issues such as significant latency, high time and space complexity, and low transmission quality.

To solve the problems of task offloading, trajectory prediction methods are used in the task offloading scheme. For example, tasks which take up a lot of computational resources can be offloaded to other edge servers using predictive-mode multi-hop transmission. Once the vehicle enters the transmission range of the edge server, it obtains the computation results directly [3,4].

The current task offloading schemes mainly focus on resource allocation. Few studies discuss the deployment of advanced trajectory prediction method. Nowadays, deep learning is often utilized for trajectory prediction. However, the computational resources in edge servers make it difficult to deploy common deep learning algorithms. Furthermore, trajectory prediction schemes have relatively poor accuracy in VEC.

Therefore, the current task offloading scheme with predictive-mode transmission encounters two issues: (1) Existing edge servers have limited resources to deploy deep learning models, which consume massive storage and computational resources. (2) Neither the short trajectory prediction based on a Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and a Gated Recurrent Unit (GRU) nor the long trajectory prediction based on the Transformer takes into account the features of VEC.

In order to solve the above problems, we propose the LTransformer, a transformer-based prediction framework in VEC. Meanwhile, a task offloading scheme applied in VEC is proposed. Specifically, the contributions of this paper can be summarized as follows:

(1) We propose a task offloading approach, in which the deep learning model can be deployed for predictive-mode transmission. On the cloud server, the predictive model is trained based on historical trajectory data. On the edge server, real-time trajectory prediction and task offloading optimization are achieved based on the predictive model.
(2) We propose the LTransformer, which has a four-module structure. In the pre-training stage, stationary latitude and longitude data are embedded. In the pre-training stage, stationary latitude and longitude data are embedded. In the input module, multidimensional information such as geography, sequence, and time are integrated. In the encoding–decoding module, encoders and decoders are used to train the trajectory data. In the output module, problematic results are removed using an error correction method.
(3) Experiments are carried out in a real dataset. The proposed method is compared with other deep learning models to analyze its accuracy and applicability in VEC.

The composition of our manuscript is as follows. In Section 2, we describe existing vehicular edge computing schemes and machine learning algorithms related to trajectory prediction. In Section 3, we introduce the task offloading optimization method and the LTransformer. In Section 4, we conduct experiments to analyze the accuracy and efficiency of the LTransformer. Section 5 summarizes the accomplishments and provides an overview of potential avenues for future research.

2. Related Work

2.1. Vehicular Edge Computing

Task offloading in VEC is the process of transmitting the computing task and related parameters from the service requestor to the service providers through Vehicle-To-Vehicle (V2V) and Vehicle-To-Infrastructure (V2I) communications [1]. Saeik et al. [5] summarized the communication issues in task offloading and proposed a novel task offloading scheme that combines edge and cloud resources. An example of vehicular edge computing is shown in Figure 1 below.

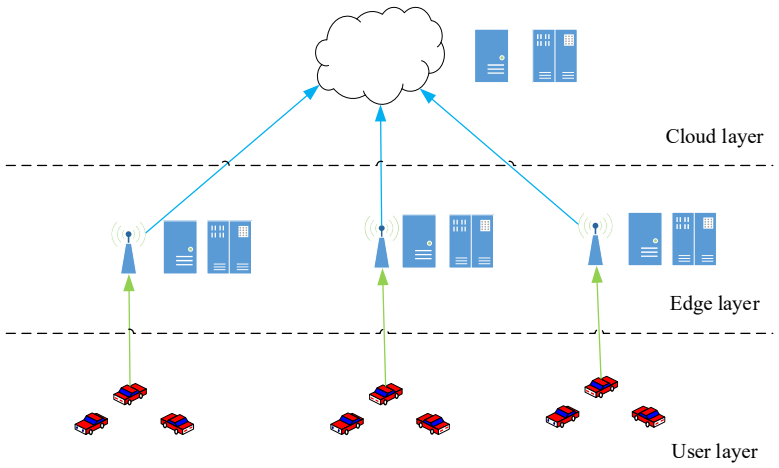

Figure 1. An example of vehicular edge computing.

The resources of edge servers can be fully utilized to provide better QoS to users via optimizing the task offloading scheme. Zhang et al. [4] presented an efficient predictive combination-mode relegation scheme wherein the tasks are adaptively offloaded to the edge servers through direct uploading or predictive relay transmissions. Zhan et al. [3] converted global task offloading optimization problem into multiple local optimization problems with a heuristic mobility-aware offloading algorithm (HMAOA) to approximate the optimal offloading scheme. Yang et al. [6] proposed a low-complexity semiparametric predictive model that takes into account the periodic characteristics and spatial/temporal correlations of dynamic road events. Although these methods have shown some improvements, they still fail to achieve an optimal balance between efficiency and accuracy in VEC. Therefore, how to predict vehicle trajectories more accurately while ensuring efficiency is a pressing issue in task offloading at this stage.

2.2. Trajectory Prediction

Trajectory prediction problems can be categorized into two types based on different data types, namely continuous trajectory prediction problems and discrete trajectory prediction problems. The continuous trajectory prediction problem is a regression problem. Alahi et al. [7] developed an LSTM model which can learn general human movement and predict their future trajectories. Han et al. [8] proposed a short-term real-time trajectory coordinate point prediction method based on a GRU (Gated Recurrent Unit) cyclic neural network. This method improves the accuracy of real-time forecasting by updating the model parameters in real time. Huang et al. [9] discussed a new traffic network modeling algorithm based on the context of traffic intersections that maps vehicle trajectory nodes into a high-dimensional space vector, so that Bi-GRU can be used to bidirectionally model the trajectory matrix for the purpose of prediction. Amichi et al. [10] designed a two-step predictive framework solely based on personal location data. This framework aims to address the prediction of visits to new places and adjust prediction resolution to account for probable explorations of new locations.

Monreale et al. [11] proposed a T-pattern tree for trajectory prediction. The tree is constructed using trajectory patterns that represent specific areas, and it can serve as a predictor for the next location of a new trajectory by identifying the best-matching path within the tree. Dong et al. [12] put forward a new method named RTMatch to predict the future location of a moving object using the storage structure, RTPT and HT, which can be updated dynamically and provide dynamic analysis of trajectory pattern according to real-time information. Zeng et al. [13] presented a next-location prediction approach based on an RNN and self-attention mechanism to predict trajectory patterns based on a sequence of discrete nodes. Feng et al. [14] proposed DeepMove, an attentional recurrent network for mobility prediction from lengthy and sparse trajectories. DeepMove effectively utilizes the periodicity nature to augment the RNN for mobility prediction. Liu et al. [15] created a geographically temporally awareness hierarchical attention network (GT-HAN) to distinguish different user preferences.

Recent research proves that the Transformer outperforms other deep learning methods in trajectory prediction. Amirloo et al. [16] proposed LatentFormer, a transformer-based model able to predict future vehicle trajectories by leveraging a novel technique to model interactions among dynamic objects in the scene. Accounting for the interaction between vehicles, Yan et al. [17] proposed two spatial attention mechanisms to help the model understand the surrounding environment better and thus improve its prediction accuracy. Yu et al. [18] introduced the Spatio-Temporal grAph tRansformer (STAR) framework, a novel framework for spatio-temporal trajectory prediction based purely on a self-attention mechanism, with TGConv, a Transformer-based graph convolution mechanism. Dai et al. [19] proposed a novel neural architecture, Transformer-XL, which enables learning dependency beyond a fixed length without disrupting temporal coherence. Wang et al. [20] used a low-rank approximation method to approximate a self-attention mechanism, which maintains high performance while reducing the computational cost.

