Diverse and Admissible Trajectory Forecasting through Multimodal Context Understanding

Supplementary Material

Seong Hyeon Park[1], Gyubok Lee[2], Jimin Seo[3⋆], Manoj Bhat[4⋆], Minseok Kang[5], Jonathan Francis[4,6], Ashwin Jadhav[4], Paul Pu Liang[4], and Louis-Philippe Morency[4]

[1] Hanyang University, Seoul, Korea
[2] Yonsei University, Seoul, Korea
[3] Korea University, Seoul, Korea
[4] Carnegie Mellon University, Pittsburgh, PA, USA
[5] Sogang University, Seoul, Korea
[6] Bosch Research Pittsburgh, Pittsburgh, PA, USA

A Additional Experimental Details

A.1 nuScenes Trajectory Extraction Process

nuScenes tracking dataset contains 850 different real-world driving records, each of which spans 40 frames (20 seconds) of LiDAR point-cloud data, RGB camera images, ego-vehicle pose records, and 3D bounding-box annotations of the surrounding vehicles, pedestrians, animals, etc. It also provides a map API that gives an access to the drivable area information. Based on this setting, we generate trajectories by associating the bounding boxes of the same agents, using the agent IDs available in the dataset. The resultant sequences, however, include noise and missing points, due to annotation errors and occlusion issues, as depicted in Fig. 1(b). To combat this, we employ *Kalman smoothing* [12] with the constant velocity linear model in Eq. (1), in order to filter the noise and impute missing points. When initializing the Kalman filter, we utilize *Expectation-Maximization* (EM) [1] to fit the initial states and the covariance parameters (transition and emission), with respect to the sequences.

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ z_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \\ \dot{z}_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ z_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \end{bmatrix} \tag{1}$$

Next, we construct the episodes for our trajectory dataset. The lengths of the past and prediction segments are set to 2 and 3 seconds (4 and 6 frames), respectively. As a result, an individual episode should contain a snippet of length 5 seconds (10 frames) from the smoothed trajectory, with a maximum of 30 samples extracted

⋆ Authors contributed equally

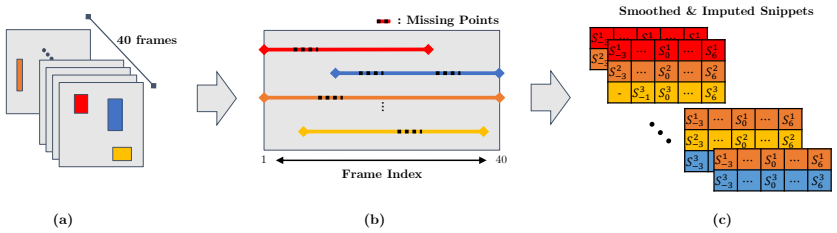

Fig. 1. Trajectory association, filtering and imputation process. (a), (b), (c) depict the bounding boxes in different frames, the associated and concatenated sequences, and the final trajectory snippets, respectively. Each agent is assigned a different color.

from one episode. For each snippet, we normalize the coordinate system, such that the ego-pose at the present time (4th frame) is placed at the origin $(0, 0)$.

For each episode, we generate the road mask of dimension 224×224 that covers $112 \times 112(m^2)$ area around the ego-pose, at the present time. The map API enables access to the drivable area information around the spot where the dataset is collected. Based on the ego-pose, we query the drivable area and save the returned information into a binary mask that is assigned 1's at the pixels belonging to the drivable area, and 0's at the other pixels.

A.2 Argoverse Motion Forecasting Data

Argoverse motion forecasting contains about 320,000 episodes, each of which spans 50 frames (5 seconds), along with a map API that gives an access to the drivable area information. All of the episodes are pre-processed for the trajectory forecasting task and, hence, additional extraction processes (such as association, smoothing, and imputation) are unnecessary. However, to condition the data samples to be as similar to the samples in the nuScenes trajectory (Section A.1) as possible, we down-sample each episode to episode lengths of 10 frames (5 seconds). We also normalize the coordinate system, such that the ego-agent at the present time is placed at the origin. As with nuScenes, we generate for each episode the road mask of dimension 224×224 which covers a $112 \times 112(m^2)$ area around the ego-agent at the present time.

