IEEE Open Journal of
Intelligent Transportation
Systems

Received 26 September 2022; revised 20 November 2022; accepted 30 December 2022. Date of publication 3 January 2023;
date of current version 17 January 2023.

Digital Object Identifier 10.1109/OJITS.2023.3233952

Lane Transformer: A High-Efficiency Trajectory Prediction Model

ZHIBO WANG[1], JIAYU GUO[1], ZHENGMING HU[2], HAIQIANG ZHANG[2], JUNPING ZHANG[3] (Senior Member, IEEE), AND JIAN PU[1] (Member, IEEE)

[1]Institute of Science and Technology for Brain-Inspired Intelligence, Fudan University, Shanghai 200433, China

[2]Autonomous Driving General Algorithm Department, Mogo Auto Intelligence and Telematics Information Technology Company Ltd., Beijing 100013, China

[3]School of Computer Science, Fudan University, Shanghai 200433, China

CORRESPONDING AUTHOR: J. PU (e-mail: jianpu@fudan.edu.cn)

This work was supported in part by the Shanghai Municipal Science and Technology Major Project under Grant 2018SHZDZX01;
in part by ZJ Lab; and in part by the Shanghai Center for Brain Science and Brain-Inspired Technology.

ABSTRACT Trajectory prediction is a crucial step in the pipeline for autonomous driving because it not only improves the planning of future routes, but also ensures vehicle safety. On the basis of deep neural networks, numerous trajectory prediction models have been proposed and have already achieved high performance on public datasets due to the well-designed model structure and complex optimization procedure. However, the majority of these methods overlook the fact that vehicles' limited computing resources can be utilized for online real-time inference. We proposed a *Lane Transformer* to achieve high accuracy and efficiency in trajectory prediction to tackle this problem. On the one hand, inspired by the well-known transformer, we use attention blocks to replace the commonly used Graph Convolution Network (GCN) in trajectory prediction models, thereby drastically reducing the time cost while maintaining the accuracy. In contrast, we construct our prediction model to be compatible with TensorRT, allowing it to be further optimized and easily transformed into a deployment-friendly form of TensorRT. Experiments demonstrate that our model outperforms the baseline LaneGCN model in quantitative prediction accuracy on the Argoverse dataset by a factor of $10\times$ to $25\times$. Our $7ms$ inference time is the fastest among all open source methods currently available. Our code is publicly available at: https://github.com/mmdzb/Lane-Transformer.

INDEX TERMS Trajectory prediction, transformer, multi-head attention, TensorRT.

I. INTRODUCTION

WITH the rapid development of computer science and electrical engineering, autonomous systems have begun to play an increasingly important role in our daily life [1], [2], [3]. Among them, autonomous driving is undoubtedly one of the most important research fields [4], [5]. Due to autonomous driving's close relationship with human life, safety is the primary concern. Predicting the future trajectories of nearby vehicles is necessary to assist the autonomous driving system in determining the optimal path planning solution and preventing potential collisions [6], [7], [8], [9].

For human drivers, future trajectories of nearby vehicles are predicted based on two types of information: past trajectories of nearby vehicles and the structure of roads. For autonomous vehicles, things are the same. Basically, the inputs of a trajectory prediction model are twofold: past trajectories and High Definition (HD) maps. Autonomous driving systems have to understand the meaning of this information and make good use of it. An example of a trajectory prediction task is shown in Fig. 1.

Currently, learning-based methods have shown their effectiveness in trajectory prediction tasks. On the one hand, researchers considered trajectories and HD maps as raster

The review of this article was arranged by Associate Editor Dr. Vesna Šešum-Cavić.

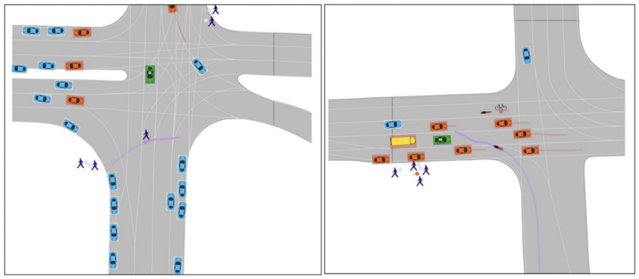

FIGURE 1. An example of trajectory prediction scenario. Normally, a scenario includes trajectory information of different agents (e.g., cars, trucks, bikes and pedestrians) and HD map information. The target is to predict the future of interested agents.

images and applied a CNN-based method to extract features and make predictions [10], [11]. Thanks to substantial off-the-shelf CNN-based models, such as VGG-16 [12] and ResNet [13], the implementation of these models is easy. However, these methods actually neglect the geometric features of trajectories, which is the key feature for understanding their hidden information. Recently, researchers started to develop the vectorized approach, i.e., they use a set of points and vectors to represent the geometric features of both trajectories and maps and then use graph convolution networks (GCNs) as the backbone of the model. The idea is fairly successful and achieves state-of-the-art performance metrics on open datasets [14], [15]. Although the prediction performance is significantly improved by those methods, the successful deployment on real vehicles is always overlooked. The models achieving SOTA performance are usually too complicated to deploy, as they require both large computing power and memory.

To address this challenge, in this work, we adopt the attention mechanism from transformers to replace the time-consuming graph convolution. Compared with convolution, the attention mechanism can achieve the same performance according to existing experiments and at the same time is more deployment-friendly because many related works have been performed. Therefore, it is a great choice for both feature extraction and combining vehicle-vehicle and vehicle-road information.

Moreover, since autonomous driving is a highly application-driven application, the model's most important performance characteristic should be whether it can be deployed on real cars, in other words, whether the model can be transformed into the form of TensorRT, which is the most commonly used deployment SDK. To accomplish this, we strictly adhere to the coding style of TensorRT when implementing the model. As a result, our model can be easily converted into TensorRT and accelerated further.

Our *Lane Transformer* is inspired by the well-known LaneGCN model [16], which was one of the first deep learning models to apply GCN to trajectory prediction tasks. The results of quantitative and qualitative experiments indicate that our model possesses the following benefits.

- By replacing graph convolution with an attention mechanism, our model achieves state-of-the-art efficiency while maintaining the performance of the LaneGCN model's baseline metrics.
- We minimize the model's parameters so that the model's size is smaller than other existing methods, making our model more appropriate for applications with limited memory.
- We strictly adhere to the TensorRT coding style so that our model can be easily converted into TensorRT format, which is a prerequisite for deployment.

