GRIP: Graph-based Interaction-aware Trajectory Prediction

Xin Li, Xiaowen Ying, Mooi Choo Chuah
Department of Computer Science and Engineering, Lehigh University
{xil915, xiy517}@lehigh.edu, chuah@cse.lehigh.edu

Abstract— **Nowadays, autonomous driving cars have become commercially available. However, the safety of a self-driving car is still a challenging problem that has not been well studied. Motion prediction is one of the core functions of an autonomous driving car. In this paper, we propose a novel scheme called GRIP which is designed to predict trajectories for traffic agents around an autonomous car efficiently. GRIP uses a graph to represent the interactions of close objects, applies several graph convolutional blocks to extract features, and subsequently uses an encoder-decoder long short-term memory (LSTM) model to make predictions. The experimental results on two well-known public datasets show that our proposed model improves the prediction accuracy of the state-of-the-art solution by 30%. The prediction error of GRIP is one meter shorter than existing schemes. Such an improvement can help autonomous driving cars avoid many traffic accidents. In addition, the proposed GRIP runs 5x faster than the state-of-the-art schemes.**

I. Introduction

In the past few years, thanks to technology advancement in the fields of computer vision, sensor signal processing, and hardware designing, etc., autonomous driving has gone from “may be possible” to “commercially available”. However, two traffic accidents caused by autonomous driving cars from Tesla and Uber in 2018 raised people’s concern about the safety of self-driving vehicles. Thus, it is critically important to improve the performance of the intelligent algorithms running on autonomous driving cars. Prediction of the future trajectories of the surrounding objects, e.g., vehicles, pedestrians, bicycles, etc., is one of such intelligent algorithms. Experts argue that we can avoid such a traffic accident if each autonomous driving car involved could precisely predict the locations of its surrounding objects.

Nevertheless, accurately predicting the motion of surrounding objects is an extremely challenging task, considering that there are so many factors that affect the future trajectory of an object. Prior works [1], [2], [3], [4], [5] proposed to predict future locations by recognizing maneuver (change lanes, brake, or keep going, etc.). However, these methods fail to predict the positions of objects accurately when they recognize the type of maneuver wrongly. Such an issue happens when a scheme makes a prediction only based on sensors like GPS that misses visual clues, e.g., turn signals. Then, Karasev et al. [6] proposed to predict the motion of pedestrians by modeling their behaviors as jump-Markov processes. Even though they claimed that they could predict the route of an observed pedestrian, the proposed method requires a semantic map and one or several goals of the pedestrian, which is not useful in the autonomous driving scenario because an autonomous driving car cannot know the destination of a pedestrian (or other objects) in advance. Bhattacharyya et al. [7] tried to predict the bounding boxes of objects in RGB camera frames by predicting future vehicle odometry sequence. Yet, the predicted bounding boxes in RGB frames still need to be mapped to the coordinate system of the self-driving car. Otherwise, the self-driving car cannot make a correct response to these predicted locations.

Besides, almost all of the schemes we discussed above only consider the state of one predicted object, i.e., few of them take the states of surrounding objects into account. We argue that the motion states of surrounding objects are crucial for motion prediction especially in the field of autonomous driving.

Thus, in this paper, we propose a robust and efficient object trajectory prediction scheme for autonomous driving cars, namely GRIP, that can infer future locations of nearby objects simultaneously and is trainable end-to-end.

In summary, our contributions of this paper include:

- A robust and efficient object trajectory prediction scheme to precisely predict future locations of objects surrounding an autonomous driving car.
- The proposed scheme considers the impact of inter-object interactions on the motion.
- Extensive evaluation using two popular traffic datasets show that our scheme achieves higher accuracy and runs an order of magnitude faster than existing schemes.

The rest of this paper is organized as follows. In Section II, we briefly discuss related work followed by the problem formulation in Section III. In Section IV, we describe our proposed object trajectory prediction scheme and implementation details. We report our experimental results in Section V. Finally, we conclude this paper in Section VI.

II. Related work

Much work has been done on object trajectory prediction, but few of them considers the impact of nearby objects. In recent years, researchers have realized this issue and started exploring possible solutions. Thus, here we merely summarize the more recent works that take inter-object interactions into account.

Luo et al. proposed a convolutional network for fast object detection, tracking and motion forecasting in [8]. Their model takes a series of bird’s eye view LiDAR data as input and processes 3D convolutions across space and time. Then, they add two extra branches of convolutional layers, one of them calculates the probability of being a vehicle at a given location and another predicts the bounding box over the current frame as well as several frames in the future. They argue that such a structure is able to forecast motion because the model takes multiple frames as input and can learn velocity and acceleration features. However, the forecasting branch simply takes the 3D convolutional feature map as an input, so visual features of all objects are represented in the same feature map. In this case, the model will lose track of objects and hence cannot perform well in a scene that consists of crowded objects.

