Appendix — Aware of the History: Trajectory Forecasting with the Local Behavior Data

Yiqi Zhong$^{1[0000-0002-0928-8018]}$, Zhenyang Ni$^{2[0000-0001-7134-620X]}$, Siheng Chen$^{2,\star[0000-0001-6199-529X]}$, and Ulrich Neumann$^{1,\star[0000-0001-8977-7112]}$

^{1} University of Southern California, Los Angeles, CA 90089, USA
{yiqizhon,uneumann}@usc.edu
^{2} Shanghai Jiao Tong University, Shanghai, China
{0107nzy,sihengc}@sjtu.edu.cn

A Dataset Construction

In this work, we use two widely used autonomous driving benchmark: *nuScenes*[1] and *Argoverse*[2]. Here we describe how we derive local behavior data from each dataset for the experiments use.

A.1 nuScenes

nuScenes collects scene from four different places from Singapore and Boston, USA. The four places are labeled as *singapore-onenorh*, *boston-seaport*, *singapore-queenstown* and *singapore-hollandvillage* by the dataset. For each data split, we collect all the 2-second observed trajectories in the four places separately to build the behavior database $\mathcal{D}_{\mathcal{B}}$. Since the sample rate of nuScenes is 2Hz, each 2-second observed trajectory suppose to have two-dimensional coordinates at 5 timestamps. However, in the dataset, there are missing timestamps for some observed trajectories. To make the learning procedure more stable, we only pick the observed trajectories those have no missing data. Meanwhile, we also filter out the static trajectories (whose speed is lower than 2m/s). Table 1 shows the size of the behavior database for each data split in nuScenes dataset.

Table 1. The number of historical observed trajectories contained in the behavior database for each location in *nuScenes* dataset. We build the database for each split separately to simulate the real-world scenario.

Location Label	train	val	test
singapore-onenorth	92893	54878	50570
boston-seaport	708527	226813	163812
singapore-queenstown	59702	3696	26568
singapore-hollandvillage	92924	N/A	7340

* Corresponding authours

A.2 Argoverse

Argoverse collects data from Pittsburgh and Miami. We build the behavior $\mathcal{D}_{\mathcal{B}}$ for each city of each data split using the all 2-second observed trajectories correspondingly. In argoverse, since the sample rate is 10Hz, for each 2-second observed trajectory, there are 20 timestamps geometric position data. Similar to what we do for nuScenes, we only pick the observed trajectory that has no unavailable timestamp and whose average speed is larger than 2m/s. The statistic of the behavior database size is shown in Table 2

Table 2. The number of historical observed trajectories contained in the behavior database for each location in *Argoverse* dataset.

Location	train	val	test
Miami	1532574	281809	538004
Pittsburg	980827	162470	583311

B Implementation

We pick three SOTA methods for implementation. The implemented works in our paper are all based on existing official code packages. We **only modify the scene encoder related modules while remain the other parts exactly the same** as the baseline methods, including the decoders and the encoders for HD maps and motion data. For the LBF training, the network parameters are randomly initialized by pytorch, i.e. we did not use LBA teacher network as the initialize ion.

B.1 LaneGCN[5]

Implementation details: We upgrade LaneGCN to Local-Behavior-Aware and Local-Behavior-Free framework based on their official code (https://github.com/uber-research/LaneGCN). During the training, on both nuScenes and Argoverse dataset, we use 2 NVIDIA GeForce RTX 3090 GPUs with the batch size of 32. The initial learning rate is 1e-3 and will decay to 1e-4 at epoch 32. The total training epoch number is 36. We replicate the exact training scheduler of the LaneGCN listed in the readme file of their official code package. We use $\epsilon = 0.5$ and $\lambda_{kd} = 1.5$ as the default setting.

Dataset Preprocessing: For the argoverse dataset, we use their official code to generate the preprocessed data, including the lane graph and the motion data. For the nuScenes dataset, we follow the instruction in their paper[5], generating the lane graph using the lane information from nuScenes dataset. In the nuScenes, besides the lane information, there are also other map objects like sidewalks. Because in the LaneGCN paper, there is no indication for how to process those map objects, we decide to ignore them during our data preprocessing. It may explain that why the baseline performance of the LaneGCN on nuScenes (See Table 1 in the paper) is not as good as it shows on Argoverse. It also explains by the boost brought by the local behavior data is significant even compared to other methods: local behavior data provide extra complementary information to the system.

Network Architecture: We show the architecture of the implementation on LaneGCN based framework in Figure 1.