II. RELATED WORK

A. TRAJECTORY PREDICTION

To date, many trajectory prediction methods have been proposed, and they perform well on open datasets. First, due to the sequential feature of trajectories, it is natural for researchers to develop an idea based on RNNs [17], [18], [19]. However, although this idea works well for the prediction of pedestrians' trajectories, researchers soon found that it is not suitable for the predictions of vehicles because unlike pedestrians, the past trajectory of a vehicle is insufficient for predicting its future trajectory. The trajectories of other nearby vehicles and the geometric structures of roads are the real keys that determine a vehicle's future trajectory. To this end, today's trajectory prediction models can be roughly divided into two categories: raster-based methods and vector-based methods.

1) RASTER-BASED METHODS

The basic idea of raster-based methods is simple: we encode the trajectories and HD maps into a stack of images and use CNN-based methods to extract the hidden features and model the interactions between vehicles and roads [10], [11], [20], [21], [22], [23]. Since CNN-based computer vision has been a hot research field in recent years, we have a large number of off-the-shelf models to use, which is convenient. For example, [20] model the scene into an image with multiple channels, and each channel represents a type of information in the scene (such as past trajectories, roads or the position of other vehicles). However, rasterization of the scene neglects the geometric features of vehicles and roads, such as the direction of cars and roads. As a result, raster-based methods show poorer performance than vector-based methods.

2) VECTOR-BASED METHODS

Similar to raster-based methods, vector-based methods also try to unify the inputs of the trajectories and HD maps [14], [15], [16], [24]. However, the difference is that vector-based methods treat both trajectories and roads as sets of points and encode them into a directed graph. Thus, the nodes and edges of this graph keep both the position and geometric structures of the scene. Then, we can use graph convolution as the tool to extract the features and combine them. Vector Net [14] is one of the first models based on this idea, and it reaches state-of-the-art performance back then. LaneGCN [16] is

similar to Vector Net in the basic idea but different in the total structure of the model. Both VectorNet and LaneGCN can be considered the origination of vector-based methods. Largely part of following works these two years are based on their ideas and achieve a better performance on open datasets [25].

To ensure the multi-modality of outputs to gain a higher performance, recent researchers have paid more attention to the trajectory generation block of the model. GOHOME [22] outputs a heatmap representation of the possible final points of the future trajectory and uses NMS (Non-Maximum Suppression) to avoid the similar outputs. DenseTNT [24] first samples a group of points on the road as the candidates for final points and outputs a confidence score for each point. Multipath [10] outputs the distribution of future behavior parameterized as a GMM(Gaussian Mixture Model).

B. TRANSFORMER

After being proposed in 2017, Transformer soon became one of the most popular model structures in the deep learning field [26]. On the one hand, its unbelievable high performance in NLP(Natural Language Processing) shows its strong ability as a novel model. On the other hand, the unexplainable feature of Transformer causes heated discussion on whether we can depend on this 'black box'.

The most important part of the transformer is the attention mechanism, which can be partly seen as a weighting procedure. Many experiments have already proven that the attention mechanism has the ability to learn, encode and decode complex features. Recent works show that transformer and attention mechanisms can also be used in computer vision tasks [27], [28] or to be more specific, trajection prediction tasks [29], [30]. For example, mmTransformer [31] uses a pure transformer as the backbone and achieves state-of-the-art performance. Reference [32] used an attention mechanism as a part of a trajectory prediction model and provided a series of experiments to show the influence of different types of attention blocks. Scene Transformer [33] adopts a scene-centric approach based on the attention mechanism to achieve high performance on both single- and multiple-agent prediction tasks. All of these works show that the transformer has the potential to replace the convolution blocks in a trajectory prediction model.

III. METHOD

A. PROBLEM DEFINITION

In the section above, we only provide a vague definition of the trajectory prediction task: input trajectories, HD maps and output future trajectories. Here, we provide a more detailed description of the trajectory prediction task.

Here is the basic mathematical modeling of trajectory prediction. First, for a given scene, there are n agents in it, including vehicles, pedestrians, and bikes, denoted as $\mathbb{A}$. We have the past trajectory information of the target vehicle a_{target} and its nearby neighbors $\mathbb{A}_{nbrs} = \{a_1, a_2, \ldots, a_{n-1}\}$, in an observation time period t_{obs}. Each past trajectory is defined as follows (taking a_i as an example):

$$\mathbf{P}_i = \left[\mathbf{p}_i^{-t_{obs}+1}, \mathbf{p}_i^{-t_{obs}+2}, \ldots, \mathbf{p}_i^{0}\right], \tag{1}$$

where $\mathbf{p}_i^t = [x_i^t, y_i^t]$ denotes the position of agent i at time step t. Usually we use the bird's-eye-view coordinate system to represent the scene.

On the other hand, we also have the HD map information in a given area, including lanes, traffic lights, sidewalks, etc. The mathematical definition of roads is similar to trajectories since they are all modeled as a series of points.

For the output, we aim to predict the future trajectory of the target vehicle a_{target} in the time period t_{pred}, defined as follows:

$$\mathbf{P}_i = \left[\mathbf{p}_i^1, \mathbf{p}_i^2, \ldots, \mathbf{p}_i^{t_{pred}}\right]. \tag{2}$$

Similar to the input, $\mathbf{p}_i^t = [x_i^t, y_i^t]$ denotes the position of agent i at time step t. Finally, we evaluate the performance of the model based on $\mathbf{P}_i$.

B. OVERALL MODEL

The overall framework of the *Lane Transformer* is illustrated in Fig. 2. Basically, inspired by LaneGCN [16], our model is made up of four main components. First, we preprocess the raw input data, including trajectories and HD maps, into vector form. Second, we use an attention-based encoder to extract the hidden feature of each agent and road. Third, we use a stack of transformer blocks to fuse the features between agents and roads and an attention-based block to aggregate the high-order interaction. Finally, an MLP-based decoder receives the combined information of trajectories and maps and then generates the future trajectories of the target agent.

C. DATA PREPROCESSING

Normally, trajectories and HD maps in datasets are annotated as a series of points with additional semantic information, (*e.g.,* speed limit of the road, whether the traffic line is dashed and solid). However, points themselves can only represent the static position of vehicles and roads, instead of their geometric motion features such as directions. Therefore, we adopt a vectorization preprocess for both trajectories and HD maps.