A.3 CARLA Trajectory Extraction Process and Experiments

CARLA [6] is a vehicle and pedestrian behavior simulator wherein the agents follow the physical laws of motions driven by Unreal Engine. In order to validate the generalizability of our models, we synthesize trajectory forecasting data using CARLA vehicles and record the trajectory for all simulation time-steps. We make sure the data format is consistent with that of Argoverse and nuScenes. The data extracted from the simulator includes: vehicle trajectories, 2D birds-eye-view map images, lane center-lines, and LiDAR point-cloud data. The physics engine (Unreal Engine 4.19) provides limited capability for producing actual surface contour points, with the provided in-built CARLA ray-tracing LiDAR sensor. Therefore, we utilize the depth image sensor to extract accurate depth

Fig. 2. Visualizing predictions over simulated CARLA dataset at a snapshot of simulation-time with birds-eye-view image of drivable and non-drivable area and map. The RED points indicate ground truth, GREEN points indicate past trajectory points and BLUE indicates future trajectory points

information. We use four depth sensors, each with 90 deg field-of-view to capture depth images at each time-step. Then points are uniformly sampled on each image plane according to the number of channels and rotation frequency configuration of selected LiDAR. These points are corresponded with the pixel depth value by 2D bi-linear interpolation and projected into 3D space. The points are assigned a distance and an azimuth angle and converted from spherical coordinate to Cartesian coordinate system. This simulates precise object-shape contoured LiDAR point-cloud. Additional map Lane-center information is acquired from the map way-points, by maintaining a curvature at turning intersections.

In this paper, only birds-eye-view maps are used. These maps are utilized to build the distance transform of these images, as mentioned in Sec. 4.3 of the main paper. The drivable and non-drivable area information is extracted, based on the semantic segmentation of objects provided by the C++ API. The API is extended to specifically capture only the drivable area, and we utilize the Python API to save as a binary road mask with respect to ego-vehicle in simulation-time.

As the calibration parameters are not provided by the Unreal Engine physics kit, the intrinsic and extrinsic camera parameters and the distortion parameters are estimated by multiple camera view bundle adjustment. These parameters are used to transform segmentation images from a perspective view to an orthographic birds-eye-view. The maps are synchronized with the trajectory data 20 points in the past trajectory segment and 30 points in the future trajectory segments for all agents in a snapshot. In each map, we fix the number of agents to 5, including the ego-vehicle. Engineered data can be further generated on similar lines, for specialized edge-case handling of model. We will release our extended CARLA dataset which presents multiple data modalities, corresponding simulation and data extraction codes for our fellow researchers.

Performance Analysis: Model generalizability depends on both the map characteristics as well as the statistical distribution of trajectory samples provided by the dataset. The CARLA dataset has simpler cases compared to the the ones in Argoverse motion forecasting. But there are scenic disturbances, like parked and moving agents in the private property-areas, which is considered as non-drivable area. Even considering these cases, the model is able to predict valid admissible

Table 1. Results of baseline models and our proposed model in CARLA dataset. Local-CAM-NF is an ablation, whereas AttGlobal-CAM-NF is our full proposed model. The metrics are abbreviated as follows: minADE(**A**), minFDE(**B**), rF(**C**), DAO(**D**), DAC(**E**). Improvements indicated by arrows.

Model	CARLA				
	A (↓)	B (↓)	C (↑)*	D (↑)*	E (↑)*
LSTM	0.866	1.752	1.000	5.838	0.984
CSP	0.671	1.271	1.000	5.827	0.990
MATF-D	0.599	1.108	1.000	5.807	**0.992**
DESIRE	0.748	1.281	1.745	21.68	0.969
MATF-GAN	0.566	1.015	1.212	11.94	0.988
R2P2-MA	0.658	1.116	1.892	33.93	0.990
CAM	0.695	1.421	1.000	5.615	0.981
CAM-NF	0.543	1.006	2.135	**36.60**	0.979
Local-CAM-NF	0.483	0.874	**2.248**	33.94	0.986
Global-CAM-NF	0.490	0.898	2.146	33.00	0.986
AttGlobal-CAM-NF	**0.462**	**0.860**	2.052	30.13	0.987

trajectories, only influenced by the provided drivable area. We rigorously analyze the performance of our models in the CARLA dataset. The results are illustrated in Table 1 where our model outperforms in minADE and minFDE metrics by about 15% compared to baselines. CAM-NF model provides good performance in diversity metrics DAO illustrating its capability in improving diversity by using the flow-based decoder. From the results, our proposed AttGlobal-CAM-NF model is better able to mimic the true distribution of the autopilot in the simulator, showing considerable improvement in forecasting, over the baselines.

Table 2. Analysis on α on nuScenes.