In addition, Deo et al. [9], [10] proposed a unified framework for surrounding vehicles' maneuver classification and motion prediction on freeways. They first use an LSTM model to represent the track histories and relative positions of all observed cars (the one being predicted and its nearby vehicles) as a context vector. Then, they use this context vector to do maneuver classification and use an LSTM to predict the vehicle's future position. Considering that the LSTM model fails to capture the interdependencies of the motion of all cars in the scene, they then enhance their scheme by adding convolutional social pooling layers in [11]. Such a model indeed improves the accuracy of future motion prediction, because it has access to the motion states of surrounding objects and their spatial relationships. Although all of these models take the trajectory histories of all objects in the scene as their inputs, they merely predict the trajectory of one specific car (the one in the middle position) each time. Hence, these existing approaches require intensive computation power if they want to predict trajectories of all surrounding objects which is highly inefficient especially for autonomous driving cars scenarios. In addition, these schemes are maneuver based, so wrong classification of the maneuver type will negatively impact the trajectory prediction.

III. Problem Formulation

Before introducing our proposed scheme, we would like to formulate the trajectory prediction problem as one which estimates the future positions of all objects in a scene based on their trajectory histories. Specifically, the inputs X of our model are trajectory histories (over t_h time steps) of all observed objects:

$$X = [p^{(1)}, p^{(2)} \cdots, p^{(t_h)}] \tag{1}$$

where,

$$p^{(t)} = [x_0^{(t)}, y_0^{(t)}, x_1^{(t)}, y_1^{(t)}, \cdots, x_n^{(t)}, y_n^{(t)}] \tag{2}$$

are the co-ordinates of all observed objects at time t, and n is the number of observed objects. This format is the same as what Deo et al. defined in [9] and [11]. Global coordinates are used here. Using relative measurement in the ego-vehicle-based coordinate system will improve the prediction accuracy, but will be left for future work.

Considering that we feed track histories of all observed objects into the model, we argue that it makes more sense to predict future positions for all of them simultaneously for an autonomous driving car. Thus, instead of only predicting the position of one particular object as done in [9] and [11], the outputs Y of our proposed model are the predicted positions of all observed objects from time step $t_h + 1$ to $t_h + t_f$ in the future:

$$Y = [p^{(t_h+1)}, p^{(t_h+2)}, \cdots, p^{(t_h+t_f)}] \tag{3}$$

where $p^{(t)}$ is the same as equation (2) and t_f is the predicted horizon.

IV. Proposed Scheme

To solve the limitations of existing approaches, we propose a novel deep learning model for object trajectory prediction in this section. Our model, illustrated in Figure 1, consists of three components: (1) Input Preprocessing Model, (2) Graph Convolutional Model, and (3) Trajectory Prediction Model.

A. Input Preprocessing Model

1) Input Representation:

Before feeding the trajectory data of objects into our model, we convert the raw data into a specific format for subsequent efficient computation. Assuming that n objects in a traffic scene were observed in the past t_h time steps, we represent such information in a 3D array F_{input} with a size of $(n \times t_h \times c)$ (as shown in Figure 1). In this paper, we set $c = 2$ to indicate x and y coordinates of an object. All coordinates are normalized to the range of $(-1, 1)$.

2) Graph Construction:

Considering that, in the autonomous driving application scenario, the motion of an object is profoundly impacted by the movements of its surrounding objects. This is highly similar to people's behaviors on a social network (one person is usually to be impacted by his/her friends). This inspires us to represent the inter-object interaction using an undirected graph $G = \{V, E\}$ as what researchers have done for a social network.

In this graph, each node in node set V corresponds to an object in a traffic scene. Considering that each object may have different states at different time steps, the node set V is defined as $V = \{v_{it} | i = 1, \cdots, n, t = 1, \cdots, t_h\}$, where n is the number of observed objects in a scene, and t_h is the observed time steps. The feature vector v_{it} on a node is the coordinate of ith object at time t.