For agents, we sample the trajectory points with the same time interval and then connect the neighboring points into vectors. Thus, each trajectory contains several vectors. For roads, we first cut them into segments with the same length(some researchers call these segments *polylines* or *splines*), then sample them with the same distance and connect the neighboring points into vectors. Similar to a trajectory, each road segment contains several vectors. Therefore, we successfully unify the form of trajectories and HD maps as vectors, as shown below:

$$d_i = \left[x_i, y_i, x_i^{pre}, y_i^{pre}, \Delta x, \Delta y, vid, sid\right], \tag{3}$$

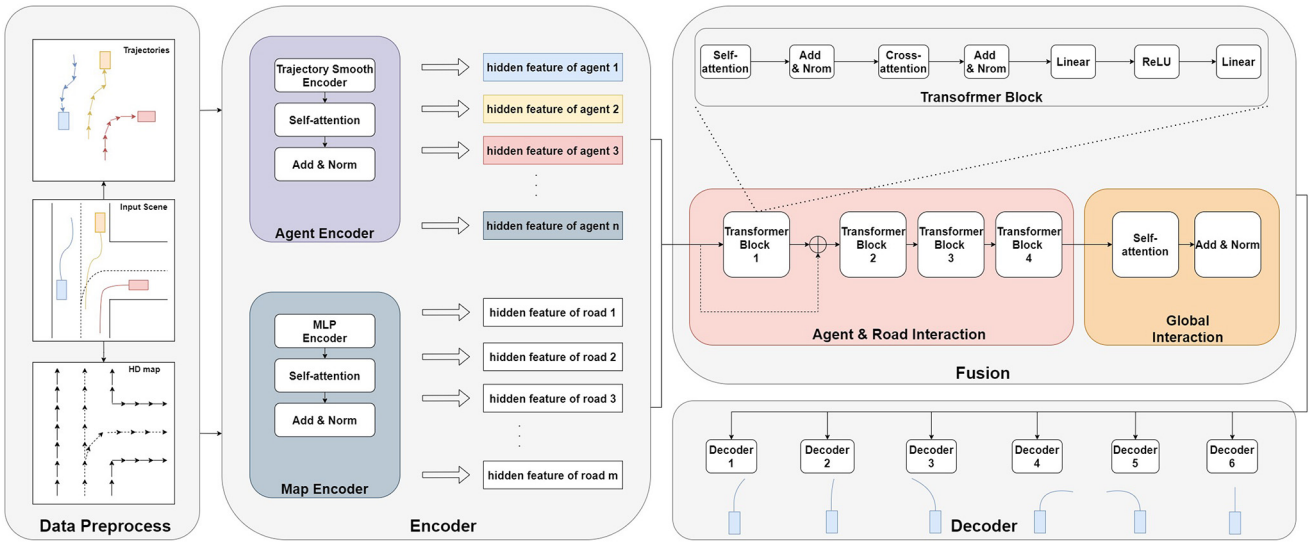

FIGURE 2. Overall framework of the *Lane Transformer*. Trajectory and HD map data are first preprocessed into the form of vectors and passed through the agent and map encoder separately. Then, the encoded features are sent into the fusion block to aggregate both local and global interactions. Finally, the feature vectors are sent into different decoders, which ensures the multi-modality of the results, to generate the final prediction of the future trajectory.

where $[x_i, y_i]$ is the position of point i in the BEV coordinate system, and $[x_i^{pre}, y_i^{pre}]$ is the position of its pre-neighbor. Δx and Δy are the displacement between point i and its pre-neighbor, which represent the direction of trajectories or roads. *vid* and *sid* are the identification of the vector and the segment (trajectory or road) to which it belongs. These two IDs help us to specify the position of this vector in the scene.

It is worth mentioning that to improve the robustness of the model, we avoid the use of absolute coordinates. To be more specific, we first normalize all of the trajectories and HD maps into the new relative coordinate system centered at the target agent's position at the last time stamp. This process helps us avoid the potential quantitative issue (*e.g.,* the large difference in the absolute coordinates between scenes A and B).

D. SCENE CONTEXT ENCODING

After preprocessing, we now have the trajectories and HD maps in the unified form of vectors. Then, the model needs to learn the spatial information of inputs by encoding them into feature vectors. Unlike most of the previous trajectory prediction studies, we take efficiency as an important issue. Thus, we try to design an effective encoder while keeping its structure as simple as possible. Referencing existing works and our own experiments, we adopt the combination of an attention block, CNN and MLP as the encoder. More details are as follows.

For roads, we first use an MLP-based encoder to transform the input vector at each time stamp (*e.g.,* vector of road segment i at position t) into features:

$$f_i^t = \delta_{map}\left(d_i^t; W_{map}\right), \tag{4}$$

where δ_{map} is a 3-layer MLP with a ReLU non-linear layer and W_{map} is the weight matrix that is learnable. Thus, we now have the feature matrix with a size of (M, L, H), where M is the number of road segments in the scene, L is the fixed length of one segment (*e.g.,* 10m) and H is the length of the hidden feature. However, to predict the future trajectory, the separate feature of each vector is insufficient. For example, even if two road segments in the first half have the same structure, the difference in the last half can result in a total difference in geometric meaning. Therefore, we use a self-attention block to encode the overall feature of one road segment and form a single feature vector for each agent.

To be more specific, we first calculate the query, key and value matrix:

$$q_i^t = W^q f_i^t, k_i^t = W^k f_i^t, \boldsymbol{v}_i^t = W^q f_i^t, \tag{5}$$

where W^q, W^k, W^v are the learnable weight matrices. Then, we take these three matrices as the inputs of the weighting block based on softmax:

$$h_i^t = softmax\left(\frac{q_i^t \cdot k_i^{tT}}{\sqrt{d_k}}\right)\boldsymbol{v}_i^t, \tag{6}$$

where d_k is the length of matrix k. Finally, we adopt a 2-layer MLP to aggregate the features of vectors within a road segment:

$$h_i = \delta_{agg}\left(h_i^t; W_{agg}\right), \tag{7}$$

where δ_{agg} is a 2-layer MLP with a ReLU non-linear layer and W_{agg} is the weight matrix that is learnable. Now, we have the feature vector for each road segment, stored as a 2D matrix (M, H), where M is the number of road segments and H is the length of hidden features.

For agents, we use similar techniques to encode and aggregate the information. In particular, we use a trajectory encoder block to encode each vector into the form of a feature vector. Then, similar to roads, even two vehicles

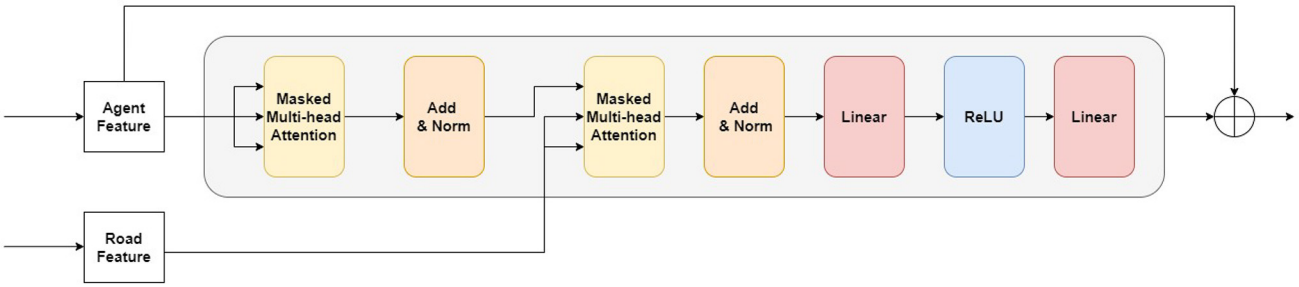

FIGURE 3. We adopt a multi-head attention block to aggregate the information of the agent and road.

have the same movement in the first half of their trajectory, and the differences in the last half of trajectories can lead to a totally different future trajectory. Therefore, we use a self-attention block to encode the overall feature of one trajectory in the observed time period and form a single feature vector for each agent. Finally, a 2-layer MLP-based aggregator is used to construct a single feature vector for each trajectory.