Kitaev et al. [21] introduced reversible residual layers that reduce the memory consumption of the model and give the model the ability to handle larger datasets. Kong et al. [22] proposed the Spatial-Temporal Graph Attention Network (STGAT) for traffic flow forecasting. They demonstrated that STGAT can be generalized directly not only to graphs with an arbitrary structure, but also to completely unseen graphs. None of the existing deep-learning-based prediction methods consider the features in VEC. These deep learning methods need a large amount of storage and computational resources. However, edge servers have limited resources, which leads to the fact that these methods cannot be directly applied to VEC.

3. Method

There are two problems with the existing research: (1) in VEC, task offloading methods based on trajectory prediction encounter limitations in resources, which impedes the deployment of deep learning; (2) in deep learning, existing deep learning methods cannot be directly applied to VEC, and do not incorporate the features in VEC.

For (1), a task offloading optimization approach is proposed. It supports the deployment of deep learning models to optimize task offloading through prediction results. In this approach, the model is trained on the cloud server and stored in edge servers to provide trajectory prediction services in real time, which consequently optimizes task offloading. For (2), the LTransformer is proposed and applied to the task offloading in VEC. The proposed model uses the Transformer as the overall architecture and incorporates the stationary and adjacent features of edge servers.

3.1. Task Offloading

Existing machine learning methods consume a lot of edge server computational resources. Therefore, we add a portion of cloud computing to VEC, in order to break through the limitation of edge server resources. In our optimization approach, resource-hungry computational tasks are implemented in the cloud server. The edge servers only store the well-trained model and predict the trajectory based on the data given from vehicles.

3.1.1. Task Offloading Optimization Approach

The task offloading optimization approach can be divided into the training process and the prediction and optimization process. The training process involves gathering historical trajectory data, training the predictive model, and deploying the model. The input data of this process are the trajectory data stored by the vehicle, and the output datum of this process is the trajectory predictive model stored by edge servers. The detailed steps are shown in Figure 2a. The prediction and optimization process encompasses the collection of trajectories for prediction, the generation of prediction results, and the optimization of computational tasks. The input data are the trajectory data stored by the vehicle, and the output data are the results of the computational task transmitted to the vehicle, the detailed steps of which are shown in Figure 2b.

The training process can be divided into three steps, each of which corresponds to the serial number in Figure 2a. The details of the training process are as follows:

Input: the vehicle trajectory set A is stored in the vehicle set V, where $A = \{T_1, T_2, \ldots, T_n\}$, $T = \{s_1, s_2, \ldots, s_l\}$, $V = \{v_1, v_2, \ldots, v_n\}$.

Output: the predictive model M is stored in the set edge servers, S, where $S = \{s_1, s_2, \ldots, s_m\}$.

1. Historical vehicle trajectories stored in the vehicle are uploaded into the edge server. Specifically, the vehicle trajectory T_h stored in the vehicle v_h is uploaded to the surrounding edge server s_h through the roadside units (RSU).
2. Edge servers upload vehicle trajectories to the cloud server. Specifically, the vehicle trajectories, $T_p, \ldots, T_q$ stored in the edge server s_h are uploaded to the cloud server c.
3. The predictive model is trained on the cloud server based on the trajectory data of vehicles. Once the training process is completed, the predictive model is transmitted to the edge servers. Specifically, the predictive model M is trained on the cloud

server c based on the vehicle trajectory set T. After training, the predictive model M is transmitted to each edge server s_h.

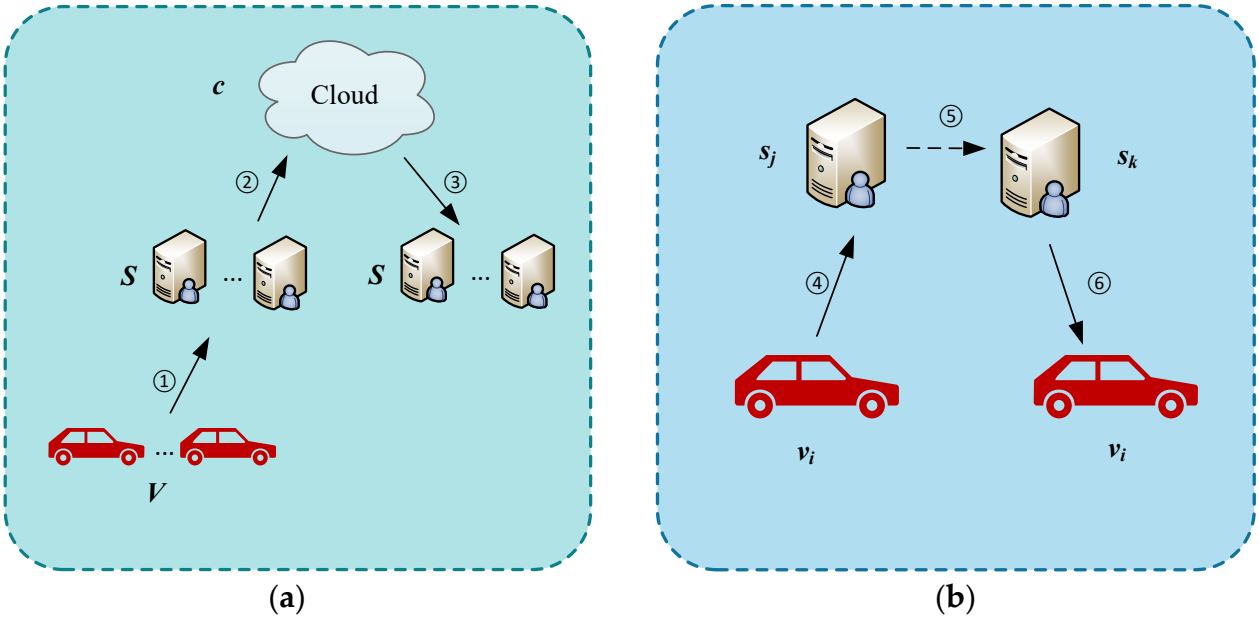

Figure 2. (**a**) Training process; (**b**) prediction and optimization process.

The prediction and optimization process is similarly divided into three steps, each of which corresponds to a serial number in Figure 2b. The detailed process of prediction and optimization is as follows:

Input includes the following three components:

- The computational task D_p.
- The vehicle trajectory T_i is stored in the vehicle v_i.
- The model M is stored in the edge server s_j.

Output: the completed computational task D_q stored in the vehicle v_i.

4. Computational tasks and vehicle trajectories stored in the vehicle are uploaded onto the edge server. Specifically, the computational task D_p and vehicle trajectory T_i stored in the vehicle v_i are uploaded into the surrounding edge server s_j via the RSU.
5. The edge server uses the predictive model to predict the vehicle trajectory and optimizes the task offloading according to the prediction result. Specifically, the vehicle trajectory, T_i, is predicted in the edge server s_j using the predictive model M. Assuming the prediction result is the edge server s_k, the computational task or the result of the task is transmitted to the edge server s_k via V2V or V2I.
6. The vehicle downloads the results of the computational task as it arrives around the predicted location. Specifically, the computational task result D_q in the edge server s_k is transmitted to the vehicle v_i when the vehicle v_i arrives in its vicinity.