At each time step t, objects that have interactions should be connected with edges. In the autonomous driving application scenario, such an interaction happens when two objects are close to each other. Thus, the edge set E is composed of two parts: (1) The first part describes the interaction information between two objects in spatial space at time t. We call it a "spatial edge" and denote it as $E_S = \{v_{it}v_{jt} | (i, j \in D)\}$, where D is a set in which objects are close to each other. In this paper, we define that two objects are close if their distance is less than a threshold of D_{close}. In Figure 1, we demonstrate this concept on "Raw Data" using two blue circles with a radius of D_{close}. All objects within the blue circle are regarded as close to the one located in the middle of the circle. Thus, the top object has three close neighbors, and the lower one only has

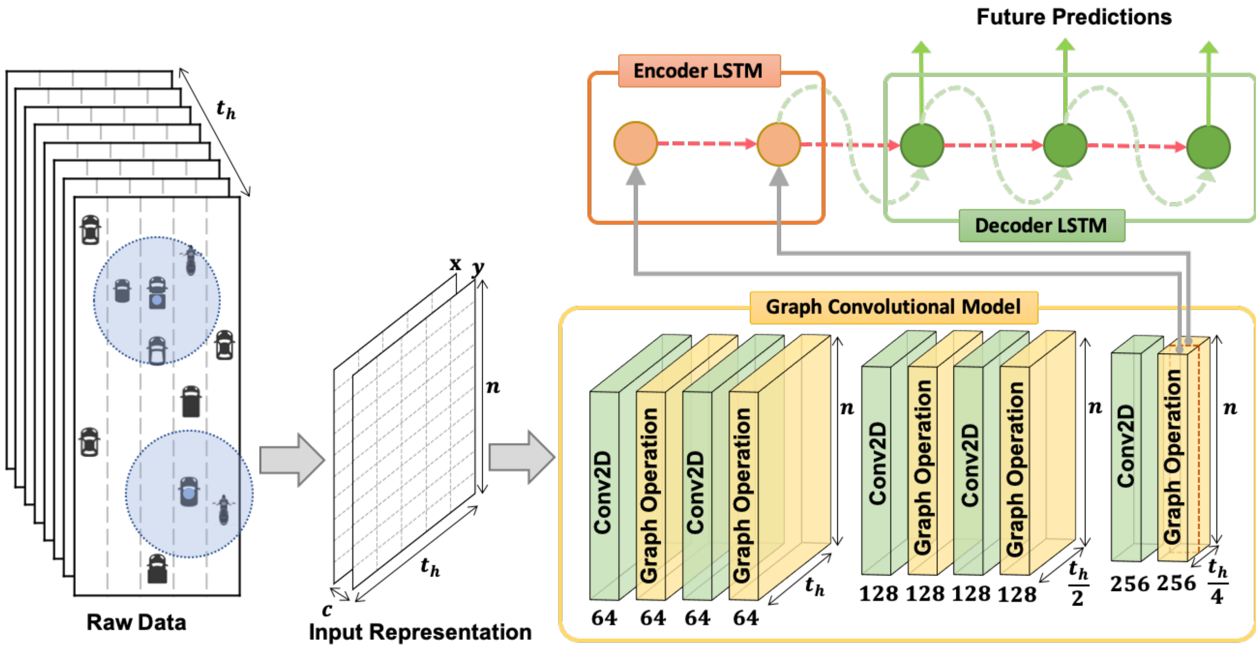


Fig. 1: The architecture of the proposed Scheme.

one neighbor. (2) The second part is the inter-frame edges, which represents the historical information frame by frame in temporal space. Each observed object in one time-step is connected to itself in another time-step via the temporal edge and such edges are denoted as $E_F = \{v_{it}v_{i(t+1)}\}$. Thus, all edges in E_F of one particular object represent its trajectory over time steps.

To make the computation more efficient, we represent this graph using an adjacency matrix $A = \{A_0, A_1\}$, where A_0 is an identity matrix I representing self-connections in temporal space, and A_1 is a spatial connection adjacency matrix. Thus, at any time t,

$$A_0[i][j](or A_1[i][j]) = \begin{cases} 1, \text{if edge } \langle v_{it}, v_{jt} \rangle \in E \\ 0, \text{otherwise} \end{cases} \tag{4}$$

Both A_0 and A_1 have a size of $(n \times n)$, where n equals to the number of observed objects in a scene.

B. Graph Convolutional Model

The Graph Convolutional Model consists of several convolutional layers as well as graph operations. These convolutional layers are designed to capture useful temporal features, e.g., motion pattern of one object, and graph operations to handle the inter-object interaction in spatial space. Thus, as shown in Figure 1 (5 convolutional layers and 5 graph operation layers are illustrated), one graph operation layer is added to the end of each convolutional layer in this Graph Convolutional Model to process the input data temporally and spatially alternatively.

*1) **Convolutional Layer**:* Given a preprocessed input data $F_{input} := \mathbb{R}^{N \times T \times C}$, the model first passes it through a convolutional layer to compute convolutional feature maps f_{conv}. We set the kernel size of convolutional layers to (1×3) to force them to process the data along the temporal dimension (second dimension). Appropriate paddings and strides are added to make sure that each layer has an output feature map with expected size.