One aspect worth mentioning is the agent encoder. In our experiments, we found that, unlike roads, trajectory data are usually non-smooth. Reasons for this phenomenon are unclear and might be due to the uncertainty of the location system, such as GPS. However, this phenomenon does influence trajectory prediction because non-smooth trajectories cannot represent the real motion pattern of vehicles. Thus, we use a 1D CNN-based trajectory encoder in the first stage of the agent encoder, instead of the MLP-based block in the road encoder. Because of the wider receptive field compared with MLP, the CNN-based encoder smooths the trajectories and reduces the influence of bad data quality.

E. HIERARCHICAL INFORMATION FUSION

For now, we have the feature vector of each agent and road in the form of a 2D matrix. The shape of agents' feature matrix is (N, L) and the shape of roads' feature matrix is (M, L), where N is the number of agents, M is the number of roads and L is the length of feature vectors. Since the motion of a vehicle is influenced by its past trajectory, its neighbors' past trajectories and the road structure, we need to fuse the feature vectors of both agents and roads. Moreover, local interactions and global interactions should both be considered. For example, imagine a scene with three vehicles in it, A, B and C, where A is our target agent. On the one hand, the geometric relation between vehicles A and B may influence A's future trajectory, the same for A and C. On the other hand, when we take the scene as a whole and consider both A, B and C, their interactions might result in a totally different future motion. Thus, we propose a hierarchical information fusion block based on a transformer, which fuses the information at both the local and global levels.

Thanks to the ability to capture long-range spatio-temporal dependencies, attention-based transformer blocks are well-used in deep learning models, especially those focused on the combination of information. Therefore, we adopt the attention mechanism as the foundation of our hierarchical information fusion block.

1) LOCAL INTERACTION

For local information fusion, we adopt a structure similar to the transformer decoder. The network structure is shown in Fig. 3. The first multi-head attention block can be considered a self-attention procedure, and the second multi-head attention block, which is responsible for information aggregation, works as a cross-attention. We adopt four layers of this transformer block as a stack to increase the model's ability to represent complex situations. It is worth mentioning that the positions of the agent feature and road feature in Fig. 3 are exchangeable, which means that we can fuse the information from roads to agents or vice versa. Actually, in our implementation, only the second and fourth layers of the transformer block are the same as in Fig. 3. The first and third layers of the transformer block exchange the input positions of the agent and road. Through this exchange technique, we can better realize the information aggregation and avoid the bias caused by agents or roads alone.

2) GLOBAL INTERACTION

The attention block does not change the shape of the feature matrix, so the feature matrix of agents and roads are still (N, L) and (M, L). However, the L length feature now contains not only its own geometric feature but also its interaction information with each other agent and road. Therefore, what we need is now to aggregate all of the local interactions in the scene, or so-called global interaction fusion. Similar to local fusion, we still adopt an attention-based block to achieve this goal, or to be more specific, a self-attention process between each agent's feature based on the target agent, resulting in a single feature vector that contains the whole geometric information in the scene from the view of the target agent.

F. TRAJECTORY PREDICTION

Multi-modality is a significant feature of trajectory prediction tasks. In other words, for a given scene including past trajectories and road structures, multiple possible future trajectories exist. For example, at a crossroad, turning right or left is possible. Therefore, it requires the decoder of our

model, the trajectory construction block, to have the ability to generate K different future trajectories, where K is a parameter to be defined.

Referencing the trajectory prediction works by far [14], [32], we adopt the MLP-based decoder in our model. To ensure the multi-modality of our model, our trajectory prediction block consists of six MLP-based trajectory decoders with the same structure. During training, the parameters of these six decoders have different forms and thus achieve the multi-modality of trajectories:

$$p_i^k = \phi_{dec}^k(h_i; W_k), \tag{8}$$

where p_i^k is the k_{th} predicted trajectory of agent i, ϕ_{dec}^k is the k_{th} trajectory decoder, W_k is the weight matrix of the k_{th} trajectory decoder and h_i is the feature vector.

Furthermore, we adopt a single MLP to generate the confidence score of each prediction, which helps us choose the most possible trajectory:

$$c_i = \phi_{scoring}(h_i; W_{scr}), \tag{9}$$

where c_i is the vector consisting of the confidence score of each trajectory, $\phi_{scoring}$ is the scoring decoder, $W_{scoring}$ is the weight matrix and h_i is the feature vector.

G. TRAINING LOSS

Inspired by the variety loss [34], [35] and LaneGCN's loss function [16], we adopt the following loss function:

$$\mathcal{L} = \mathcal{L}_{reg} + \mathcal{L}_{cls}, \tag{10}$$

where $\mathcal{L}_{reg}$ is the regression loss and $\mathcal{L}_{cls}$ is the classification loss.

In the calculation of loss, we first define the trajectory that has the closest distance to the ground truth at the final point as the target trajectory, annotated as k^*. The following calculation will be based on this trajectory.

For classification loss, we define the loss function as follows:

$$\mathcal{L}_{cls} = \frac{1}{N(K-1)} \sum_{i=1}^{N} \sum_{k \neq k^*} max\Big(0, c_i^k + \epsilon - c_i^{k^*}\Big), \tag{11}$$

where N is the number of agents, c_i^k is the confidence score of the k_{th} predicted trajectory of agent i and ϵ is the threshold, which is 0.2 here.

For regression loss, unlike previous work, we adopt a two-part structure loss function:

$$\mathcal{L}_{reg} = \alpha \mathcal{L}_{FDE} + (1-\alpha)\mathcal{L}_{ADE}, \tag{12}$$

where $\mathcal{L}_{FDE}$ is the MSE loss between trajectory k^* and ground truth, $\mathcal{L}_{ADE}$ is the Euclidean distance between trajectory k^* and ground truth at the final time stamp and α is a weight factor that balances the two loss factors (we adopt α=0.5 in our experiments).