Model	minADE (↓)	minFDE (↓)	rF (↑)	DAO (↑)	DAC (↑)
AttGlobal-CAM-NF(0.25)	0.775	1.349	2.310	**23.68**	0.912
AttGlobal-CAM-NF(0.5)	**0.639**	**1.171**	**2.558**	22.62	**0.918**
AttGlobal-CAM-NF(0.75)	0.666	1.246	2.431	23.04	0.914
AttGlobal-CAM-NF(1.0)	0.840	1.679	1.970	22.83	0.904

A.4 Analysis of α

We experiment with various values of α—a degradation coefficient of constant velocity. As shown in Table 2, models tend to perform better when trained with a certain range of α less than 1.0, reflecting that degrading the assumption of constant velocity (when $\alpha = 1.0$) prevents the model to avoid trivial solutions and encourages it to actively seek for necessary cues for predictions in non-constant velocity. Among values we experimented, we choose to set $\alpha = 0.5$ because the largest rF and DAC are observed in this setting, while remaining competitive in DAO, as well as the best precision in terms of minErrors of the prediction. This is a desirable property in our task of diverse and admissible trajectory forecasting. A possible future direction of our degradation approach is to replace

α with a learnable parametric function of noise centered at 1.0 that can perturb the physical model of the world, similarly discussed in [7].

B Scene Context Processing

B.1 The Drivable-area Map and Approximating p

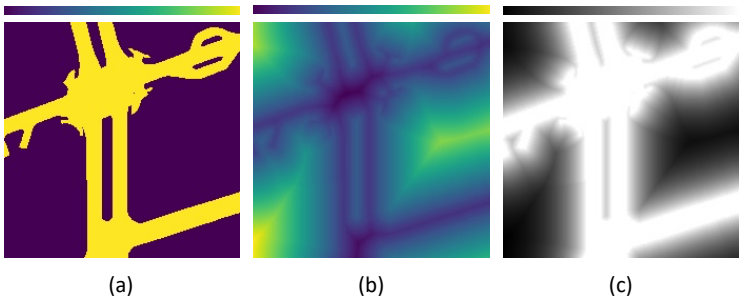

Fig. 3. (a) binary road mask, (b) the distance-transformed map, and (c) $\tilde{p}$ in nuScenes. The colorbars indicate the scales of the pixel value (increasing from left to right).

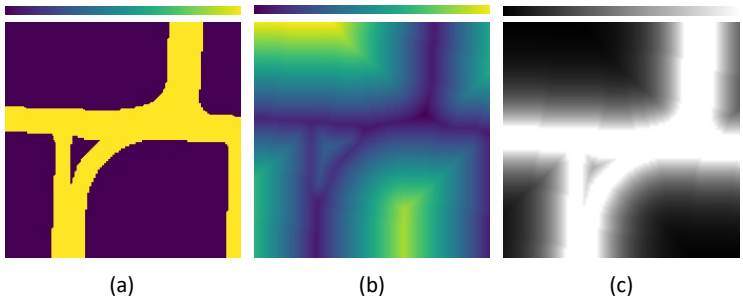

Fig. 4. (a) binary road mask, (b) the distance-transformed map, and (c) $\tilde{p}$ in Argoverse. The colorbars indicate the scales of the pixel value (increasing from left to right).

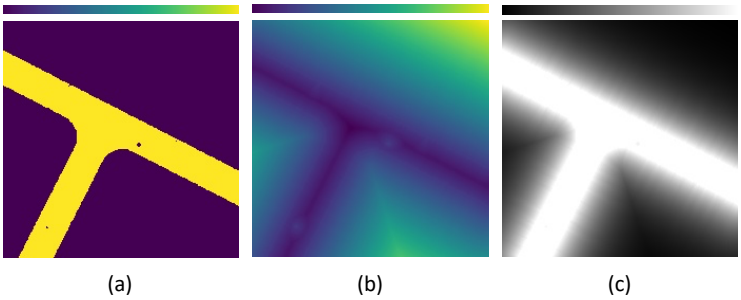

Fig. 5. (a) binary road mask, (b) the distance-transformed map, and (c) $\tilde{p}$ in CARLA. The colorbars indicate the scales of the pixel value (increasing from left to right).