*2) **Graph Operation Layer**:* Then, we feed the generated convolutional feature maps f_{conv} to a graph operation layer to take the interactions of surrounding objects into account. The graph operation involves multiplying normalized version of matrix A with f_{conv} using the following formula:

$$f_{graph} = \sum_{j=0}^{1} \Lambda_j^{-\frac{1}{2}} A_j \Lambda_j^{-\frac{1}{2}} f_{conv} \tag{5}$$

where A is the adjacency matrix we constructed in subsection IV-A.2 and Λ_j is computed as:

$$\Lambda_j^{ii} = \sum_k (A_j^{ik}) + \alpha \tag{6}$$

$\Lambda^{-\frac{1}{2}} A \Lambda^{-\frac{1}{2}}$ is a normalized version of A, which is used to make sure that the value range of feature maps remain unchanged after performing the graph operations.We set $\alpha = 0.001$ to avoid empty rows in A_j.

C. Trajectory Prediction Model

This model predicts the future trajectories for all observed objects in a scene. The Trajectory Prediction Model is an LSTM encoder-decoder network that takes the computed output of the Graph Convolutional Model f_{graph} as input. The output of the graph convolutional model is fed into the encoder LSTM at each time step. Then, the hidden feature of the encoder LSTM, as well as coordinates of objects at the previous time step, are fed into a decoder LSTM to predict the position coordinates at the current time step.

Such a decoding process is repeated several times until the model predicts positions for all expected time steps (t_f) in the future.

D. Implementation Details

Our scheme is implemented using Python Programming Language and PyTorch Library [12]. We report the implementation details of our scheme and the settings of important parameters as follows.

Input Preprocessing Model: In this paper, we process a traffic scene within 180 feet (± 90 feet). All objects within this region will be observed and predicted in the future. While constructing the graph, we consider two objects are close if their distance is less than 25 feet ($D_{close} = 25$). Thus, any pair of objects within 25 feet are connected using a spatial edge, $e_s \in E_S$. Please refer to our ablation study in section V-C for more details.

Graph Convolutional Model: The Graph Convolutional Model consists of 10 convolutional layers, denoted as $\{conv2d\_i | i = 1, 2, \cdots, 10\}$. All Conv2D layers have a convolutional kernel with a size of (1×3). Among all of these 10 Conv2D layers, we set $stride = 2$ for $conv2d\_5$ and $conv2d\_8$ to achieve some pooling effects, but use $stride = 1$ for remaining layers. The output channel of the first Conv2D is set to 64. We double the number of output channels when $stride = 2$. Thus, the final output of the Graph Convolution Model has 256 channels.

Each of these convolutional layers is followed by a graph operation layer. Graph operation layers do not change the size of features, and they share the same adjacency matrix. To avoid overfitting, we randomly dropout features (0.5 probability) after each graph operation.

Trajectory Prediction Model: Both the encoder and decoder of this prediction model are a two-layer LSTM. We set the number of hidden units of these two LSTMs equals to the output dimension ($2 \times n$, where n is the number of objects and 2 is the x, y coordinates). The input of the encoder has 256 channels that are the same as the output of the Graph Convolutional Model. We add a $tanh$ activation function to the output layers of both LSTMs to rescale the output to range of (-1, 1).

Optimization: We train our model as a regression task at each time. The overall loss can be computed as:

$$Loss = \frac{1}{t_f} \sum_{t=1}^{t_f} loss^t \tag{7}$$

$$= \frac{1}{t_f} \sum_{t=1}^{t_f} \left\| Y_{pred}^t - Y_{GT}^t \right\|^2 \tag{8}$$

where t_f is the time step in the future (in Figure 1, $t_f = 3$), $loss^t$ is the loss at time t, Y_{pred} and Y_{GT} are predicted positions and ground truth respectively. The model is trained to minimize the $Loss$.

Training Process: We train the model using Stochastic Gradient Descent (SGD) optimizer with 0.001 starting learning rate. The learning rate is reduced by multiplying with 0.1 once per 5 epochs until the loss becomes converged. As done in [11], we set $batch\_size = 128$ during training.

V. Experiments

We run our scheme on a desktop running Ubuntu 16.04 with 4.0GHz Intel Core i7 CPU, 32GB Memory, and a NVIDIA Titan Xp Graphics Card.

A. Datasets

We evaluate our scheme on two well known trajectory prediction datasets: NGSIM I-80 [13] and US-101 [14]. Both datasets were captured at 10 Hz over 45 minutes and segmented into 15 minutes of mild, moderate and congested traffic conditions. These two datasets consist of trajectories of vehicles on real freeway traffic. Coordinates of cars in a local coordinate system are provided.