H. CODE STYLE

To make it possible for our model to be deployed on real cars, we must ensure that our model can be easily transformed into the form of TensorRT. TensorRT [36] is an SDK for high-performance deep learning inference in C++ form that includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for inference applications, developed by Nvidia. It is the foundation of the deployment of deep learning models and has been used in many real scenarios, such as online shopping and recommender systems. However, due to the asynchronous development between TensorRT and Python, some codes cannot be transformed into TensorRT. Here are some examples: we always use '−1' in Python to represent the default length of a vector, but this can not be recognized in TensorRT; the data type of the mask in the model input has to be 'bool' in some versions of TensorRT but could be 'float' in some other versions; some operation functions are not implemented in TensorRT, so we need to avoid these functions, such as the 'flatten()' function in PyTorch. Overall, we carefully avoid these issues and strictly follow the code style of TensorRT when building the model. Therefore, we ensured that our model could be easily transformed into TensorRT with minor adjustments.

IV. EXPERIMENTS

In this section, we first describe the experimental settings and details, including the datasets, implementation details and metrics. Then, we show the overall performance of our model from the view of quantity and quality compared with other state-of-the-art models. Finally, comprehensive ablation studies are performed to prove the efficiency and accuracy of our model structure.

A. EXPERIMENTAL SETUP

1) DATASETS

We evaluate our model on the Argoverse dataset [37], the widely used large-scale motion forecasting dataset. The Argoverse dataset contains a total of 323557 trajectory prediction scenes sampled from two cities, Pittsburgh and Miami, in 2019. These scenes are divided into three parts, training, validation and test, with 205,942, 39,472, and 78,143 samples, respectively. Each sample contains an interesting target called an "agent", and our task is to predict its future trajectory. For the training and validation sets, each scene contains sequence data of 5 seconds, sampled at 10 Hz. The first two seconds of data are used as the input of the model and the last three seconds of data are only used as the ground truth to evaluate the performance of the model. For the test set, it only provides the data of the first two seconds of trajectory and requires our model to propose the prediction of three seconds of trajectory in the future. For the data structure, both trajectory data and map data are in the form of a series of points. Each point contains its X and Y coordinates. For trajectories, these points are the position of the agent at each timestamp. Maps are the set of lane

TABLE 1. Quantitative results on the Argoverse Dataset. The results of our model are compared with Argoverse baseline and LaneGCN in the table.

Method	K=6			K=1		
	minADE	minFDE	MR	minADE	minFDE	MR
Argoverse Baseline [37]	1.71	3.29	0.54	3.45	7.88	0.87
LaneGCN [16]	0.87	1.36	0.16	**1.71**	**3.78**	**0.59**
Lane Transformer	**0.86**	**1.31**	**0.15**	1.75	3.84	**0.59**

centerlines and their semantic information, including traffic lights or turning rules.

2) IMPLEMENTATION DETAILS

As we mentioned in the previous section, to improve the robustness of our model, we first centered the BEV coordinate system of a given scene at the position of the agent at $t = 0$, which is the last observed position of the agent. We train the model on three GeForce GTX 1080Ti for 64 epochs with a batch size of 128 and the Adam optimizer. For the learning rate, we adpot the warm-up trick [38]: the learning rate gradually increases from 5×10^{-4} to 5×10^{-3} in the first 10 epochs and then decays to 80% after every 10 epochs [39]. The training process takes approximately 17 hours to complete.

3) METRICS

We evaluate our model on the standard metrics for trajectory prediction, including minADE (minimum Average Displacement Error), minFDE (minimum Final Displacement Error) and MR (Miss Rate). For each sample, our model outputs K different predictions ($K = 6$ in our experiments) and defines the trajectory that has the closest distance to the ground truth at the last timestamp as the best-predicted trajectory, annotated as $\hat{p}$.

The minADE score measures the l_2 distance in meters between the best-predicted trajectory and the ground-truth trajectory averaged over all future time stamps:

$$minADE = \frac{1}{T}\sum_{t=1}^{T}\left\|\hat{\boldsymbol{p}}_i^t - \mathbf{p}_i^t\right\|_2, \tag{13}$$

and the minFDE score measures the displacement at the final time stamp T:

$$minFDE = \left\|\hat{\boldsymbol{p}}_i^T - \mathbf{p}_i^T\right\|_2, \tag{14}$$

where $\hat{p}_i^t$ here denotes agent i's predicted position at time stamp t and p_i^t is agent i's true position at time stamp t.

As $\hat{\mathbf{p}}$ and $\mathbf{p}$ are vectors containing x and y coordinates of agents, minADE and minFDE represent both the longitudinal and lateral errors of predictions.

The MR score refers to the ratio of samples where the distance between the best-predicted trajectory endpoint and the ground truth endpoint is larger than two meters.

TABLE 2. Inference time on the Argoverse Dataset. The inference time is reported by the authors or calculated using the official implementations.

Method	Inference Time(ms)
LaneGCN [16]	173
HiVT-128 [40]	69
HOME+GOHOME [22]	32
THOMAS [41]	20
DenseTNT (w/ 100ms opt.) [24]	2644
DenseTNT (w/ goal set pred.) [24]	531
Lane Transformer	**16**
Lane Transformer(TensorRT)	**7**

B. QUANTITATIVE RESULTS

1) METRICS PERFORMANCE

As we mention before, the main target of our model is to achieve a higher efficiency while maintaining the metrics performance of the baseline LaneGCN. Thus, here, we mainly compare the metrics of minADE, minFDE and MR of our model with LaneGCN. Furthermore, we also provide the baseline result of the Argoverse official to show that our model predicts an ideal future trajectory. We provide the result in two different settings: $K = 1$ and $K = 6$. The $K = 6$ result shows the multi-modality ability of the model, and the $K = 1$ result shows the model's ability to judge the most possible trajectory.

As shown in Table 1, our model achieves a similar performance as our baseline model LaneGCN, even a few better when K=6, which is the main ranking metric on open datasets. On the other hand, comparing with the Argoverse Baseline, which is a non-deep-learning Weighted Nearest Neighbor regression method, shows that our model predicts an ideal and convincible future trajectory.

2) INFERENCE TIME AND MODEL SIZE

High inferencing efficiency and small model size are the main advantages of the *Lane Transformer*. Here, we compare our model with several state-of-the-art models on the leader board of the Argoverse Motion Forecasting Challenge. We define the inference time as the time cost of predicting the three-second future trajectory of the target vehicle in one scene. The results are shown in Table 2.

Among the models we compare, LaneGCN [16] and HiVT [40] are based on the GCN block and predict the future motion based on the regression of the whole trajectory. On the other hand, HOME+GOHOME [22], [42], THOMAS [41] and DenseTNT [24] focused on the possibility distribution of the final point and used optimization

TABLE 3. Parameter number of different models. The model size is reported by the authors or calculated using the official implementations.