The task offloading optimization approach has many applications. For example, in smart cities, this approach can be used to deploy deep learning in VEC to predict and control the overall flow of vehicles, and thus optimize the traffic situation. In in-vehicle services, this approach also deploys deep learning to improve the QoS of users.

3.1.2. Search Stage and Energy Management

In edge computing, we propose a task offloading strategy that combines the dynamism of edge computing networks with the hardware environment. Let r_s represent the remaining computational resources of the server. f_s represents the computing efficiency. P_s is the server's computational power. Computational tasks typically consist of several sub-tasks.

Therefore, we define N consecutive sub-tasks as $n = \{1, 2, \ldots, N\}$, and the computational resource required for task n as r_n. After receiving a computational task, there are two scenarios that need to be considered in each edge server:

1. The current edge server is capable of completing the computational task on its own. In that case, the energy consumption required for the edge server to complete computational task n is
$$E = \frac{r_n P_s}{f_s},\ r_s > r_n \tag{1}$$
2. The remaining computational resources on this edge server cannot meet the minimum requirements of this computational task. The edge server transfers this computational task to an edge server in the direction of the vehicle's movement. In that case, the total energy consumption required for the edge server to complete computational task n is
$$E = E_T + E_p + \frac{r_n P_s}{f_s},\ r_s \leq r_n \tag{2}$$

where E_T represents the energy consumption for task transmission, and E_p represents the energy consumption for trajectory prediction. Namely, if each edge server completes the computational task in the first scenario, the energy consumption is minimized. However, if an edge server handles a lot of computational tasks in the first scenario, it may lead to the depletion of resources on this edge server. Therefore, this strategy can be used to search for idle resources on edge servers and reduce the occurrence of denial-of-service incidents caused by computational task accumulation. The relevant pseudocode is presented in Algorithm 1 below.

Algorithm 1. LTransformer-based Predictive-mode Task Offloading Algorithm.

```
for server in edge_servers:
    while data = receive_data():
        if data.r_n> server.r_s:
            trajectory = receive_trajectory(data.vehicle)
            next_server = predict(trajectory)
            data.send(next_server)
        else:
            results = exec(data.task)
            results.send(data.vehicle)
```

It is worth noting that if the predicted edge server still faces resource constraints, it will continue to follow the second scenario for execution. However, prediction tasks will update the trajectory as the vehicle moves, thus dynamically altering the prediction results in real time.

As a result, trajectory prediction becomes a crucial component of this optimization approach. To enhance prediction accuracy, we introduce the LTransformer.

3.2. LTransformer

In the task offloading optimization process, this paper proposes the LTransformer framework for trajectory prediction. The input datum of this framework is the set $T = \{s_1, s_2, \ldots, s_n\}$, where $s_i \in \mathbb{R}^{d_{in}}, d_{in} = 2$. s_i denotes the edge server nodes and d_{in} denotes the input dimension. Each edge server node s_i includes (1) the timestamp of the vehicle trajectory when it passes through this node; (2) the serial number of the edge server node. The output datum of this framework is the prediction result, p, where $p \in \mathbb{R}^{d_{out}}$, $d_{out} = 2$. The prediction result p represents the position of the predicted edge server node and d_{out} denotes the output dimension. p also contains two dimensions, namely the longitude and

latitude of the edge server node. Therefore, this prediction process is defined as a function on any vehicle trajectory set, T.

$$f(T) = p \tag{3}$$

The optimization objective of the LTransformer framework is

$$\begin{gathered} \min \sum_{i=1}^{m} \Pr\left(p = \hat{p_1}\right) \log(\Pr(p = p_i)) \\ s.t.\ p_i \in \{p_1,\ p_2, \ldots, p_m\} \end{gathered} \tag{4}$$

$\Pr(p = p_i)$ denotes the probabilities that the prediction result p is the edge server node with the serial number i, and p_i belongs to a finite set of edge server nodes.

The LTransformer consists of four modules: the pre-training module, the input module, the encoding–decoding module, and the output module. The LTransformer uses the Transformer framework in general, and improves on the original Transformer by improving the input module and adding the pre-training and the output modules. In the pre-training module, stationary latitude and longitude data are embedded into the multidimensional space. There are three aspects considered in the input module, which are location, position, and time. In the location aspect, we use trainable embedding and local linear embedding (LLE). In the position aspect, we combine the trajectories with their serial numbers. In the time aspect, we add temporal information using the method in Informer [23]. In the output module, this paper proposes an error correction mechanism to remove the faulty results. The overall framework of the LTransformer is shown in Figure 3.

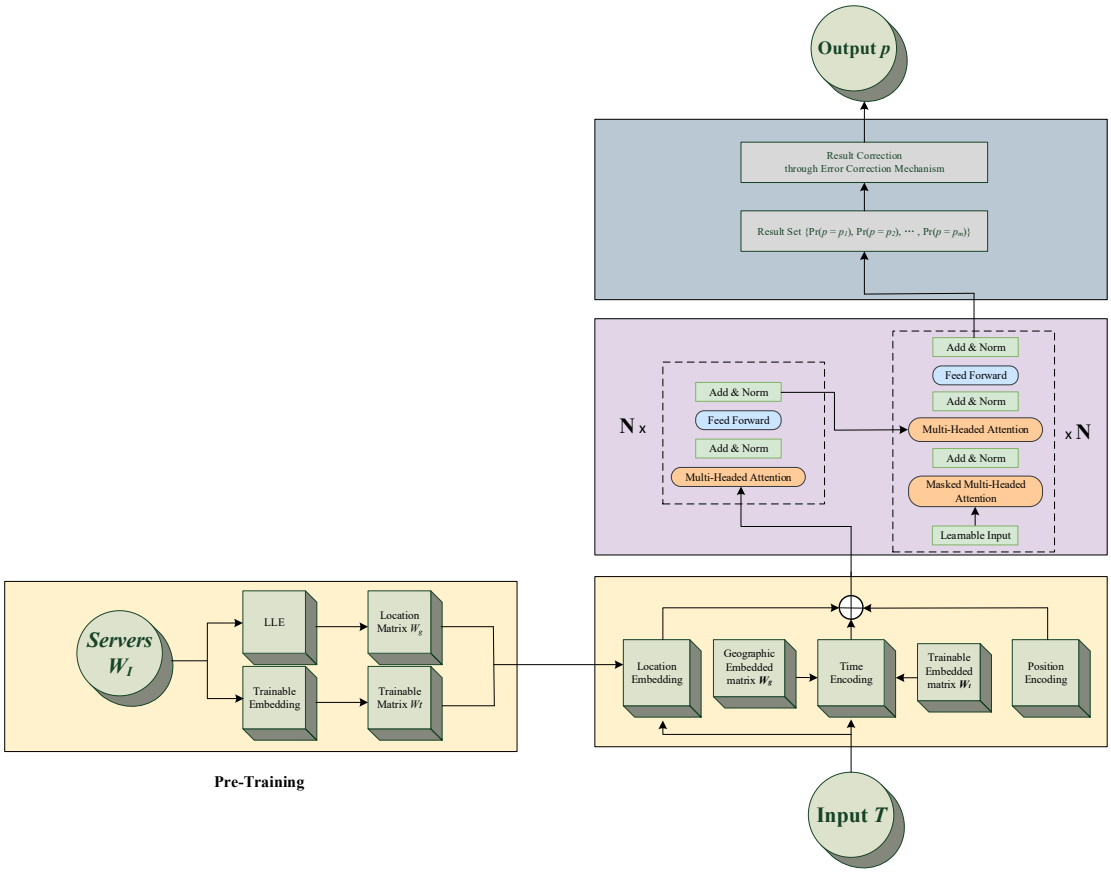

Figure 3. Overall framework of the LTransformer.