We follow Deo et al. [9], [10], [11] to split these two datasets into training and testing sets. One-fourth of the data from each of the three subsets (mild, moderate, and congested traffic conditions) are selected for testing. Each trajectory is segmented into 8 seconds clips that the first 3 seconds are used as observed track history and the remaining 5 seconds are the prediction ground truth. To make a fair comparison, we also do the same downsampling for each segment by a factor 2 as Deo et al. did, i.e. 5 frames per second. The code for dataset segmentation can be downloaded from their Github [1].

B. Metrics

We use the same experimental settings and evaluation metrics as [11] and [15]. In this paper, we report our results in terms of the root of the mean squared error (RMSE) of the predicted trajectories in the future (5 seconds horizons). The RMSE at time t can be computed as follows:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (Y_{pred}^t[i] - Y_{GT}^t[i])^2} \tag{9}$$

where n is the number of observed (predicted) objects, Y_{pred}^t and Y_{GT}^t are predicted results and ground truth at time t correspondingly.

C. Ablation Study

In this subsection, we do two ablation studies about our scheme:

(1) We defined a threshold D_{close} in section IV-A.2. Two objects within D_{close} range are regarded as close to each other. We first explore how this threshold impacts the performance of our model. In Figure 2, we compare results when D_{close} is set to different values. One can see that the prediction error when $D_{close} = 0$ (when none of the surrounding objects are considered, blue bars in Figure 2) is higher than the results when $D_{close} > 0$ (taking nearby objects into account). Thus, considering the surrounding object indeed helps our model make a better prediction.

[1] https://github.com/nachiket92/conv-social-pooling

TABLE I: Root Mean Square Error (RMSE) for trajectory prediction on NGSIM I-80 and US-101 datasets. Data are converted into the meter unit. All results except ours are extracted from [11]. The smaller the value, the better.

Prediction Horizon (s)	CV	V-LSTM	C-VGMM + VIM [10]	GAIL-GRU [15]	CS-LSTM(M) [11]	CS-LSTM [11]	GRIP (△CS-LSTM)	GRIP (ALL)
1	0.73	0.68	0.66	0.69	0.62	0.61	**0.37** (40%↑ -0.24)	0.64
2	1.78	1.65	1.56	1.51	1.29	1.27	**0.86** (32%↑ -0.41)	1.13
3	3.13	2.91	2.75	2.55	2.13	2.09	**1.45** (31%↑ -0.64)	1.80
4	4.78	4.46	4.24	3.65	3.20	3.10	**2.21** (29%↑ -0.89)	2.62
5	6.68	6.27	5.99	4.71	4.52	4.37	**3.16** (28%↑ -1.21)	3.60

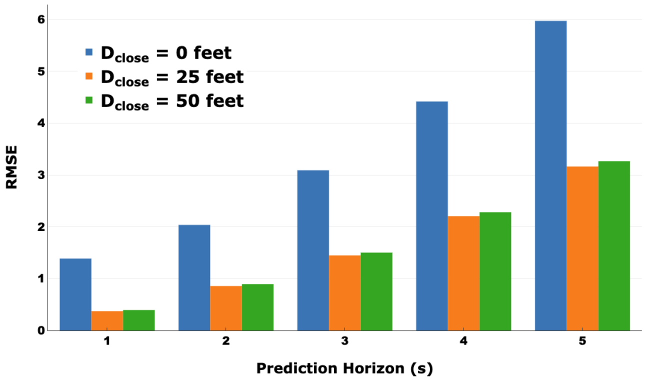


Fig. 2: Comparison among various D_{close} values.

Also, we notice that the prediction error increases when D_{close} increases from 25 feet (orange bars) to 50 feet (green bars). This is because more objects are used to predict the motion of an object with larger $Dclose$. In real life, a traffic agent is more likely to be only impacted by its closest objects. Thus, considering too many surrounding objects does not help to improve the prediction accuracy. Based on this observation, in this paper, we set $D_{close} = 25$ feet as our default setting unless specified otherwise.

(2) Given an input stream consisting of observed objects' past trajectories, our model is able to predict future trajectories for all observed objects. Thus, in Figure 3, we report the prediction error for objects at different locations, e.g., -60 or -45 feet, within the observed area. In Figure 3, traffic agents are moving from location -90 to location 90 (left to right).