Method	Model Size($\times 10^3$)
LaneGCN [16]	3,701
HiVT-128 [40]	2,529
HOME+GOHOME [22]	5,100
mmTransformer [31]	2,607
Scene Transformer [33]	15,296
Lane Transformer	**1,545**

to increase the accuracy of the model. All of these models are or used to be the state-of-the-art model on the leader board, considering their metrics performance. However, as for efficiency, our model performs better. This result proves that our methods, including replacing the GCN block with transformer blocks and reducing the model parameters, are beneficial for model efficiency.

To further prove this, we show the model size of these models in Table 3. There is an interesting phenomenon here: unlike the intuitive sense that a larger model runs slower, the model size and model efficiency are not exactly the same. Better model construction can also influence model efficiency. However, for models that are based on similar ideas, such as LaneGCN and HiVT, a smaller model size will reduce the inference time. For our model, which has a similar idea as LaneGCN, this rule works clearly. With the smallest model size, our model achieves the best efficiency compared with others.

On the other hand, as we mentioned before, we follow the code style of TensorRT when constructing the model, which means our model can be easily transformed into the form of TensorRT. If the code of the model does not follow the code style of TensorRT, which is common for many state-of-the-art models, it will face many obstacles during the transformation from Python to TensorRT. Actually, among the models we compared before, after removing those that are closed source so we cannot read the code, none of them can be easily transformed into TensorRT, which means these models can never be put into real vehicles. Here, the advantage of our model is clear, i.e., it can satisfy the requirement of deployment in real vehicles, which is the main goal of autonomous driving. Besides, TensorRT can further accelerate the model. As shown in the last line of Table 2, the TensorRT version of our model is even faster than the Python version, which is sufficient for real-time inference.

C. QUALITATIVE RESULTS

As shown in Fig. 4, we present the qualitative results of our model on the Argoverse validation set. To clearly show the scene, we only visualize the centerlines of roads and the trajectories of the target agent. In the image, the past trajectories are shown in orange and the ground truth trajectories are shown in red. To show the multi-modality of our model, we visualize the total of 6 predicted trajectories with blue lines.

TABLE 4. Ablation experiments of the fusion block.

Local Fusion	Global Fusion	minADE	minFDE	MR
✓	✓	**0.8657**	**1.3156**	**0.1521**
	✓	0.9556	1.5796	0.2089
✓		0.8674	1.3212	**0.1521**

TABLE 5. Ablation experiments of the attention mechanism in the scene context encoder.

Agent Encoder	Map Encoder	minADE	minFDE	MR
✓	✓	**0.8657**	**1.3156**	**0.1521**
	✓	0.8684	1.3184	**0.1521**
✓		0.8663	1.3178	0.1522

The visualization results prove that our model can make the accurate prediction of future trajectories since the blue arrows, which show the final position of the predicted trajectories, are close to the red arrows, which is the ground-truth. In addition, the multi-modality of our model is shown. For example, in the crossroad, our model will give the prediction of both going straightforward or turning.

D. ABLATION STUDY

1) IMPORTANCE OF HIERARCHICAL INFORMATION FUSION BLOCK

As we mentioned before, we adopt the transformer-based hierarchical information fusion block to aggregate the local and global information between roads and trajectories. To illustrate the importance of this block, we conduct the following experiment: since attention-based blocks do not change the size of the vector, we can simply remove those blocks. By comparing the results before and after, the importance of these blocks is shown. The results of these experiments are shown in Table 4.

According to the results, the local fusion block plays a more important role in the model than global fusion, while both of these blocks improve the performance of the model. This phenomenon inspires us that although global information will influence the prediction of future trajectories, a vehicle's future is more likely to be decided based on its nearby vehicles.

2) IMPORTANCE OF ATTENTION MECHANISM IN SCENE CONTEXT ENCODER

To further illustrate the function of the attention mechanism, we also need to prove its importance in the scene context encoder. Unlike the fusion block, we cannot simply remove the whole block since the structure of the data has been changed in this block. Thus, we remove the attention block in the actor encoder or road encoder to compare their influence on the performance of the model. The results are shown in Table 5.

From the experiments, we find that, basically, the attention mechanism in both the agent and map encoder helps the model to better encode the feature and improve the model performance, which proves that the attention block is useful