The performance of the prediction is increased. In pre-training, the parameters for training are reduced to save time. In the input module, multiple aspects are considered to improve the accuracy. In the output module, faulty results are removed to correct the output.

3.2.1. Pre-Training

In VEC, the edge servers' geographic locations are stationary. During pre-training, the latitude and longitude of each edge server node are incorporated and embedded into a multidimensional space. Meanwhile, we observed the location of the edge server nodes in a vehicle trajectory set. There is a local linear relationship between the nodes. In other words, the latitude and longitude between nodes are roughly linearly distributed in the same vehicle trajectory set. During the training period, the model's parameters are adjusted through backpropagation to capture the location relationship between nodes. Therefore, in pre-training, we use LLE and trainable embedding methods. The process of pre-training is depicted in Figure 4.

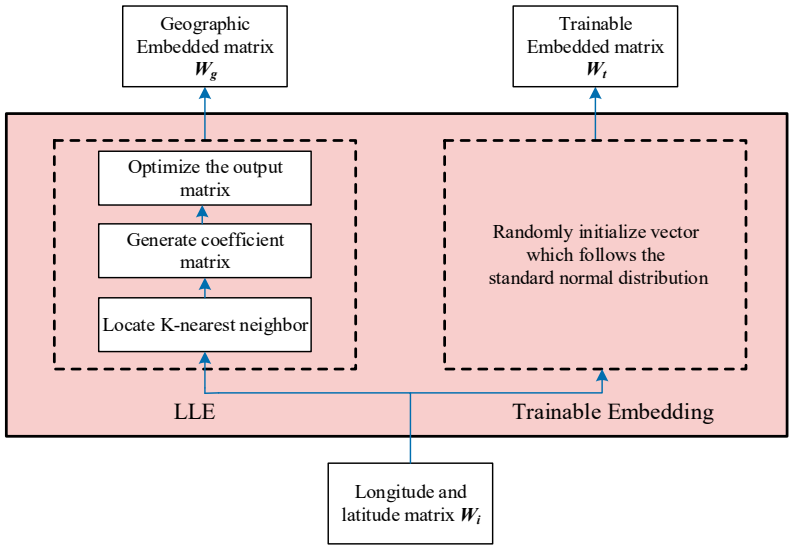

Figure 4. Pre-training.

In pre-training, according to the node order, a matrix, W_i, is generated as the input to trainable embedding and LLE. The matrix W_i consists of nodes a_i which represent the edge server node and only contains location information.

Trainable Embedding

The trainable embedding maps the a_i of the matrix W_i from a two-dimensional space to a multidimensional space, following the standard normal distribution. According to the results of hyperparameter tuning, 512-dimensional input data are the best hyperparameters, leading to the best prediction results. Therefore, trainable embedding generates the trainable matrix W_t using the matrix W_i, $W_t \in \mathbb{R}^{m\times 512}$, where m denotes the total number of edge server nodes; i.e., each row of the matrix W_t corresponds to the latitude and longitude features of a node and numerically follows a normal distribution. During the training process, backpropagation updates the parameters of trainable embeddings and reduces the cross-entropy loss.

LLE

LLE generates the geographic embedded matrix W_g through two optimization processes. The embedded matrix W_g effectively integrates the positional stationarity and local linear relationship of edge server nodes.

Optimization Process 1: Generate Weight Coefficients Based on Local Linear Features

Optimization Process 1 aims to calculate the weight coefficient, w_{ij}, which is an intermediate result that preserves the local linear relationship.

Initially, the Euclidean distance is computed between each node and other nodes. Subsequently, k neighboring nodes are selected with the closest Euclidean distances. Then,

k weight coefficients w_{ij} corresponding to each node a_i are optimized, where the weight coefficient w_{ij} denotes the weights between node a_i and its neighboring node a_j. Since the mean square error can reflect a linear relationship, the optimization process involves calculating the minimum value of the mean square error with the constraint that the sum of the weight coefficient w_{ij} of each node is 1, i.e., the normalization. The optimization process is shown as follows:

$$\begin{array}{c} \min \sum_{i=1}^{m} \| a_i - \sum_{j \in Q(i)} w_{ij} a_j \|_2^2 \\ s.\,t. \sum_{j \in Q(i)} w_{ij} = 1 \end{array} \tag{5}$$

where the set $Q(i)$ represents the set of serial numbers corresponding to the neighboring nodes of node a_i. Assuming k = 2 and the 2 neighboring nodes of a_1 are selected as a_2 and a_3, then $Q(1)$ = {2, 3}.

Optimization Process 2: Generate the Geographic Embedded Matrix Based on the Weight Coefficients

In order to maintain the local linear relationship between each node after embedding, LLE generates the geographic embedding matrix W_g based on the weight coefficient w_{ij}. Therefore, the optimization process involves computing the minimum of the mean square error after embedding and the constraints are the normalization of the embedded vectors. The optimization process is shown as follows:

$$\begin{array}{c} \min \sum_{i=1}^{m} \| y_i - \sum_{j \in Q(i)} w_{ij} y_j \|_2^2 \\ s.\,t. \sum_{i=1}^{m} y_i = 0; \frac{1}{m} \sum_{i=1}^{m} y_i {y_i}^T = I \end{array} \tag{6}$$

Based on the training results, the LTransformer sets the dimension of y_i to 512. All y_i vectors after embedding constitute the geographic embedding matrix W_g, $W_g \in \mathbb{R}^{m \times 512}$, where m denotes the quantities of edge server nodes.

3.2.2. Input Module

The input module consists of three components: location embedding, position encoding, and temporal encoding, which encode geographic, sequential, and temporal information, respectively. The input datum of this module is a set consisting of a multiple vehicle trajectory set, T. We use a matrix, M, to represent the input data, i.e.,

$$M = \begin{bmatrix} s_{11} & \cdots & s_{1q} \\ \vdots & \ddots & \vdots \\ s_{p1} & \cdots & s_{pq} \end{bmatrix} \tag{7}$$

where $M \in \mathbb{R}^{p \times q \times d_{in}}$, p denotes the quantity of trajectories (batch size) at each training epoch, q denotes the length of the longest trajectory, and d_{in} denotes the dimension of each node.

In the input module, the matrix M can be divided into two parts according to d_{in}, the matrix M_l (d_{in} = 1), containing serial information, and the matrix M_s (d_{in} = 1), containing temporal information.

Location Embedding

The input of location embedding process is the matrix M_l. It is generated by the results of pre-training process. Specifically, two embedded matrices, M_t and M_g, are generated by replacing the serial numbers in M_l with the corresponding embedded vectors in the trainable embedded matrix W_t and the geographic embedded matrix W_g, where $M_t \in \mathbb{R}^{p \times q \times 512}$, $M_g \in \mathbb{R}^{p \times q \times 512}$. Finally, the matrix M_t and the matrix M_g are summed by weights to obtain the matrix M_L, which represents the location information.

$$M_L = \theta_1 M_t + \theta_2 M_g \tag{8}$$

where θ_1 and θ_2 are the weights of the matrix M_t and the matrix M_g. The weights will be updated in the backpropagation process.