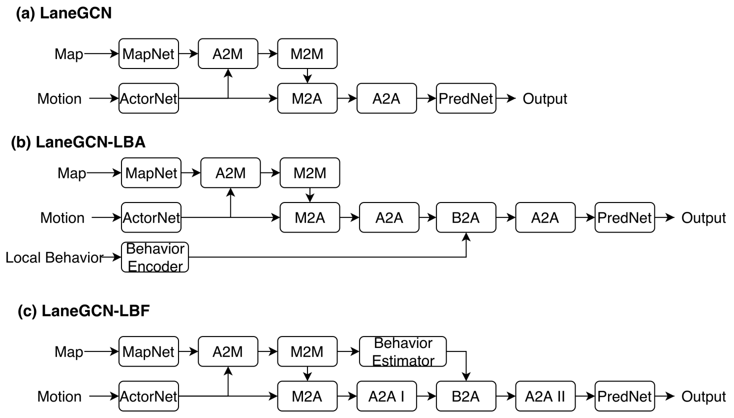

Fig. 1. LaneGCN Network Architecture

In the network architecture drawn in the figure, B2A is the auxiliary fusion module for LBF framework. When applied knowledge distillation loss, we use the output of B2A, output of A2A II from the LBF as $\mathbf{F}_s$ and the output of B2A and the output of A2A as $\mathbf{F}_t$. In the Table 4 of the paper, to do the abalation study, we pick an intermediate layer inside the PredNet as the third supervised feature for comparison.

Detailed inner structure of behavior encoder, behavior estimator and auxiliary fusion module B2A are in Figure 2

B.2 DenseTNT[4]

Implementation Details. In the experiment of DenseTNT, we adopt their official code package from https://github.com/Tsinghua-MARS-Lab/DenseTNT. We use $\epsilon = 0.5$ and $\lambda_{kd} = 1.5$ as the default setting. We use 8 NVIDIA GeForce RTX 3090 GPUs with a batch size of 64 for training. Different from their paper which claims a two-stage training strategy, in their official code, they train the network in a end-to-end style. We adopt the training strategy in the official code and set the learning rate with an initial value of 0.001 decays to 30% every 5 epochs. The total training epoch number is 30. The hidden size of the feature

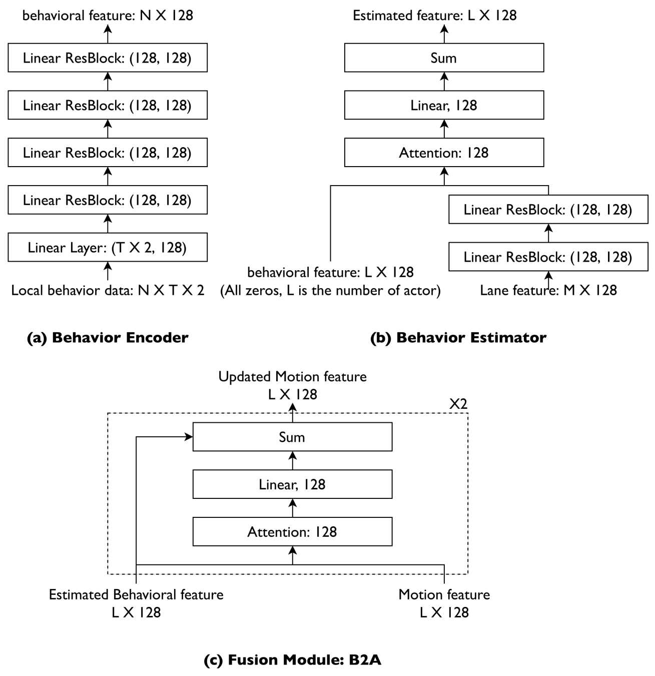
behavioral feature: N X 128
Linear ResBlock: (128, 128)
Linear ResBlock: (128, 128)
Linear ResBlock: (128, 128)
Linear ResBlock: (128, 128)
Linear Layer: (T X 2, 128)
Local behavior data: N X T X 2
(a) Behavior Encoder
Estimated feature: L X 128
Sum
Linear, 128
Attention: 128
Linear ResBlock: (128, 128)
Linear ResBlock: (128, 128)
behavioral feature: L X 128
(All zeros, L is the number of actor)
Lane feature: M X 128
(b) Behavior Estimator
Updated Motion feature
L X 128
X2
Sum
Linear, 128
Attention: 128
Estimated Behavioral feature
L X 128
Motion feature
L X 128
(c) Fusion Module: B2A

Fig. 2. Innter structure of important module in LaneGCN implementation

vectors is set to 128. The head number of our goal set predictor is 12. No data augmentation is used.