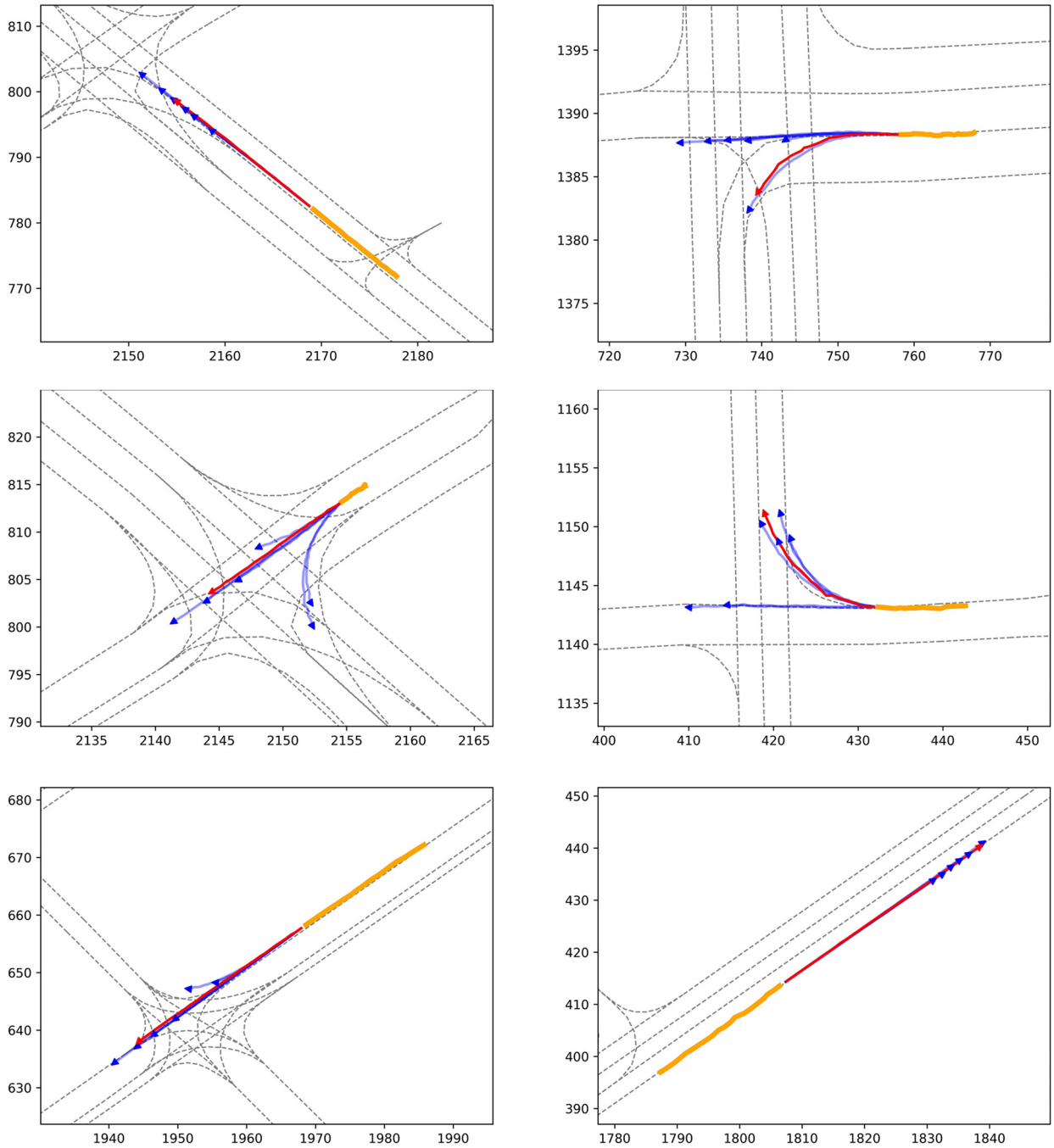

FIGURE 4. Visualization of the road and the trajectories of the target agent on the Argoverse validation set. The red, blue and orange lines represent the ground truth, predicted trajectories and observed trajectories, respectively.

TABLE 6. Ablation experiments of the bottle neck and smooth encoder. Both the quantitative and efficiency results are shown in the table.

Bottle Neck	Smooth Encoder	minADE	minFDE	MR	inference time(ms)
✓	✓	**0.8641**	**1.3119**	**0.1503**	21
	✓	0.8657	1.3156	0.1521	**16**
✓		0.8906	1.3694	0.1715	17

for the information encoding procedure. Although MR is almost the same, lower minADE and minFDE ensure a better trajectory quality.

3) EXPERIMENTS OF BOTTLE NECK BLOCK AND SMOOTH BLOCK

In some recent research works [43], we noticed that some model structures can further improve the performance of

TABLE 7. Comparison of efficiency between transformer block and GCN block.

Block	Time(ms)
GCN	3.0
Transformer	**1.7**

the model, especially those that can be transformed into TensorRT. Despite those that are not suitable for our model due to the difference in backbone, we found that the bottle neck block might be an interesting attempt. The bottle neck block is proposed in ResNet [13]. Basically, in this block, the length of feature vectors are reduced at first and then enlarged to the original length. Through this procedure, the information is distilled and thus reduces the influence of unrelated information. We try to put the bottle neck block at the end of each transformer block, according to the instruction of [43] and the results are shown in Table 6.

On the other hand, as we mentioned before, we adopt a CNN-based encoder to address the non-smooth trajectory data. To illustrate the performance of this block, we compare the performance of our model with and without this smooth encoder block, and the results are also shown in Table 6.

From the table, it is clear that both the bottle neck and smooth encoder have some benefits for the model's performance. However, the smooth encoder causes a much greater improvement than the bottle neck block. This phenomenon shows that compared with the model structure, better data quality might be the key for higher performance. Furthermore, since both the bottle neck block and smooth encoder are CNN based, which is quite time consuming, both of these blocks will delay the inference of the model. Since we are looking forward to obtaining a model with high efficiency, we give up the bottle neck block and only keep the smooth encoder in our model's final version.

4) COMPARISON OF EFFICIENCY BETWEEN TRANSFORMER BLOCK AND GCN BLOCK

To further prove that the transformer block is more efficient than the GCN block, we compare the efficiency of the fusion block between our model and LaneGCN. The *A2L* block in LaneGCN acts the same function as the first transformer fusion block in our model. The inputs of these two blocks are similar: both include the agent features and road features, the sizes of which are (N, L) and (M, L). They both generate the same output: the road features, which keep the size (M, L), after being fused with the agent features. To ensure equality, we run both our model and LaneGCN on the Argoverse validation set and compute the average time consumed by our transformer block or GCN block. The efficiencies of these two blocks are shown in Table 7.

From the table, we note that the transformer block in our model is more efficient than the GCN block in the LaneGCN, while the i/o data structure and function are similar.

V. CONCLUSION

In this paper, we have proposed the *Lane Transformer* model, which is a highly efficient model for trajectory prediction. Due to the switch from the GCN to the transformer block, our model achieves state-of-the-art efficiency performance in comparison to other existing open source models. In contrast to the baseline LaneGCN, our speedup is 10× to 25× while maintaining comparable prediction accuracies on the public dataset. Importantly, since we strictly adhere to the code style of TensorRT, our model can be easily transformed into TensorRT format, which facilitates future deployment on real vehicles.

REFERENCES

[1] E. Zhang, N. Masoud, M. Bandegi, and R. K. Malhan, "Predicting risky driving in a connected vehicle environment," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 17177–17188, Oct. 2022.

[2] Z. Li, X. Huang, T. Mu, and J. Wang, "Attention-based lane change and crash risk prediction model in highways," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 22909–22922, Dec. 2022.

[3] O. Scheel, N. S. Nagaraja, L. Schwarz, N. Navab, and F. Tombari, "Recurrent models for lane change prediction and situation assessment," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 17284–17300, Oct. 2022.

[4] Q. Zhang et al., "TrajGEN: Generating realistic and diverse trajectories with reactive and feasible agent behaviors for autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 24474–24487, Dec. 2022.

[5] Y. Guo, D. Yao, B. Li, Z. He, H. Gao, and L. Li, "Trajectory planning for an autonomous vehicle in spatially constrained environments," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 18326–18336, Oct. 2022.

[6] Y. Lu, W. Wang, X. Hu, P. Xu, S. Zhou, and M. Cai, "Vehicle trajectory prediction in connected environments via heterogeneous context-aware graph convolutional networks," *IEEE Trans. Intell. Transp. Syst.*, early access, May 24, 2022, doi: 10.1109/TITS.2022.3173944.

[7] K. Chen, X. Song, H. Yuan, and X. Ren, "Fully convolutional encoder-decoder with an attention mechanism for practical pedestrian trajectory prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 11, pp. 20046–20060, Nov. 2022.

[8] K. Zhang, X. Feng, L. Wu, and Z. He, "Trajectory prediction for autonomous driving using spatial-temporal graph attention transformer," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 11, pp. 22343–22353, Nov. 2022.

[9] X. Liu, Y. Wang, K. Jiang, Z. Zhou, K. Nam, and C. Yin, "Interactive trajectory prediction using a driving risk map-integrated deep learning method for surrounding vehicles on highways," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 19076–19087, Oct. 2022.