We utilize the 224×224 binary road masks in Sec. A.1 and Sec. A.2 to derive $\tilde{p}$, which is used in the evaluation of the reverse cross-entropy in our loss function at Eq. (7), the main paper. Since $\tilde{p}$ is evaluated using 2×2 bilinear interpolation, the probability should be assigned to be gradually changing over the pixels, or the gradient would become 0 almost everywhere. To this end, we apply the distance transform to the binary mask, as depicted in Fig 3(b), Fig 4(b), Fig 5(b). We utilize this transformed map for deriving $\tilde{p}$, as well as the scene context input $\boldsymbol{\Phi}$. As described in the main paper, our $\tilde{p}$ is defined based on the assumptions that every location on the drivable-area is equally probable for future trajectories to appear in and that the locations on non-drivable area are increasingly less probable, proportional to the distance from the drivable area. Hence, we first set the negative-valued pixels (drivable region) in a the distance-transformed map to have 0s assigned and subtract each pixel from the maximum so that the pixel values in the drivable-area are uniformly greatest over the map. Then, we normalize the map with the mean and standard deviation calculated over the training dataset to smooth the deviations. Finally, we apply softmax over the pixels to constitute the map as a probability distribution $\tilde{p}$, as depicted in Fig 3(c), Fig 4(c), Fig 5(c).

B.2 Generating scene context input $\boldsymbol{\Phi}$

The scene context input $\boldsymbol{\Phi} \in \mathbb{R}^{224 \times 224 \times 3}$ is directly generated utilizing the the distance-transformed map. We augment 2 extra channels to the map, which embeds the unique pixel indices and Euclidean distances between each pixel and the center of the map. This way the ConvNet is enabled the access to the spatial position in the input thus it can extract the location-specific features from the scene context input. As similarly to the $\tilde{p}$, we normalize this augmented with the mean and standard deviation calculated over the training dataset to smooth the deviations.

C Additional Results Visualization

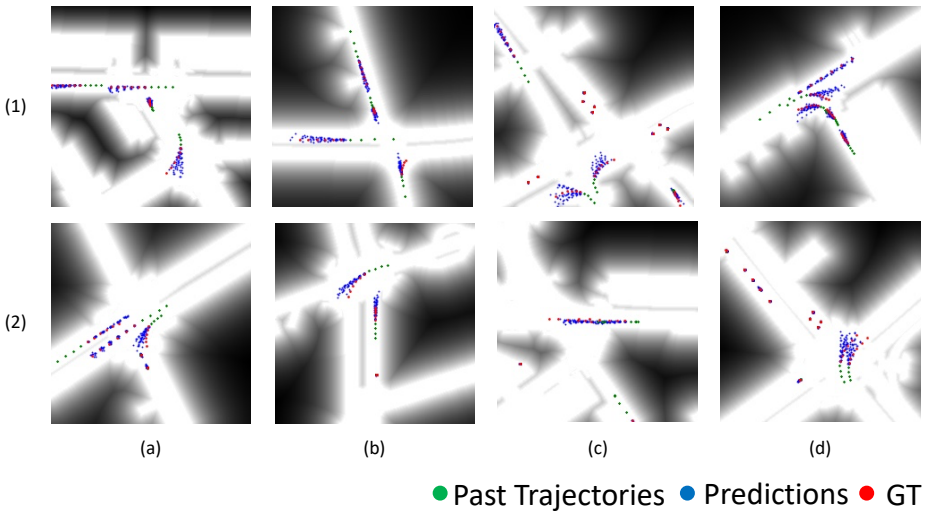

Fig. 6. Prediction results on nuScenes. Brighter background colors represent greater $\tilde{p}$ values. Our approach predicts future trajectories that show diversity while remaining admissible.