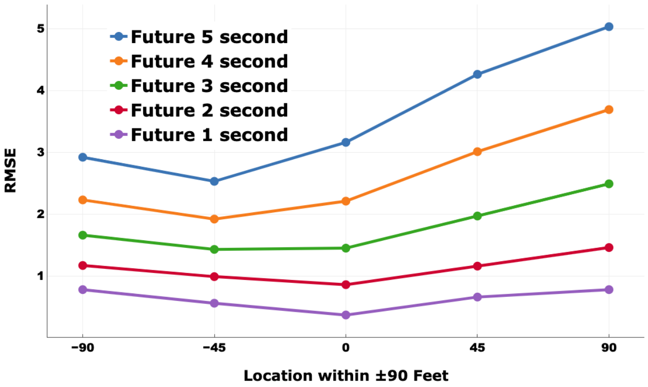


Fig. 3: Prediction error at different locations.

First, one may notice that the prediction error decreases from location -90 to -45, and then increases after -45. Such an observation is obvious on the top 3 curves ("Future 5/4/3 second"). This is impacted by the clue information from surrounding objects. Because objects are moving from left to right in Figure 3, so objects located at 90 can only observe objects behind them, while objects at -90 can only see objects in front of them. Thus, prediction error at -90 is lower than the error at 90 concludes that front objects are more important than behind objects for our trajectory prediction model. This is also the reason why prediction error increases after -45 (less and less front objects are observed from left to right).

In addition, considering that predicting the motion of an object in far future is difficult. Thus, in Figure 3, the error of a long time prediction is higher than a shorter time prediction (The "Future 5 second" curve is above the "Future 1 second" curve).

D. *Comparison Results*

In this subsection, we compare our proposed scheme to the following baselines (as done in [11]) and some existing solutions:

- Constant Velocity (CV): This is a baseline that only uses a constant velocity Kalman filter to predict trajectories in the future.
- Vanilla LSTM (V-LSTM): A baseline that feeds a tack history of the predicted object to an LSTM model to predict a distribution of its future position.
- C-VGMM + VIM: In [10], Deo et al. propose a maneuver based variational Gaussian mixture model with a Markov random field based vehicle interaction module.
- GAIL-GRU: Kuefler et al. [15] use a generative adversarial imitation learning model for vehicle trajectory prediction. However, they use ground truth data for surrounding vehicles as input during prediction phase.
- CS-LSTM (M): This is the model that an LSTM model with convolutional social pooling layers proposed by Deo et al. in [11]. A maneuver classier is included.
- CS-LSTM: A CS-LSTM model without the maneuver classifier described in [11].

Comparison results are reported in Table I. Our model can predict the trajectories for all observed objects simultaneously, while other schemes listed in Table I only predict one specific object (in the middle position) each time. Thus, to make a fair comparison, we compute the RMSE for the same objects as other schemes and report the result in the second column on the right side, "GRIP (△CS-LSTM)", of Table I. Compared to the existing state-of-the-art result (CS-LSTM [11]), our proposed GRIP improves the prediction performance by at least 28%. One may notice that, after 3 seconds in the future, the prediction error of GRIP is a half meter (or longer) shorter than CS-LSTM [11]. We believe that such an improvement can help an autonomous driving car avoid many traffic accidents.

Besides, we also report RMSE results for all predicted objects in the last column, "GRIP (ALL)", of Table I. It is worth highlighting that:

- All schemes in Table 1 take the same data (an object in the middle position and its surrounding objects) as their inputs. Our model predicts all observed objects simultaneously, while others only predict the one in the central location.
- As we discussed the ablation study subsection V-C, objects located at the edge of the observed area, e.g., located at ±90 feet position, do not have enough surrounding objects as input. Thus, the prediction errors of these objects are high, which results in the results in the column of "GRIP (ALL)" are higher than "GRIP".

Even so, our proposed GRIP still achieves better prediction results than all of the other existing solutions.

Then, compared the result of CS-LSTM(M) to CS-LSTM, one can see that CS-LSTM makes slightly better prediction than CS-LSTM(M). This is consistent with our argument mentioned in Section II that a wrong classification of maneuver type has an adverse effect on the trajectory prediction.

E. Computation Time

Computation efficiency is one of the important performance indicators of an algorithm for autonomous driving cars. Thus, we evaluate the computation time of our proposed GRIP and report the results in Table II.

To make a fair comparison, we downloaded the code of CS-LSTM [11] [2] and ran it on our machine to collect its computation time. Both CS-LSTM and GRIP are implemented using PyTorch.