[10] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," in *Proc. Conf. Robot Learn. (CoRL)*, Osaka, Japan, 2019, pp. 86–99.

[11] H. Cui et al., "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2019, pp. 2090–2096.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, 2015, pp. 1–14.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778.

[14] J. Gao et al., "VectorNet: Encoding hd maps and agent dynamics from vectorized representation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 11522–11530.

[15] M. Ye, T. Cao, and Q. Chen, "TPCN: Temporal point cloud networks for motion forecasting," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 11313–11322.

[16] M. Liang et al., "Learning lane graph representations for motion forecasting," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Glasgow, U.K., 2020, pp. 541–556.

[17] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 961–971.
[18] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction," in *Proc. IEEE Int. Conf. Intell. Transp. Syst. (ITSC)*, Yokohama, Japan, 2017, pp. 353–359.
[19] A. Khosroshahi, E. Ohn-Bar, and M. M. Trivedi, "Surround vehicles trajectory analysis with recurrent neural networks," in *Proc. IEEE Int. Conf. Intell. Transp. Syst. (ITSC)*, Rio de Janeiro, Brazil, 2016, pp. 2267–2272.
[20] J. Hong, B. Sapp, and J. Philbin, "Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, 2019, pp. 8446–8454.
[21] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Glasgow, U.K., 2020, pp. 683–700.
[22] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "GOHOME: Graph-oriented heatmap output for future motion estimation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Philadelphia, PA, USA, 2022, pp. 9107–9114.
[23] N. Djuric et al., "Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving," in *Proc. IEEE Conf. Appl. Comput. Vis. (WACV)*, Snowmass Village, CO, USA, 2020, pp. 2084–2093.
[24] J. Gu, C. Sun, and H. Zhao, "DenseTNT: End-to-end trajectory prediction from dense goal sets," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2021, pp. 15283–15292.
[25] H. Zhao et al., "TNT: Target-driven trajectory prediction," in *Proc. Conf. Robot Learn. (CoRL)*, 2020, pp. 895–904.
[26] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Long Beach, CA, USA, 2017, pp. 5998–6008.
[27] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–22.
[28] Z. Liu et al., "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2021, pp. 10012–10022.
[29] C. Yu, X. Ma, J. Ren, H. Zhao, and S. Yi, "Spatio-temporal graph transformer networks for pedestrian trajectory prediction," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Glasgow, U.K., 2020, pp. 507–523.
[30] Y. Yuan, X. Weng, Y. Ou, and K. Kitani, "Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2021, pp. 9793–9803.
[31] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou, "Multimodal motion prediction with stacked transformers," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 7573–7582.
[32] K. Messaoud, I. Yahiaoui, A. Verroust-Blondet, and F. Nashashibi, "Attention based vehicle trajectory prediction," *IEEE Trans. Intell. Veh.*, vol. 6, no. 1, pp. 175–185, Mar. 2021.
[33] J. Ngiam et al., "Scene transformer: A unified architecture for predicting future trajectories of multiple agents," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022, pp. 1–25.
[34] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social GAN: Socially acceptable trajectories with generative adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 2255–2264.
[35] L. Thiede and P. Brahma, "Analyzing the variety loss in the context of probabilistic trajectory prediction," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Seoul, South Korea, 2019, pp. 9953–9962.
[36] (Nvidia, Santa Clara, CA, USA). *TensorRT, Devoloper Guide*. (2021). Accessed: Jul. 20, 2022. Available: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html
[37] M. Chang et al., "Argoverse: 3D tracking and forecasting with rich maps," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, 2019, pp. 8740–8749.
[38] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with warm restarts," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–16.
[39] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, 2019, pp. 1–19.
[40] Z. Zhou, L. Ye, J. Wang, K. Wu, and K. Lu, "HiVT: Hierarchical vector transformer for multi-agent motion prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, 2019, pp. 8823–8833.
[41] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "THOMAS: Trajectory heatmap output with learned multi-agent sampling," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022, pp. 1–18.
[42] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "HOME: Heatmap output for future motion estimation," in *Proc. IEEE Int. Conf. Intell. Transp. Syst. (ITSC)*, Indianapolis, IN, USA, 2021, pp. 500–507.
[43] X. Xia et al., "TRT-ViT: TensorRT-oriented vision transformer," 2022, *arXiv:2205.09579*.

ZHIBO WANG received the B.S. degree in computer science from the Beijing Institute of Technology, Beijing, China, in 2021. He is currently pursuing the M.S. degree with the Institute of Science and Technology for Brain-Inspired Intelligence, Fudan University, Shanghai, China. His research focuses on trajectory prediction based on machine learning and its application in autonomous driving systems.

JIAYU GUO received the bachelor of science degree from Wuhan University in 2020. He is currently pursuing the Ph.D. degree with ISTBI, Fudan University, China. He is interested in transformer-based models for trajectory prediction and robust prediction across different scenarios.

ZHENGMING HU received the master's degree in mechanical design from the Wuhan University of Science and Technology in 2011. He is a Senior Engineer with Mogo Auto Intelligence and Telematics Information Technology Company Ltd. His research and expertise are multi-sensor fusion and trajectory prediction.

HAIQIANG ZHANG received the Ph.D. degree from the Beijing Institute of Technology, Beijing, China, in 2010. He is currently a Technique Director with Mogo Auto Intelligence and Telematics Information Technology Company Ltd., Beijing, China. His current research interest is to develop perception and localization methods for autonomous driving.

JUNPING ZHANG (Senior Member, IEEE) received the B.S. degree in automation from Xiangtan University, China, in 1992, the M.S. degree in control theory and control engineering from Hunan University, Changsha, China, in 2000, and the Ph.D. degree in intelligent systems and pattern recognition from the Institution of Automation, Chinese Academy of Sciences in 2003. He has been a Professor with the School of Computer Science, Fudan University since 2006. His research interests include machine learning, image processing, biometric authentication, and intelligent transportation systems. He has been an Associate Editor of IEEE INTELLIGENT SYSTEMS since 2009 and was an Associate Editor of IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS from 2010 to 2018. He has been an Associate Editor of IEEE TRANSACTIONS ON INTELLIGENT VEHICLES since 2022.

JIAN PU (Member, IEEE) received the Ph.D. degree from Fudan University, Shanghai, China, in 2014, where he is currently a Young Principal Investigator with the Institute of Science and Technology for Brain-Inspired Intelligence. He was a Postdoctoral Researcher with the Institute of Neuroscience, Chinese Academy of Sciences, China, from 2014 to 2016, and was an Associate Professor with the School of Computer Science and Software Engineering, East China Normal University from 2016 to 2019. His current research interest is to develop machine learning and computer vision methods for autonomous driving.