Position Encoding

The positional information determines the order of the nodes in the trajectory, so position encoding is critical. The LTransformer follows the Transformer's position encoding method; i.e., position information is represented by trigonometric functions. The encoding results are computed for each position and dimension, i.e.,

$$PE(t,i) = \begin{cases} \sin\left(\frac{t}{10000^{i/d}}\right), & if\ i\ is\ even \\ \cos\left(\frac{t}{10000^{(i-1)/d}}\right), & if\ i\ is\ odd \end{cases} \tag{9}$$

where t denotes the position number, i denotes the dimension, and d denotes the dimension after embedding. For each element in M and each of its dimensions, M_P is calculated according to Equation (7). The calculation is shown in Equation (8).

$$M_P = \begin{bmatrix} PE(1,i) & \cdots & PE(q,i) \\ \vdots & \ddots & \vdots \\ PE(1,i) & \cdots & PE(q,i) \end{bmatrix},\ i = 1,2,3,\ldots 512 \tag{10}$$

The result of position encoding is the matrix M_P, representing the sequential information.

Time Encoding

In VEC, there is a temporal pattern in the vehicle trajectories. For example, traffic volumes and overall direction of movement in the morning are different from those in the evening. Therefore, the LTransformer framework uses the time encoding in Informer, which is able to express the temporal pattern. First, the LTransformer selects five aspects of temporal pattern expression based on the training results, which are hour, day, week, month, and year. Then, we calculate each expression of the timestamp in the matrix M_s. The calculation of each aspect has its own characteristic; e.g., in the hour aspect, we calculate the proportionality of the current minute in an hour. The specific formulation of each aspect is as follows:

$$\begin{gathered} h = minute/60 \\ d = hour/24 \\ w = day/7 \\ m = day/30 \\ y = day/365 \end{gathered} \tag{11}$$

After the calculation, the results are concatenated and embedded into a 512-dimensional space using a linear layer to generate M_T, where $M_T \in \mathbb{R}^{p \times q \times 512}$. The matrix M_T represents the temporal information.

$$M_T = LN(Concat(h, d, \mathrm{w}, m, y)) \tag{12}$$

It is demonstrated in the Transformer that if the dimensions are summed directly, the distinctions and relationships between the dimensions can be captured during the training process. Therefore, in the input module, the summation of M_L, M_P, and M_T represents the geographic, sequential, and temporal information. It is calculated to generate the output M_O of the input module, i.e.,

$$M_O = M_L + M_P + M_T \tag{13}$$

3.2.3. Encoding–Decoding Module

The main structure of the encoding–decoding module is similar to that of the Transformer, comprising N encoders and N decoders. Each encoder or decoder includes multi-

head attention with p heads and a fully connected neural network with a q-dimensional hidden layer, which employs residual concatenation and normalization after each encoding or decoding. Based on the training performance, the LTransformer sets the hyperparameters to $N = 6$, $p = 16$, and $q = 4096$.

The features of the vehicle trajectories and edge server nodes are trained via encoders. The decoder adjusts the weights to reduce the cross-entropy loss by forcing learning, so that the prediction results gradually become closer to the real ones during the training process. In addition, since the input module generates the matrix with geographic, sequential, and temporal information, the multidimensional features of vehicle trajectories and the edge server nodes can be captured by the multi-head attention in the encoding–decoding module. Thus, the prediction of the vehicle trajectory is generated with multiple aspects' information.

3.2.4. Output Module

The output module filters the results using the error correction mechanism, which takes into account the adjacent relationships of edge servers. Specifically, in VEC, the result of trajectory prediction must be adjacent to the last node of the vehicle trajectory. If the prediction is not adjacent to the last node, the result will be detected as an error. Therefore, this paper proposes the error correction mechanism to prevent erroneous results.

First, we define the concept of the adjacent node; if the binary trajectory sequence T_j is a subsequence of any trajectory T_i, i.e., $T_j = \{s_p, s_q\}$, $T_i = \{s_1, s_2, \ldots, s_m\}$, satisfying $T_j \subseteq T_i$, then we refer to the node s_p in the trajectory set T_j as the adjacent node of s_q. Based on this definition, this paper details the process of the error correction mechanism, as shown in Figure 5.

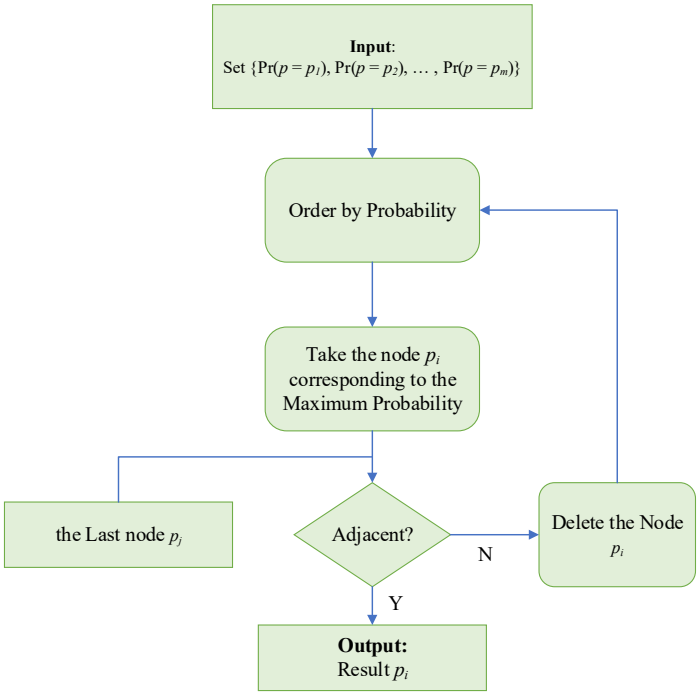

Figure 5. Error correction mechanism.

In the error correction mechanism, first, the prediction results are ranked in order according to their probability. Second, the node p_i corresponding to the maximum probability is obtained. Third, it is determined whether the node p_i is the adjacent node of the last node of the vehicle trajectory. If the node p_i is not the adjacent node, this node is deleted and the results are ranked again. Finally, if the node p_i is the adjacent node, we consider the output p_i the final prediction result of the LTransformer.

The LTransformer fully considers the features of edge computing, and this framework can be applied to in-vehicle edge computing. Specifically, traffic control and in-vehicle services require efficient task offloading mechanisms, and the LTransformer framework can optimize the task offloading mechanisms to provide high-quality services.

3.3. LTransformer Complexity

In this subsection, we will analyze the time complexity of the LTransformer in various stages and use this analysis to assess its feasibility for application in VEC.

Since pre-training will only be performed once, the time complexity of the LTransformer primarily lies in the analysis of the encoding–decoding module. During training, the time complexity of the encoding–decoding module is linked to the computation of Attention.

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(QK^T\right)V \tag{14}$$

where $Q, K, V \in \mathbb{R}^{n \times d}$, n refers to the length of the sequence, and d is the dimension.