Dataset Preprocessing: We directly use their preprocess code to process the Argoverse dataset.

Network Architecture: We show the implemented architecture of DenseTNT in Figure 3. The inner architecture of behavior estimator and the behavior encoder are identical to the ones in the LaneGCN, which is shown in Figure 2. In the training of LBF framework ,$\mathbf{F}_s$ includes the output of Behavior estimator and the output of Dense goal encoder while $\mathbf{F}_t$ contains the target goal features of the Sparse context encoder and the output of the Dense goal encoder correspondingly.

B.3 P2T[3]

Implementation Details: The implementation of P2T is based on their published code package https://github.com/nachiket92/P2T. For the baseline result, we directly use their released pre-trained model which is included in their code package. We use one NVIDIA GeForce 1080 Ti GPU for training and the batch size is 32. P2T trains the network in a three-stage style. It first trains a reward network for 25 epochs with the learning rate as 1e-4 and then trains a coarse trajectory predictor for 100 epochs whose learning rate is 1e-3. Afterwards, it

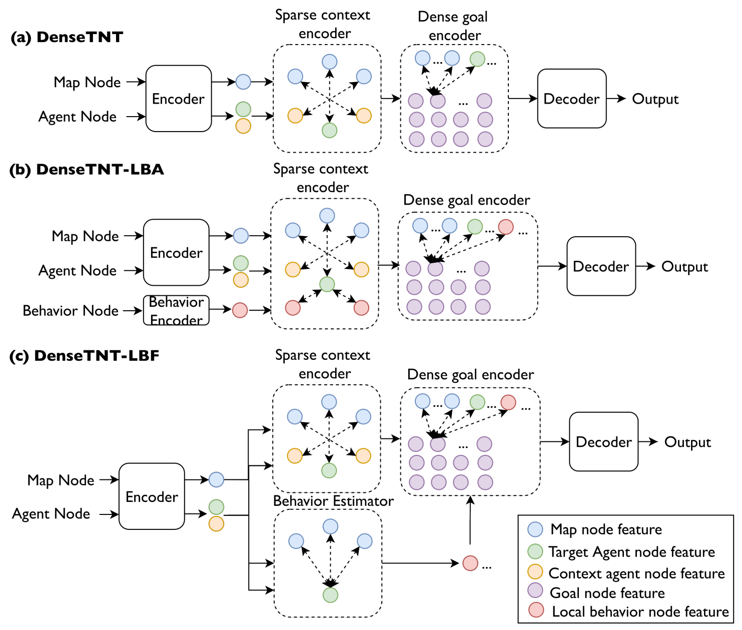

Fig. 3. DenseTNT Implementation Architecture

will train a finetuned trajectory predictor for 400 epochs with the learning rate of 1e-4. We directly follow the default training strategy stated in the official code package for the network training.

Dataset Preprocssing: We use the provided preprocess code to generate rasterization of map images for each data sample. For the behavior probability maps, we use the equation:

$$\mathcal{P}_{\mathcal{B}}^{(i,p)}(x,y) = \frac{\mathcal{P}_{\mathcal{B}}^{(i,p)}(x,y)}{\max(\mathcal{P}_{\mathcal{B}}^{(i,p)}(x,y))}, \tag{1}$$

which is stated in the paper to generate each probability map.

Network Architecture: P2T uses reinforcement learning to solve the prediction problem. Since we only modify the scene encoder part, in Figure 4, we only shows the detailed modification in the encoder part and skip the description of the reward model and the decoder. The behavior encoder is one linear layer with activation and the output channel is 16. The behavior estimator is implemented as stacked 3-layer-conv2D structure, the kernel size is 1 and the output feature channel is 16. During the training of LBF framework, we apply KD loss on the output of the behavior estimator as well as the input of the Decoder.