TABLE II: Computation time

Scheme	Predicted #	Time (s) 128 batch	Time (s) 1 batch
CS-LSTM [11]	1000	0.29	35.13
GRIP	1000	**0.05**	**6.33**

From Table II, one can see that, when using 128 batch size, CS-LSTM [11] needs 0.29s to predict trajectories for 1000 objects, while our proposed GRIP only takes 0.05s (5.8x faster). In the autonomous driving application scenario, considering the limited resources, we can only set $batch\_size = 1$, so we report the results in the last column of Table II. It shows that GRIP can still run 5.5 times faster than CS-LSTM [11].

F. Visualization of Prediction Results

In Figure 4, we visualize several prediction results in mild, moderate, and congested traffic conditions (from left to right) using the datasets NGSIM I-80 and US-101. After observing 3 seconds of history trajectories, our model predicts the trajectories over 5 seconds horizon in the future. From Figure 4, one can notice that:

- 1. From Figure 4a to Figure 4c, it is obvious that green-dashed lines (CS-LSTM) are longer than yellow-dashed lines (ours) and farther from the red-dashed lines (ground truth). This proves that when feeding the same history trajectories (all objects in the scene) to models, our proposed GRIP makes a better prediction for the central object than CS-LSTM.
- 2. In Figure 4b, our model precisely predicts the trajectory of the top car even when it is going to change lane in the next 5 seconds. In addition, the car in the left lane is affected by the top car, and our model still successfully predict the trajectory for the car in the left lane.
- 3. Our proposed GRIP can predict all objects in the scene simultaneously, while CS-LSTM can only predict the one located in the middle. Especially, in Figure 4e, we show a prediction result in a scene that involves 15 cars. In this scene, although some cars move slowly (vehicles in the middle lane) while others move faster (cars in the right lane), our proposed GRIP model is able to predict their future trajectories correctly and simultaneously.

Based on these observations from the visualized results, we can conclude that our proposed scheme, GRIP, indeed improves the trajectory prediction performance compared to the existing methods. Even though Figure 4 only shows straight high way scenario, our approach equally works for curved roads.