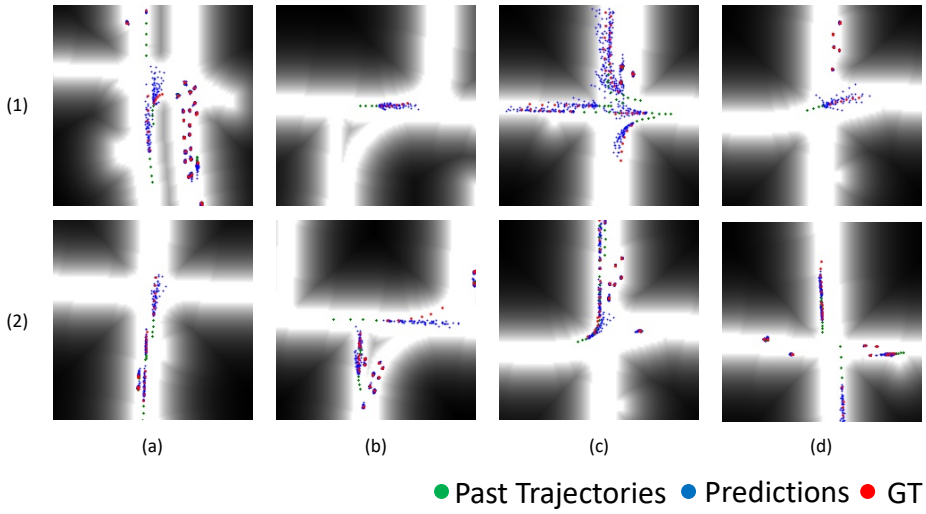

Fig. 7. Prediction results on Argoverse. Brighter background colors represent greater $\tilde{p}$ values. Our approach predicts future trajectories that show diversity while remaining admissible.

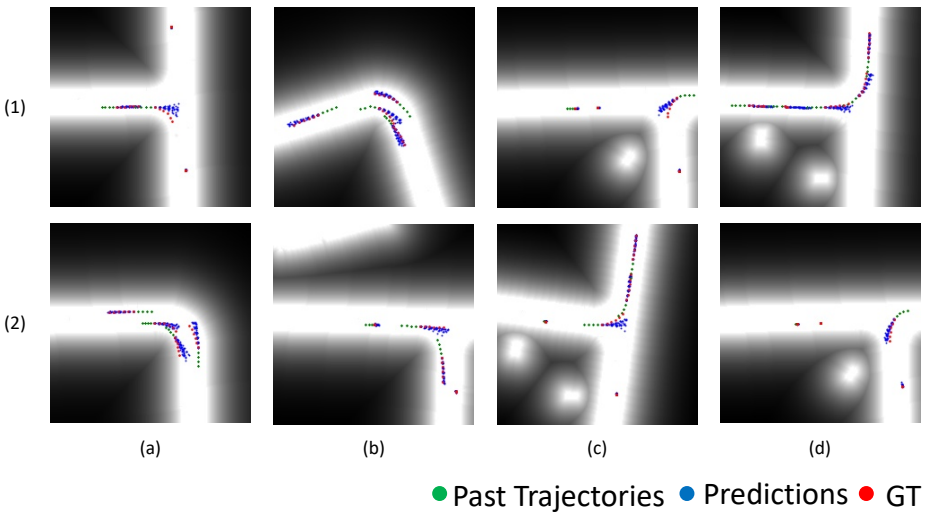

Fig. 8. Prediction results on CARLA. Brighter background colors represent greater $\tilde{p}$ values. Our approach predicts future trajectories that show diversity while remaining admissible.

Prediction results on nuScenes, Argoverse, and CARLA are illustrated in Fig 6, Fig 7, and Fig 8, respectively. In most cases throughout all three datasets, our model shows diverse and admissible predictions, even in the case of rapid left or right turns as in (2)(d) in Fig 6 and (1)(d) and (2)(d) in Fig 8. An clear insight on the quality of our model can be noticed when comparing predictions in cases of one path and multiple paths in front. For example, cases in (1)(b) and (2)(c) in Fig 6 show predictions are mostly going forward given the scene and the past trajectories going forward, whereas cases in (1)(c) and (2)(d) in Fig 6 show predictions are diverse given the shape of the road and the past trajectories that start to make turns. One weakness of our experimental setting is that it is quite challenging for our model to predict an explicit left or right turn given the past trajectory going only forward as illustrated in the bottom-most trajectory in (1)(b) in Fig 6 and (1)(a) in Fig 8. One possible feature that can be added to avoid such cases would be to incorporate encoded lane information in our cross-agent interaction module along with trajectory encodings.

D Architecture and Training Details

D.1 Training Setup

For all experiments, we implemented models using the `PyTorch` 1.3.1 framework. We performed all data extraction and model training on a desktop machine with a NVIDIA TITAN X GPU card. We directly utilized the default implementations of Adam optimizer [9] with the initial learning rate of 1.0E-04 and an adaptive scheduler that halves the learning rate with patience parameter 3 based on the sum of the avgADE and avgFDE in validation. The batch size is 64 for all baselines and the proposed model except AttGlobal-CAM-NF, where the batch size of 4 is used. We train the model for maximum 100 epochs, however, the early stopping is applied when the over-fitting of a model is observed.