The main computational step in the above equation involves calculating the similarity using QK^T, which is essentially a matrix multiplication between $n \times d$ and $d \times n$, resulting in an $n \times n$ matrix with a complexity of $O(n^2d)$. The time complexity of the SoftMax function is $O(n)$; thus, the overall time complexity of the formula is $O(n^2d)$.

In the predication process, the time complexity of the LTransformer remains the same as during training. Similarly, its time complexity is also $O(n^2d)$. During prediction, the sequence length (i.e., n) is generally not long, resulting in fast computation speed, which can meet the demands of edge computing.

It is worth noting that the LTransformer model is continuously updated and iterated on the cloud server. Over time, the LTransformer will perform incremental training based on the latest trajectory data, updating itself to achieve higher accuracy. Due to the pre-training process, the LTransformer does not need to train the parameters in different dimensions. Therefore, when updating parameters in the LTransformer, the required training time can meet the demands of regular version iteration.

4. Experiment and Analysis

4.1. Experimental Environment and Dataset

This section introduces the experimental environment. The experiment analysis was conducted on a Linux server, and the source code was written in Python. The detailed platform parameters are shown in Table 1.

Table 1. Parameters of the experimental platform.

Software and Hardware Environment	Parameters
CPU	Intel(R) Xeon(R) Gold 6326
GPU	NVIDIA A40
Operating System	Linux 3.10.0
Python	python 3.10.10
Pytorch	1.12.1+cu113

The experimental data were real vehicle trajectory data detected using real equipment in a city in China. Vehicle trajectory data include four fields, namely longitude, latitude, timestamp, and serial number, as presented in Table 2.

Table 2. Source and fields of the experimental dataset.

Source	Field Name	Field Type
Real trajectory data	Longitude	Double-precision floating point
	Latitude	Double-precision floating point
	Timestamp	Integer
	Serial number	Integer

4.2. Data Processing

This section primarily discusses the data processing, training process, and prediction process during the experiment. Each trajectory in the dataset was composed of several nodes. Each node contained information about latitude and longitude and a timestamp indicating the time at which the vehicle passed a specific geographical location. Let the length of the trajectory be n.

Before starting the experiment, each trajectory was split into a sequence of length n-1 and the last node. The sequence of length n-1 served as the input to the model. The latitude and longitude of the last node served as the ground truth for the model's predictions. In other words, the model needed to predict the location of the last node of the trajectory based on the sequence of the preceding n-1 nodes.

During the training process, we divided the dataset into training and testing sets in a ratio close to 2:1. All baseline algorithms and the LTransformer were trained using the teacher forcing mode.

During the prediction process, we put the test dataset into the trained model and compared the obtained results with the ground truth. Finally, we calculated the accuracy according to the comparison results.

4.3. Approach Comparing

In order to compare and analyze the performance of the LTransformer, the following baseline methods were chosen in this experiment.

RNN [24]: the RNN (Recurrent Neural Network) is a deep learning method which is always used to predict sequential data.

LSTM [25]: LSTM (Long-Short Term Memory) improves the RNN with gated structures and solves the short-term memory problem of the RNN.

GRU [8]: the GRU (Gated Recurrent Unit) simplifies LSTM with two gating mechanisms (reset gate and update gate) and solves the slow loss descent problem.

Transformer [26]: the Transformer performs attention mechanisms in the encoder layers and decoder layers to analyze or predict sequential data.

Additionally, the parameters of the model affect the training and prediction performance, and the corresponding experimental parameters are shown in Table 3.

Table 3. Models and corresponding parameters.

Model	Hidden Layer Dimension	Batch Size	Optimizer
RNN	512	200	Adam
LSTM	512	200	Adam
GRU	512	200	Adam
Transformer	512	200	SGD
LTransformer	512	200	SGD

The RNN, LSTM, and GRU typically achieve better performance with the Adam optimizer, while their loss reduction is slower when using the SGD optimizer. Conversely, the Transformer tends to show a suboptimal performance with the Adam optimizer, while the SGD optimizer proves to be more suitable and effective.

4.4. Accuracy Verification

We conducted a comparative analysis by training all of the models for the same epochs on our dataset. We define accuracy as

$$Accuracy = \frac{T}{T+F} \tag{15}$$

where T denotes the number of correct predictions and F denotes the number of incorrect predictions. Following this definition, the experiments compared and analyzed the accuracy of each model. All models were trained for the same rounds using the same training set and loss function. To reduce experiment variability, we selected three different samples from the dataset. Each sample contained the same number of trajectories but was drawn from different parts of the dataset. The specific results are presented in Tables 4 and 5.

Table 4. Accuracies in different samples of each model.

Sample	GRU	LSTM	RNN	Transformer	LTransformer
Sample 1	0.783	0.784	0.782	0.780	0.788
Sample 2	0.802	0.801	0.797	0.801	0.803
Sample 3	0.812	0.808	0.809	0.809	0.813

Table 5. Average accuracy and loss of each model.

Model	Accuracy	Loss
GRU	0.799	0.479388
LSTM	0.798	0.479554
RNN	0.796	0.498660
Transformer	0.797	0.249246
LTransformer	0.801	0.248059

Tables 4 and 5 demonstrate that the LTransformer achieved higher accuracy and a better fitting performance compared to existing commonly used sequential deep learning methods. In terms of accuracy, the LTransformer improved the average accuracy to 80.1%. In terms of loss, the LTransformer reduced the loss to 0.248. This suggests that the LTransformer has a superior predictive performance. As the loss value reflects the fitting status, the correlation between the loss and the number of epochs was recorded during training. Figure 6 shows the specific results.

Figure 6 records the average loss per round for each model during training. After extensive experimental validation, we found that after 30 rounds of training, the loss rates of all models did not change significantly anymore. Therefore, we chose 30 rounds as the number of training rounds for all models. This suggests that the LTransformer has a slower rate of loss reduction, which is attributed to the incorporation of multiple dimensions in the LTransformer, which was required to fit different features across these dimensions. Simultaneously, the LTransformer was trained to fit multidimensional features, and its final fitting performance (at the 30th epoch) was better than that of other baseline models.

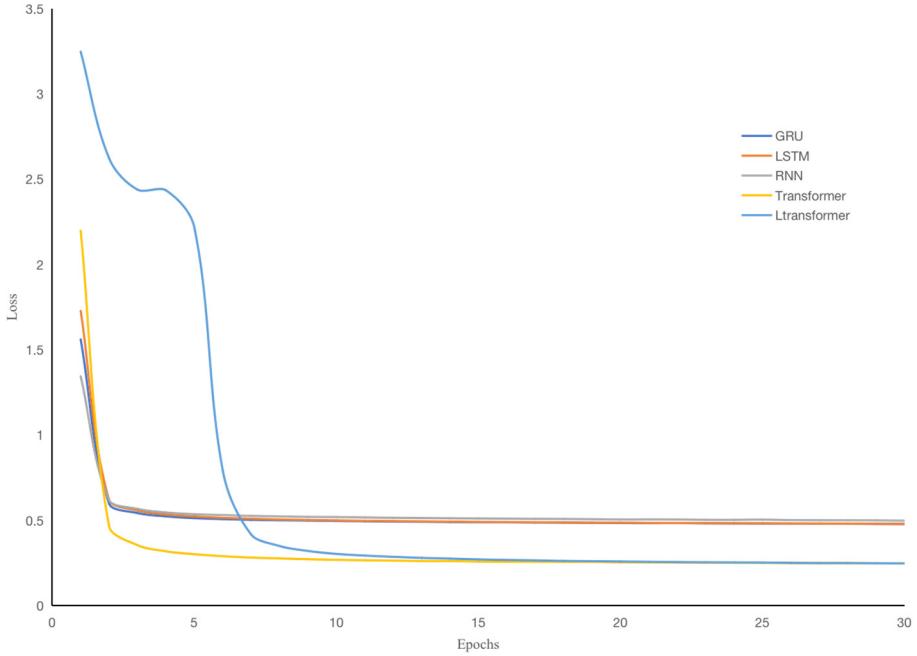

Figure 6. Comparison of losses of each algorithm.