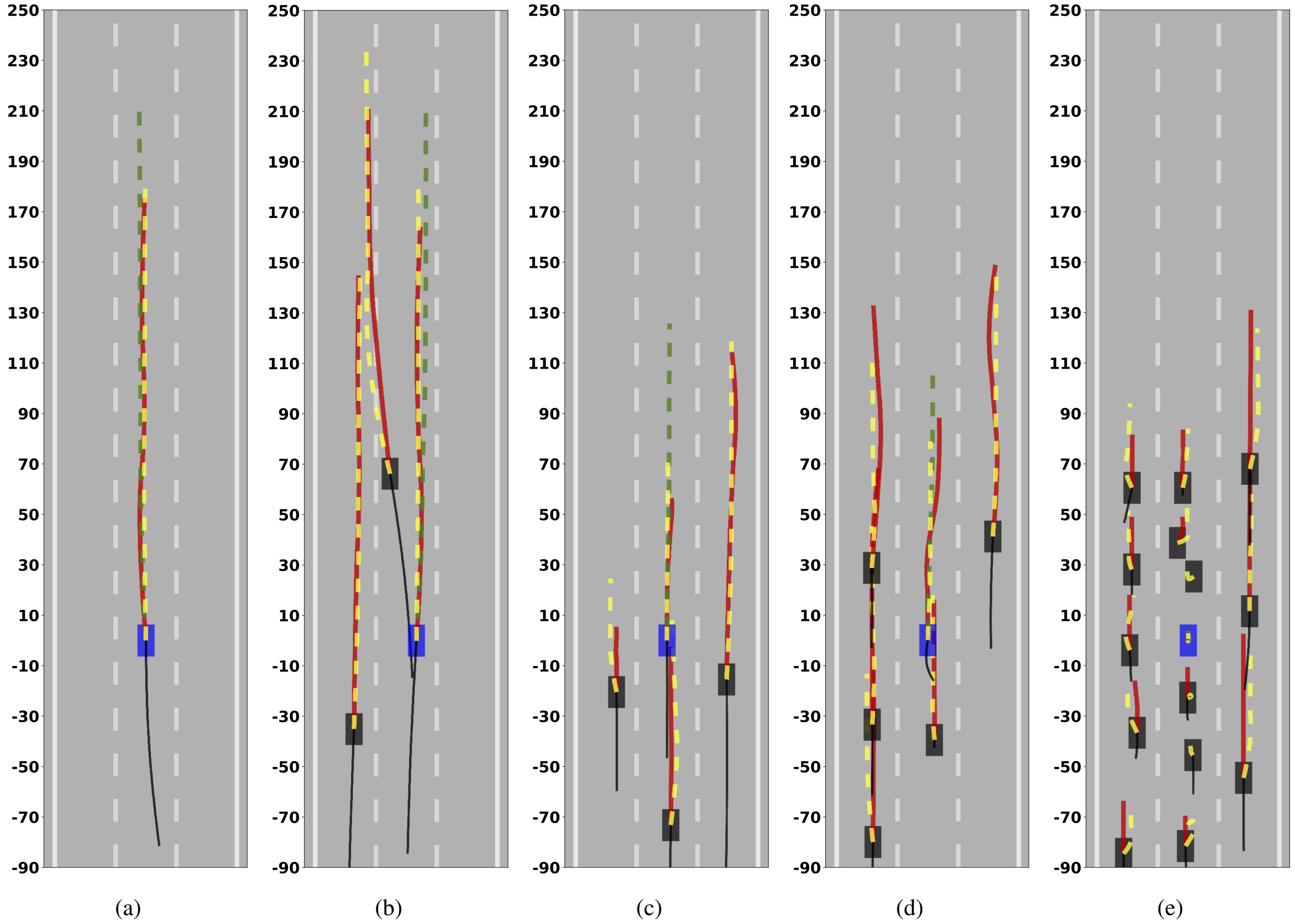
VI. Conclusion

In this paper, we propose a novel scheme (GRIP) for autonomous driving cars to predict object trajectories in the future. The proposed model uses a graph to represent the interaction among all close objects and employs an encoder-decoder LSTM model to make predictions. Unlike some existing solutions that only predict the future trajectory for a single traffic agent each time, GRIP is able to predict trajectories for all observed objects simultaneously. The experimental results on two well-known public datasets show that our proposed model achieves much better prediction results than existing methods and run 5 times faster than the state-of-the-art schemes.

In the near future, we would like to evaluate the proposed model on other datasets, e.g. the Appollo dataset [16], in which data is captured not only on a highway but also from urban areas. Besides, we want to extend the proposed model by adding visual data collected using RGB cameras, etc. to further improve the prediction performance.

Acknowledgement: This work is partially supported by a Qualcomm gift and a GPU donated by NVIDIA.

[2] https://github.com/nachiket92/conv-social-pooling



(a)

(b)

(c)

(d)

(e)

Fig. 4: Visualized Prediction Results. Blue rectangles are the cars located in the middle which is the car that CS-LSTM [11] trys to predict. Black boxes are surrounding cars. Black-solid lines are the observed history, red-dashed lines are the ground truth in the future, yellow-dashed lines are the predicted results (5 seconds) of our GRIP, and the green-dashed lines are the predicted results (5 seconds) of CS-LSTM [11]. Region from −90 to 90 feet are observed areas.

References

[1] R. Toledo-Moreo and M. A. Zamora-Izquierdo, "Imm-based lane-change prediction in highways with low-cost gps/ins," *IEEE Transactions on Intelligent Transportation Systems*, 2009.

[2] A. Houenou, P. Bonnifait, V. Cherfaoui, and W. Yao, "Vehicle trajectory prediction based on motion model and maneuver recognition," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 4363–4369.

[3] M. Schreier, V. Willert, and J. Adamy, "Bayesian, maneuver-based, long-term trajectory prediction and criticality assessment for driver assistance systems," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2014, pp. 334–341.

[4] Q. Tran and J. Firl, "Online maneuver recognition and multimodal trajectory prediction for intersection assistance using non-parametric regression," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 918–923.

[5] J. Schlechtriemen, F. Wirthmueller, A. Wedel, G. Breuel, and K.-D. Kuhnert, "When will it change the lane? a probabilistic regression approach for rarely occurring events," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 1373–1379.

[6] V. Karasev, A. Ayvaci, B. Heisele, and S. Soatto, "Intent-aware long-term prediction of pedestrian motion," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 2543–2549.

[7] A. Bhattacharyya, M. Fritz, and B. Schiele, "Long-term on-board prediction of people in traffic scenes under uncertainty," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4194–4202.

[8] W. Luo, B. Yang, and R. Urtasun, "Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3569–3577.

[9] N. Deo and M. M. Trivedi, "Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1179–1184.

[10] N. Deo, A. Rangesh, and M. M. Trivedi, "How would surround vehicles move? a unified framework for maneuver classification and motion prediction," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 2, pp. 129–140, 2018.

[11] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1468–1476.

[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[13] J. Colyar and J. Halkias, "Us highway 80 dataset," *Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030*, 2007.

[14] ——, "Us highway 101 dataset," *Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030*, 2007.

[15] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating driver behavior with generative adversarial networks," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 204–211.

[16] Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha, "Trafficpredict: Trajectory prediction for heterogeneous traffic-agents," *arXiv preprint arXiv:1811.02146*, 2018.