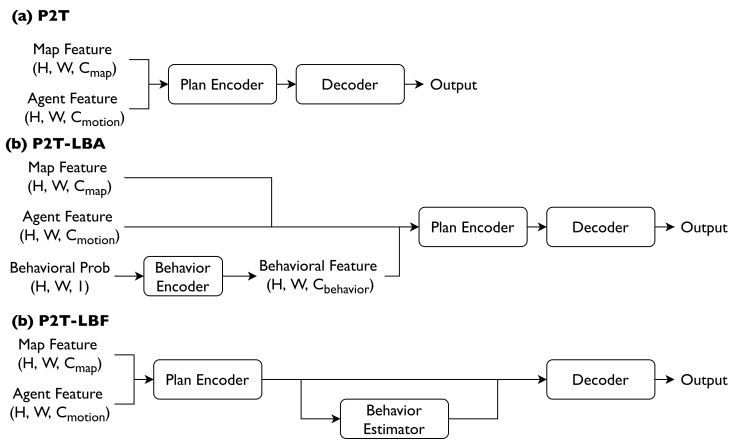
(a) P2T
Map Feature (H, W, C_map)
Agent Feature (H, W, C_motion)
Plan Encoder
Decoder
Output
(b) P2T-LBA
Map Feature (H, W, C_map)
Agent Feature (H, W, C_motion)
Behavioral Prob (H, W, 1)
Behavior Encoder
Behavioral Feature (H, W, C_behavior)
Plan Encoder
Decoder
Output
(b) P2T-LBF
Map Feature (H, W, C_map)
Agent Feature (H, W, C_motion)
Plan Encoder
Behavior Estimator
Decoder
Output

Fig. 4. Implementation of P2T

C Extensive Discussion

C.1 Practicability.

When introducing a new data source to trajectory forecasting, researchers should evaluate its accessibility in real-world applications. Our experiments use a collection of available agents' observed trajectories in each dataset to build the behavior database $\mathcal{D}_{\mathcal{B}}$. The datasets used in this work both contain the *global coordinates* for each recorded trajectory. This property of the datasets enables us to mount the trajectories to the physical location on the global map, which is fundamental to the concept of *local behavior data*. Judging our framework in real-world applications, the global coordinates of the map and the agents are relatively easy to retrieve through techniques such as the Global Positioning System (GPS). Once the global coordinates are retrieved, gathering local behavior data in the real world becomes feasible.

Furthermore, we argue that in the real-world application, the benefit brought by the local behavior data will be even more significant, since the behavior data gathering procedure in can last longer time and collect more data. Despite the improvement brought by the local behavior data in our experiments on the published datasets, in the Argoverse training split, there are still 5% of agents have no available local behavior data.

C.2 Local Behavior Information Visualization

In this paper, we exploit the local behavior data from the perspective of agent current locations. To more clearly visualize the information hidden in the local behavior data, here we show visualizations of the local behavior data for every lane segment on the map, see Figure 5.

In the figure, we focus on the visualization of two types of the information, one is the average speed of the trajectories collected from each lane segment and the other is the ratio of the trajectories that show turning actions (including turning and the lane changing). From the visualization we can see besides the location-specific information that is hidden in the local behavior data, such as the speed limit, there are more interesting information that worth further exploit. Those information includes some implicit rules of human driving behaviors that are hard to be learnt by limited number of observed trajectories, such as the turning intentions are much higher for the agents on the lanes that have intersections ahead even if the lane does not directly lead to a right-turn lane or left-turn lane.

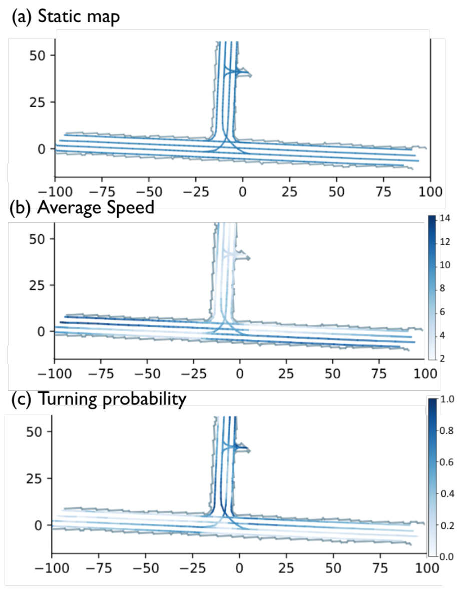

Fig. 5. Visualization of the local behavior data for each lane segment.

D Additional Experiments

D.1 Ablation study on LBF system design.

The LBF system has three factors: *if using* **i**) knowledge distillation (KD), **ii**) LBA as the teacher net in KD (to provide local behavior features guidance), **iii**) behavior estimator. Tab 3 studies each factor for LaneGCN on Argoverse, and it shows: i) A surpassed baseline (self-KD is effective); ii) B surpassed A (local behavior features from LBA network are effective), iii) C surpassed B (the behavior estimator is effective).