4.5. Resource Consumption

4.5.1. Time Consumption

This subsection primarily discusses the feasibility of using the LTransformer in real-time prediction. Relevant experiments were conducted to test the time required for prediction with trajectory data volumes of 1, 10, 100, and 1000 in three different samples. The specific results are shown in Table 6.

Table 6. Time consumption for different numbers of trajectories.

Sample	Trajectory Data Volumes			
	1	**10**	**100**	**1000**
Sample 1	0.0125	0.0875	0.8888	8.4802
Sample 2	0.0114	0.0905	0.8552	8.4952
Sample 3	0.0123	0.0878	0.8731	8.4910

It can be observed that as the number of trajectories increases, the rate of time consumption tends towards 0.008 s per trajectory. Compared to processing computational tasks, the time required for trajectory prediction is almost negligible. Therefore, the LTransformer essentially meets the requirements for real-time prediction.

4.5.2. Memory Consumption

Memory consumption is one of the key resources for machine learning model training and inference. Machine learning models deployed on edge servers should make efficient use of memory to enhance model efficiency and performance. Therefore, we compared the memory usage of the LTransformer and other baseline algorithms when predicting 80,000 trajectories. It is worth noting that the LTransformer has similar memory usage to the Transformer. Therefore, no comparison was made in terms of memory between the LTransformer and the Transformer. In addition, we compared the memory consumption in three different samples. The specific data are shown in Table 7.

Table 7. Memory consumption in different models.

Sample	Memory Consumption in Different Models			
	GRU	LSTM	RNN	LTransformer
Sample 1	5.5033	5.5063	5.5036	3.7122
Sample 2	5.5069	5.5115	5.5032	3.7133
Sample 3	5.5142	5.5253	5.5015	3.7228

It can be observed that LTransformer also has an advantage in terms of memory resource consumption. This may be attributed to the more advanced encoding techniques adopted by the Transformer model when processing trajectory sequence data, which compresses the data storage space. This allows it to be compatible with a greater number of edge servers.

4.6. Parameter Adjustment

The hyperparameters of the LTransformer are parameters whose values control the deep learning process and determine the values of parameters that the algorithm ends up learning. Different hyperparameters of the LTransformer were analyzed in the experiment. We tested the accuracy of the model by adjusting the value of one parameter, while keeping the training set and other parameters constant. Figure 7 shows the accuracy of the LTransformer model with different dimensions.

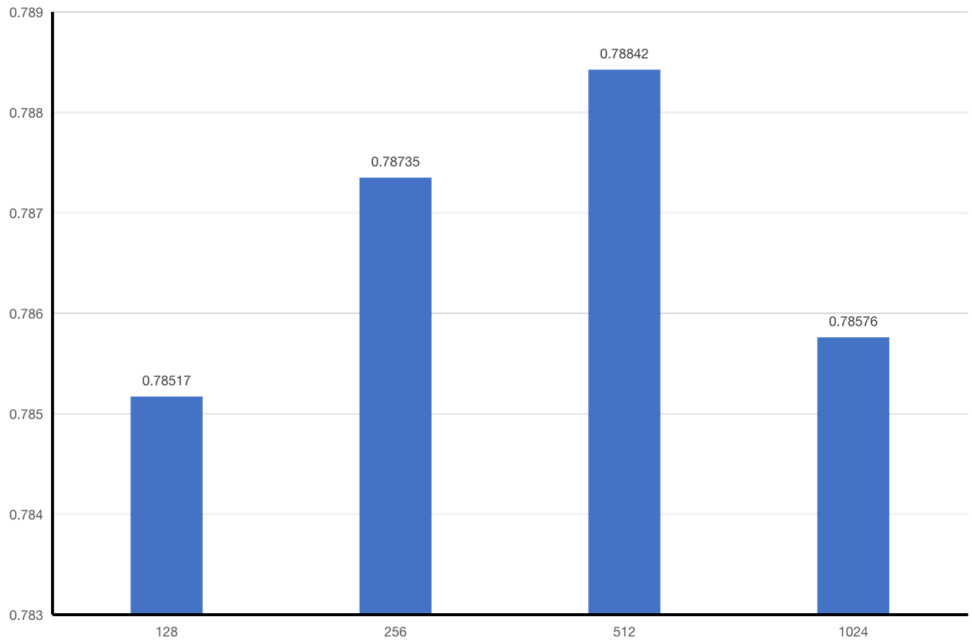

Figure 7. Comparison of accuracy of the LTransformer with different dimensions.

First, experiments were conducted to compare and analyze the embedded dimensions of the LTransformer. The results show that the LTransformer had the best prediction performance with 512 dimensions. This is because the LTransformer goes through pre-training, which reduces the number of dimensions to be fitted. Meanwhile, all of the embeddings are summed together instead of going through concatenation, which also reduces the number of dimensions.

In addition, the experiment also tested the other hyperparameters, and finally set the other hyperparameters to $N = 6$, $p = 16$, and $q = 4096$, for which the LTransformer had the best prediction performance.

Furthermore, the experiment tested the prediction time of the LTransformer for a single trajectory datum, which only requires 0.008 s. Therefore, the experimental results

indicate that the LTransformer achieves a higher accuracy compared to other baseline algorithms and meets the requirement of real-time prediction, making it more suitable for deployment in VEC. Meanwhile, the experiments also demonstrated that the LTransformer can be applied in traffic control, in-vehicle services, and other VEC applications.

5. Conclusions

This paper proposes a task offloading optimization approach in VEC which enables the edge servers to deploy deep learning methods for predicting vehicle trajectories and optimizing task offloading strategies.

Considering the trajectory prediction approach, this paper introduces the LTransformer, which has four modules. In the pre-training, two-dimensional space containing latitude–longitude information is embedded into multidimensional space. The input module integrates geographic, sequential, and temporal information. The encoding–decoding module incorporates the encoder and decoder in the Transformer to train features of multidimensional data. In the output module, the error correction mechanism is employed to remove certain error results. In the experiment, a comparison was made with four commonly used sequential deep learning methods. The experimental results demonstrate that the LTransformer achieves more accurate vehicle trajectories, making it suitable for VEC. Meanwhile, in VEC, the framework can be applied in traffic control and in-vehicle services.

In the future, we will combine other technologies [27] to further optimize the task offloading process in VEC. In trajectory prediction, we will incorporate additional dimensions and behavioral features [28] to enhance the deep learning approach, aiming to achieve higher accuracy and efficiency in VEC. Moreover, privacy and security measures also need to be taken into further consideration.