D.2 Hyperparameters

Our experiments involve predicting 3 seconds of future trajectories of surrounding agents given a past record of 2 seconds, that includes the past trajectories and the scene context at the present time. The setting is common over all three datasets (nuScenes [4], Argoverse [5], and CARLA [6]) that we experiment. The trajectories are represented as 2D position sequences recorded at every 0.5 seconds. The hyperparameters of the network structure is described in Table 3 and Sec. D.3.

D.3 Network Details

In this section, we discuss the details on the network architecture of our model, which are not discussed in the main paper. The details of the the cross-agent attention and the agent-to-scene attention are particularly included in the discussion. We also give the edge case (e.g., bilinear interpolation at the position out of scene range) handling methods in our model. Refer to Table 3 for the hyperparameters used in the components of our model.

Table 3. Network Hyperparameters.

Operation	Input (dim.)	Output (dim.)	Parameters
		ConvNet	
(Every Conv2D is with 'same' padding, and followed by BN [8] & ReLU.)			
Conv2D	$\boldsymbol{\Phi}$ (64, 64, 3)	*conv*1 (64, 64, 16)	kernel:=(3,3)
Conv2D	*conv*1	*conv*2 (64, 64, 16)	kernel:=(3,3)
MaxPool2D	*conv*2	*pool*2 (32, 32, 16)	kernel:=(2,2), stride:=(2,2)
Conv2D	*pool*2	*conv*3 (32, 32, 32)	kernel:=(5,5)
Conv2D	*conv*3	*conv*4 (32, 32, 6)	kernel:=(1,1)
Dropout	*conv*3	$\boldsymbol{\Gamma}_g$ (32, 32, 32)	p:=0.5
UpSample2D	*conv*4	$\boldsymbol{\Gamma}_l$ (100, 100, 6)	mode:=bilinear
		Trajectory Encoding Module	
(Repeated for $a \in \{1, 2, ..., A\}$ with variable encoding time length $T^a_{past} \coloneqq t^a_s + 1$)			
Difference	S^a_{past} $(T^a, 2)$	dS^a_{past} $(T^a, 2)$	zero-pad (at the *start time*)
Fully-connected	dS^a_{past}	tS^a_t $(T^a, 128)$	activation:=Linear
LSTM	tS^a_t	h^a_0 (128)	zero initial states
		Cross-agent Interaction Module	
(Repeated for $a \in \{1, 2, ..., A\}$)			
LayerNorm	h^a_0 (128)	h^a_{ln} (128)	Layer Normalization [2]
Fully-connected	$h^a_{ln} \forall a \in \{1, 2, ..., A\}$	$\boldsymbol{Q}$ $(A, 128)$	activation:=Linear
Fully-connected	$h^a_{ln} \forall a \in \{1, 2, ..., A\}$	$\boldsymbol{K}$ $(A, 128)$	activation:=Linear
Fully-connected	$h^a_{ln} \forall a \in \{1, 2, ..., A\}$	$\boldsymbol{V}$ (A, 128)	activation:=Linear
Attention	$Q^a, \boldsymbol{K}, \boldsymbol{V}$	h^a_{atn} (128)	Scaled dot-product attention
Addition	h^a_0, h^a_{atn}	$\tilde{h}^a$ (128)	$\tilde{h}^a = h^a_0 + h^a_{atn}$
		Local Scene Extractor	
(Repeated for $a \in \{1, 2, ..., A\}$)			
Bilinear	$\boldsymbol{\Gamma}_l, \hat{S}^a_{t-1}$ (2)	γ^a_t (6)	2×2 Bilinear Interpolation
Concatenate	$\tilde{h}^a, \gamma^a_t$	hg^a_t (134)	-
Fully-connected	hg^a_t	fhg^a_t (50)	activation:=Softplus
Fully-connected	fhg^a_t	lc^a_t (50)	activation:=Softplus
		Agent-to-scene Interaction Module	
(Repeated for $a \in \{1, 2, ..., A\}$ with the fixed decoding time length 6)			
Flatten	$\hat{S}^a_{0:t-1}$ $(t, 2)$	$f\hat{S}^a_{0:t-1}$ (12)	zero-pad (at the last), $\hat{S}^a_0 \coloneqq S^a_0$
GRU	$f\hat{S}^a_{0:t-1}$	$\hat{h}^a_t$ (150)	zero initial state
Fully-connected	$\hat{h}^a_t$	fh^a_t (150)	activation:=Linear
Fully-connected	$\boldsymbol{\Gamma}_g$	$f\boldsymbol{\Gamma}$ (32, 32, 150)	activation:=Linear
Attention	$fh^a_t, f\boldsymbol{\Gamma}$	$\alpha\boldsymbol{\Gamma}^a_t$ (32, 32, 1)	Additive attention
Pool	$\boldsymbol{\Gamma}_g, \alpha\boldsymbol{\Gamma}^a_t$	$\tilde{\gamma}^a_t$ (32)	Pixel-wise sum ($\boldsymbol{\Gamma}_g \odot \alpha\boldsymbol{\Gamma}^a_t$)
Concatenate	$\tilde{\gamma}^a_t, \hat{h}^a_t, lc^a_t$	gc^a_t (232)	-
		Flow Decoder	
(Repeated for $a \in \{1, 2, ..., A\}$)			
Fully-connected	gc^a_t	fgc^a_t (50)	activation:=Softplus
Fully-connected	fgc^a_t	$ffgc^a_t$ (50)	activation:=Tanh
Fully-connected	$ffgc^a_t$	$\hat{u}^a_t$ (2), $\hat{\sigma}^a_t$ (2, 2)	activation:=Linear
Expm	$\hat{\sigma}^a_t$	σ^a_t (2, 2)	impl. in [3]
Constraint	$\hat{\mu}^a_t, \hat{S}^a_{t-2:t-1}$	μ^a_t(2)	$\alpha \coloneqq 0.5$
Random.	-	z^a_t(2)	$z^a_t \sim \mathcal{N}(0, I)$
Transform	$z^a_t, \mu^a_t, \sigma^a_t$	S^a_t(2)	$S^a_t = \sigma^a_t z^a_t + \mu^a_t$