Method	KD	Behavior Estimator	minADE$_1$	minFDE$_1$
baseline	no teacher		1.71 -	3.78 -
A	self-KD		1.66$\downarrow$ 3%	3.67$\downarrow$ 3%
B	LBA as teacher		1.65$\downarrow$ 3%	3.63$\downarrow$ 4%
C (LBF)	LBA as teacher	✓	1.60$\downarrow$ 6%	3.54$\downarrow$ 6%

Table 3. Each of three factors counts validated on Argo test split

D.2 Comparison with memory-based method.

Table 4 shows that the proposed LBA/ LBF significantly outperform SOTA memory-based method, MANTRA[6], on Argoverse test split (numbers from its original paper). As mentioned in Sec 2, previous memory-based methods also leverage historical info, but three major differences mark the novelty of our work: **i**) we directly use historical trajectories as system input to avoid information loss, **ii**) we explicitly emphasize the spatial locality, **iii**) based on knowledge distillation, LBF does not need extra input while a memory-based method needs to store a huge memory bank.

Method	minADE$_1$ $\downarrow$	minFDE$_1$ $\downarrow$	minADE$_6$ $\downarrow$	minFDE$_6$ $\downarrow$
MANTRA	2.36 -	5.31 -	1.22 -	2.30 -
LaneGCN-LBA (ours)	**1.62** $\downarrow$ 30%	**3.58** $\downarrow$ 33%	**0.84** $\downarrow$ 31%	**1.30** $\downarrow$ 43%
LaneGCN-LBF (ours)	**1.60** $\downarrow$ 32%	**3.54** $\downarrow$ 33%	**0.85** $\downarrow$ 30%	**1.31** $\downarrow$ 43%

Table 4. Proposed LBA and LBF systems outperform MANTRA.

D.3 Performance gains along the time

In our paper, we only use the a few second local behavior data to avoid data snooping and for fair comparison. But in practice, by "local," local behavior data just means the data's starting position is at an agent's current location; such data can be a long trajectory reflecting long-term behavior. We want to demonstrate the potential of the local behavior data for long-term prediction in real world application by using fig 8 to show that more gain from local behavior data as prediction time goes on. One reason is that as real human behavior recordings, local behavior data can suppress error accumulation in a prediction model.

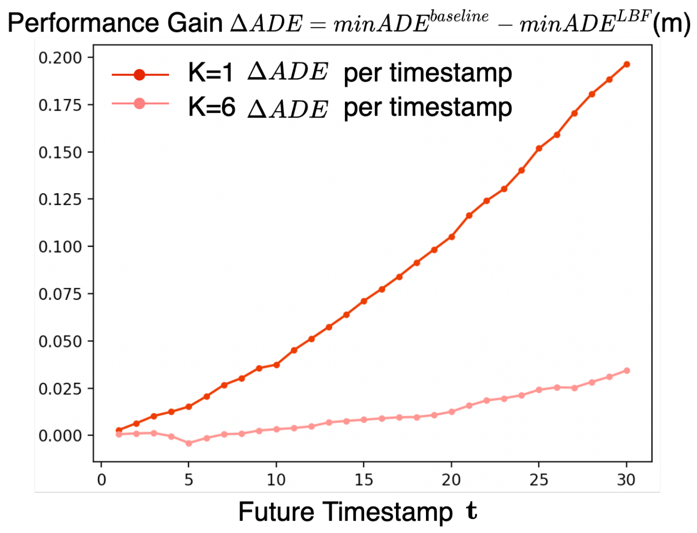

Fig. 8. Performance gain from LaneGCN-LBF compared to the baseline LaneGCN on Argoverse val set

References

1. Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: nuscenes: A multimodal dataset for autonomous driving. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11621–11631 (2020)
2. Chang, M.F., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., et al.: Argoverse: 3d tracking and forecasting with rich maps. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8748–8757 (2019)
3. Deo, N., Trivedi, M.M.: Trajectory forecasts in unknown environments conditioned on grid-based plans. arXiv preprint arXiv:2001.00735 (2020)
4. Gu, J., Sun, C., Zhao, H.: Densetnt: End-to-end trajectory prediction from dense goal sets. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 15303–15312 (2021)
5. Liang, M., Yang, B., Hu, R., Chen, Y., Liao, R., Feng, S., Urtasun, R.: Learning lane graph representations for motion forecasting. In: European Conference on Computer Vision. pp. 541–556. Springer (2020)
6. Marchetti, F., Becattini, F., Seidenari, L., Bimbo, A.D.: Mantra: Memory augmented networks for multiple trajectory prediction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7143–7152 (2020)