Author Contributions: Conceptualization, Y.Y. and R.Y.; methodology, Y.Y. and R.Y.; software, Y.Y.; validation, Y.Y., R.Y. and Y.G.; formal analysis, Y.Y.; investigation, Y.Y., R.Y. and Y.G; resources, Y.G.; data curation, Y.Y. and R.Y.; writing—original draft preparation, Y.Y.; writing—review and editing, Y.Y., R.Y. and Y.G.; visualization, Y.Y. and R.Y.; supervision, R.Y. and Y.G.; project administration, Y.Y.; funding acquisition, Y.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Double First-Class Innovation Research Project for People's Public Security University of China, grant number (2023SYL07).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author Y.G. upon reasonable request.

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

References

1. 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]
2. Luo, Q.; Hu, S.; Li, C.; Li, G.; Shi, W. Resource scheduling in edge computing: A survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2131–2165. [CrossRef]
3. Zhan, W.; Luo, C.; Min, G.; Wang, C.; Zhu, Q.; Duan, H. Mobility-aware multi-user offloading optimization for mobile edge computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 3341–3356. [CrossRef]
4. Zhang, K.; Mao, Y.; Leng, S.; He, Y.; Zhang, Y. Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading. *IEEE Veh. Technol. Mag.* **2017**, *12*, 36–44. [CrossRef]
5. Saeik, F.; Avgeris, M.; Spatharakis, D.; Santi, N.; Dechouniotis, D.; Violos, J.; Leivadeas, A.; Athanasopoulos, N.; Mitton, N.; Papavassiliou, S. Task offloading in Edge and Cloud Computing: A survey on mathematical. *Comput. Netw.* **2021**, *195*, 108177. [CrossRef]
6. Yang, S.R.; Su, Y.J.; Chang, Y.Y.; Hung, H.N. Short-term traffic prediction for edge computing-enhanced autonomous and connected cars. *IEEE Trans. Veh. Technol.* **2019**, *68*, 3140–3153. [CrossRef]

7. Alahi, A.; Goel, K.; Ramanathan, V.; Robicquet, A.; Fei-Fei, L.; Savarese, S. Social lstm: Human trajectory prediction in crowded spaces. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 961–971.
8. Han, P.; Wang, W.; Shi, Q.; Yang, J. Real-time short-term trajectory prediction based on GRU neural network. In Proceedings of the 38th Digital Avionics Systems Conference (DASC), San Diego, CA, USA, 8–12 September 2019; pp. 1–8.
9. Huang, M.; Zhu, M.; Xiao, Y.; Liu, Y. Bayonet-corpus: A trajectory prediction method based on bayonet context and bidirectional GRU. *Digit. Commun. Netw.* **2021**, *7*, 72–81. [CrossRef]
10. Amichi, L.; Viana, A.C.; Crovella, M.; Loureiro, A.A. From movement purpose to perceptive spatial mobility prediction. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems, Beijing, China, 2–5 November 2021; pp. 500–511.
11. Monreale, A.; Pinelli, F.; Trasarti, R.; Giannotti, F. Wherenext: A location predictor on trajectory pattern mining. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; pp. 637–646.
12. Zhenjiang, D.; Jia, D.; Xiaohui, J.; Yongli, W. RTMatch: Real-time location prediction based on trajectory pattern matching. In *Database Systems for Advanced Applications, Proceedings of the DASFAA 2017 International Workshops: BDMS, BDQM, SeCoP, and DMMOOC, Suzhou, China, 27–30 March 2017*; Springer: Cham, Switzerland, 2017; pp. 103–117.
13. Zeng, J.; He, X.; Tang, H.; Wen, J. A next location predicting approach based on a recurrent neural network and self-attention. In *Collaborative Computing: Networking, Applications and Worksharing, Proceedings of the 15th EAI International Conference, London, UK, 19–22 August 2019*; Springer: Cham, Switzerland, 2019; pp. 309–322.
14. Feng, J.; Li, Y.; Zhang, C.; Sun, F.; Meng, F.; Guo, A.; Jin, D. Deepmove: Predicting human mobility with attentional recurrent networks. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018; pp. 1459–1468.
15. Liu, T.; Liao, J.; Wu, Z.; Wang, Y.; Wang, J. A geographical-temporal awareness hierarchical attention network for next point-of-interest recommendation. In Proceedings of the 2019 on International Conference on Multimedia Retrieval, Ottawa, ON, Canada, 10–13 June 2019; pp. 7–15.
16. Amirloo, E.; Rasouli, A.; Lakner, P.; Rohani, M.; Luo, J. Latentformer: Multi-agent transformer-based interaction modeling and trajectory prediction. *arXiv* **2022**, arXiv:2203.01880.
17. Yan, J.; Peng, Z.; Yin, H.; Wang, J.; Wang, X.; Shen, Y.; Stechele, W.; Cremers, D. Trajectory prediction for intelligent vehicles using spatial-attention mechanism. *IET Intell. Transp. Syst.* **2020**, *14*, 1855–1863. [CrossRef]
18. Yu, C.; Ma, X.; Ren, J.; Zhao, H.; Yi, S. Spatio-temporal graph transformer networks for pedestrian trajectory prediction. In Proceedings of the Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, 23–28 August 2020; pp. 507–523.
19. Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q.V.; Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. In Proceedings of the 57th Annual Meeting of the Association-for-Computational-Linguistics (ACL), Florence, Italy, 28 July–2 August 2019; pp. 2978–2988.
20. Wang, S.; Li, B.Z.; Khabsa, M.; Fang, H.; Ma, H. Linformer: Self-attention with linear complexity. *arXiv* **2020**, arXiv:2006.04768.
21. Kitaev, N.; Kaiser, Ł.; Levskaya, A. Reformer: The efficient transformer. *arXiv* **2020**, arXiv:2001.04451.
22. Kong, X.; Xing, W.; Wei, X.; Bao, P.; Zhang, J.; Lu, W. STGAT: Spatial-temporal graph attention networks for traffic flow forecasting. *IEEE Access* **2020**, *8*, 134363–134372. [CrossRef]
23. Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; Zhang, W. Informer: Beyond efficient transformer for long sequence time-series forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 2–9 February 2021; pp. 11106–11115.
24. Bahra, N.; Pierre, S. RNN-based user trajectory prediction using a preprocessed dataset. In Proceedings of the 2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Thessaloniki, Greece, 12–14 October 2020; pp. 1–6.
25. Xiao, H.; Wang, C.; Li, Z.; Wang, R.; Bo, C.; Sotelo, M.A.; Xu, Y. UB-LSTM: A trajectory prediction method combined with vehicle behavior recognition. *J. Adv. Transp.* **2020**, *2020*, 8859689. [CrossRef]
26. Zhao, J.; Li, X.; Xue, Q.; Zhang, W. Spatial-channel transformer network for trajectory prediction on the traffic scenes. *arXiv* **2021**, arXiv:2101.11472.
27. Zhao, H.; You, J.; Wang, Y.; Zhao, X. Offloading Strategy of Multi-Service and Multi-User Edge Computing in Internet of Vehicles. *Appl. Sci.* **2023**, *13*, 6079. [CrossRef]
28. Peng, B.; Li, T.; Chen, Y. DRL-Based Dependent Task Offloading Strategies with Multi-Server Collaboration in Multi-Access Edge Computing. *Appl. Sci.* **2023**, *13*, 191. [CrossRef]