We first discuss the details in the *encoder*. As described in the main paper, the encoder is made of the trajectory encoding module and cross-agent interaction module. The trajectory encoding module utilizes a linear transform and a single layer LSTM to encode the observation trajectory for each agent $S^a_{past} \equiv S^a_{t^a_s:0}$. To encode the observation trajectory S^a_{past}, we first normalize it to dS^a_{past} by taking the differences between the positions at each time step to have a translation-robust representation. Then, we linearly transform the normalized sequence dS^a_t to a high dimensional vector tS^a_t with Eq. (2) then the vector is fed to the LSTM with Eq. (3) to update the state output h^a_t and the cell state c^a_t of the LSTM.

$$tS_t^a = \textsc{Linear}_1(dS_t^a) \tag{2}$$
$$h_{t+1}^a, c_{t+1}^a = \text{LSTM}(tS_t^a, h_t^a, c_t^a) \tag{3}$$

Wrapping Eq. (2) and Eq. (3), along with the trajectory normalization process into a single function, we get Eq. (1) in the main paper, where the same equation is copied as Eq. (4) for the convenience. Note that the initial states of the LSTM are set as zero-vectors.

$$h_{t+1}^a = \text{RNN}_1(S_t^a, h_t^a) \tag{4}$$

The cross-agent interaction module gets the set of agent embeddings $\boldsymbol{h}_0 = \{h_0^1, ..., h_0^A\}$ and models the agent-to-agent interaction via an attention architecture inspired by "self-attention" [10]. For each agent encoding h_0^a, we first apply *Layer Normalization* [2] to get the normalized representation h_{ln}^a in Eq. (5) and calculate the query-key-value triple (Q^a, K^a, V^a) by linearly transforming h_{ln}^a with Eq. (6)-(8). Note that we empirically found that Layer Normalization gives a slight improvement to the model performance.

$$h_{ln}^a = \textsc{LayerNorm}(h_0^a) \tag{5}$$

$$Q^a = \textsc{Linear}_Q(h_{ln}^a) \tag{6}$$
$$K^a = \textsc{Linear}_K(h_{ln}^a) \tag{7}$$
$$V^a = \textsc{Linear}_V(h_{ln}^a) \tag{8}$$

Next, we calculate the attention weights $\boldsymbol{w}^a \equiv \{w_1^a, w_2^a, ..., w_A^a\}$ by calculating *scaled dot-product* between the query Q^a and each key in the set of keys $\boldsymbol{K} \equiv \{K^1, K^2, ..., K^A\}$ with Eq. 9 and taking softmax over the set of products with Eq. 10.

$$\text{SD}_{a,a'} = \frac{Q^a \cdot K^{a'}}{\sqrt{dim(Q^a)}} \tag{9}$$
$$\boldsymbol{w}^a = \textsc{Softmax}(\{\text{SD}_{a,1}, \text{SD}_{a,2}, ..., \text{SD}_{a,A}\}) \tag{10}$$

Finally, we get the interaction feature $\textsc{Attention}_1(Q^a, \boldsymbol{K}, \boldsymbol{V})$ in Eq. (2) of the main paper, by taking the weighted average over $\boldsymbol{V} \equiv \{V^1, V^2, ..., V^A\}$ using the weights $\boldsymbol{w}^a$ with Eq. (11).

$$\textsc{Attention}_1(Q^a, \boldsymbol{K}, \boldsymbol{V}) = \sum_{a'=1}^{A} w_{a'}^a V^{a'} = h_{atn}^a \tag{11}$$

We now discuss the details in *decoder*. As described in the main paper, the decoder is autoregressively defined and the previous outputs $\hat{S}_{0:t-1}^a$ are fed back.

We encode the previous outputs using a GRU. Since the GRU requires a static shaped input at each time step, we reconfigure $\hat{S}^a_{0:t-1}$ by flattening and zero-padding it with Eq. (12) and the reconfigured vector $f\hat{S}^a_{0:t-1}$ is fed to the GRU at each decoding step with Eq. (13).

$$f\hat{S}^a_{0:t-1} = \{\textsc{Flatten}(\hat{S}^a_{1:t-1}), 0, ..., 0\} \tag{12}$$

$$\hat{h}^a_t = GRU(f\hat{S}^a_{0:t-1}, \hat{h}^a_{t-1}) \tag{13}$$

Wrapping Eq. (12) and Eq. (13) to a single function, we get Eq. (3) in the main paper, where the same equation is copied as Eq. 14 for the convenience. Note that the initial state of the GRU is set as a zero-vector.

$$\hat{h}^a_t = \mathrm{RNN}_2(\hat{S}^a_{1:t-1}, \hat{h}^a_{t-1}) \tag{14}$$

The encoding $\hat{h}^a_t$ at each step is utilized in the agent-to-scene interaction module. The module is designed with an attention architecture inspired by "visual-attention" [11]. We first linearly transform $\hat{h}^a_t$ and each pixel Γ^i_g in the scene feature $\boldsymbol{\Gamma}_g \equiv \{\Gamma^1_g, \Gamma^2_g, ..., \Gamma^{HW}_g\}$ with Eq (15),(16).

$$f\hbar^a_t = \textsc{Linear}_{\hat{h}}(\hat{h}^a_t) \tag{15}$$

$$f\Gamma^i = \textsc{Linear}_{\Gamma_g}(\Gamma^i_g) \tag{16}$$

Then, we calculate the additive attention $\textsc{Attention}_2(\hat{h}^a_t, \boldsymbol{\Gamma})$ in Eq. (4) of the main paper by taking the addition of $f\hbar^a_t$ to each $f\Gamma^i$ followed by ReLU in Eq. (17), then applying softmax with Eq. (18).

$$\mathrm{SR}_{a,i} = \textsc{Relu}(f\hbar^a_t + f\Gamma^i) \tag{17}$$

$$\textsc{Attention}_2(\hat{h}^a_t, \boldsymbol{\Gamma}) = \textsc{Softmax}(\{\mathrm{SR}_{a,1}, \mathrm{SR}_{a,2}, ..., \mathrm{SR}_{a,HW}\}) = \alpha\boldsymbol{\Gamma}^a_t \tag{18}$$

Further details on our model's architecture and the hyperparameters, for instance the ConvNet and fully-connected layers discussed in the main paper are listed and defined in Table 3.

References

1. Abbeel, P.: Maximum likelihood (ml), expectation maximization (em). Course Note (2011), https://people.eecs.berkeley.edu/~pabbeel/cs287-fa11/slides/Likelihood_EM_HMM_Kalman-v2.pdf
2. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
3. Bernstein, D.S., So, W.: Some explicit formulas for the matrix exponential. IEEE Transactions on Automatic Control **38**(8), 1228–1232 (1993)

4. 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)
5. 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 Conference on Computer Vision and Pattern Recognition. pp. 8748–8757 (2019)
6. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: An open urban driving simulator. arxiv 2017. arXiv preprint arXiv:1711.03938 (2017)
7. Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al.: Noisy networks for exploration. arXiv preprint arXiv:1706.10295 (2017)
8. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
10. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
11. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. In: International conference on machine learning. pp. 2048–2057 (2015)
12. Yu, B.M., Shenoy, K.V., Sahani, M.: Derivation of kalman filtering and smoothing equations. Course Note (2004), `http://users.ece.cmu.edu/~byronyu/papers/derive_ks.pdf`