Supplementary Materials
SceneGen: Learning to Generate Realistic Traffic Scenes

Shuhan Tan[1,2*] Kelvin Wong[1,3*] Shenlong Wang[1,3]
Sivabalan Manivasagam[1,3] Mengye Ren[1,3] Raquel Urtasun[1,3]
[1]Uber Advanced Technologies Group [2]Sun Yat-Sen University [3]University of Toronto
tanshh@mail2.sysu.edu.cn {kelvinwong,slwang,manivasagam,mren,urtasun}@cs.toronto.edu

Abstract

In our supplementary materials, we detail SceneGen's model architecture and training procedure (Sec. 1). Additionally, we provide additional experiment details in Sec. 2 and additional experiment results in Sec. 3. In Sec. 4, we exhibit an extensive array of qualitative results that demonstrate the realism and diversity of the traffic scenes generated by SceneGen.

1. Additional Model Details

1.1. Input Representation

At each step of the generation process, SceneGen takes a bird's eye view multi-channel image encoding the HD map $\boldsymbol{m}$, the SDV $\boldsymbol{a}_0$, and the actors generated so far $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}\}$ in the SDV's egocentric coordinate system. The image emcompasses an $80m \times 80m$ region of interest centered on the SDV $\boldsymbol{a}_0$ and has a resolution of $0.25m$ per pixel, yielding a 320×320 image. This implies that the SDV always faces right in the image. The HD map is rasterized into a multi-channel image describing all available map elements in each dataset. For ATG4D, our multi-channel image consists of: lane polygons (straight vehicle lanes, dedicated right vehicle lanes, dedicated left vehicle lanes, dedicated bus lanes, and dedicated bike lanes); lane centerlines and dividers (allowed to cross, forbidden to cross, and maybe allowed to cross); lane segments (straight vehicle lanes, dedicated right vehicle lanes, and dedicated left vehicle lanes); drivable area and road polygons; and crosswalk polygons. In addition, we also encode each lane segment's traffic light state (green, yellow, red, flashing yellow, flashing red, and unknown), speed limit, and orientation as filled lane polygons. Note that orientation angles are encoded in their Biternion representations $\theta = (\cos\theta, \sin\theta)$ [17]. In aggregate, this yields a 24-channel image.

Argoverse provides a more limited set of map elements. Here, our multi-channel image consists of: lane polygons; lane centerlines (all lanes, left turn lanes, right turn lanes, intersection lanes, and traffic-controlled lanes); lane orientations (in Biternion representation); and drivable area polygons. In aggregate, this yields a 9-channel image.

To encode the actors $\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots$, we rasterize their bounding boxes onto a collection of binary occupancy images [1], one for each class; *i.e.*, SDV, vehicles, pedestrians, and bicyclists. Furthermore, we encode their headings and velocities by rasterizing their bounding boxes onto a five-channel image, filled with their respective speed, direction, and heading. As before, direction and heading angles are encoded in their Biternion representations. See Fig. 1 for an example.

1.2. Model Architecture

The basis of our model is the ConvLSTM architecture [20]. Let $\boldsymbol{x}^{(i)} \in \mathbb{R}^{C \times H \times W}$ denote the input multi-channel image at the i-th step of the generation process. Given the previous hidden and cell states $\boldsymbol{h}^{(i-1)}$ and $\boldsymbol{c}^{(i-1)}$, the new hidden states $\boldsymbol{h}^{(i)}$, cell states $\boldsymbol{c}^{(i)}$, and backbone features $\boldsymbol{f}^{(i)}$ are given by:

$$\boldsymbol{h}^{(i)}, \boldsymbol{c}^{(i)} = \text{ConvLSTM}(\boldsymbol{x}^{(i)}, \boldsymbol{h}^{(i-1)}, \boldsymbol{c}^{(i-1)}; \boldsymbol{w}) \tag{1}$$

$$\boldsymbol{f}^{(i)} = \text{CNN}_\text{b}(\boldsymbol{h}^{(i)}; \boldsymbol{w}) \tag{2}$$

*Indicates equal contribution. Work done at Uber ATG.

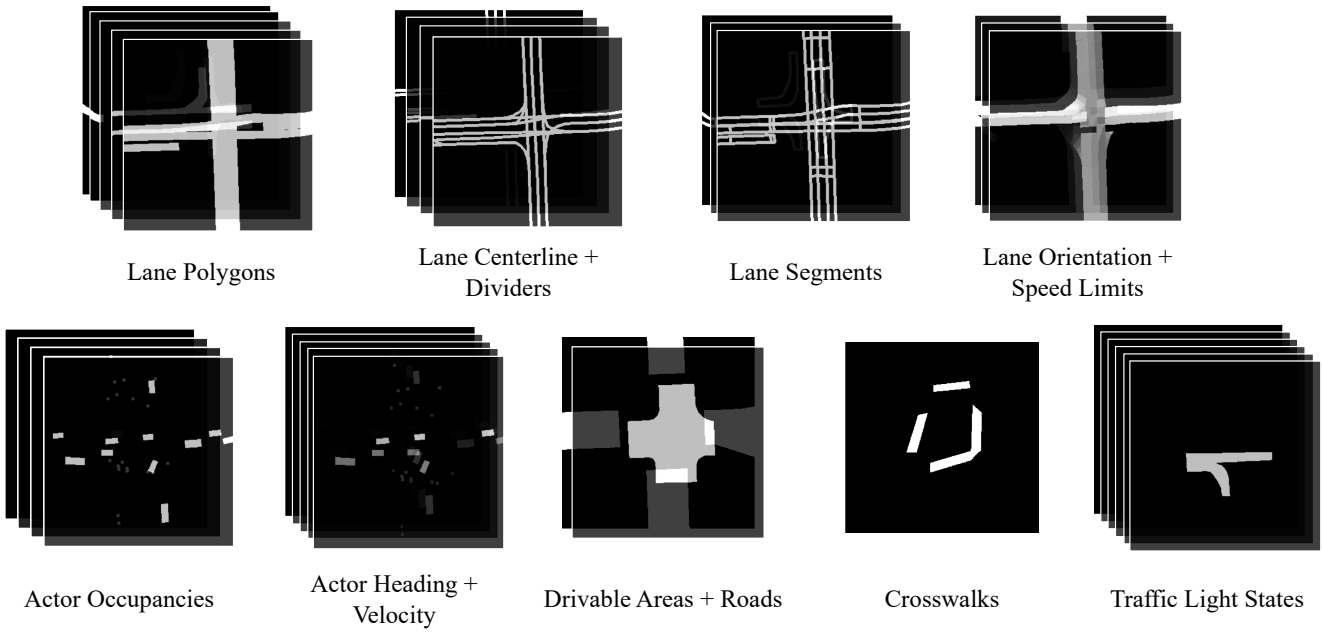

Figure 1: The input multi-channel image to SceneGen for ATG4D.

Here, ConvLSTM is a two-layer ConvLSTM with 5×5 convolution kernels and 32 hidden channels, and $\mathrm{CNN_b}$ is a five-layer convolutional neural network (CNN) with 32 feature channels per layer. Each convolution layer consists of a 3×3 convolution kernel, Group Normalization [24], and ReLU activations. The backbone features $\boldsymbol{f}^{(i)}$ summarize the generated scene so far and are given as input to the subsequent actor modules, which we detail next.

Class: We predict the class categorical distribution parameters $\boldsymbol{\pi}_c \in \Delta^{|\mathbb{C}|}$ as follows[1]:

$$\boldsymbol{\pi}_c = \mathrm{MLP_c}(\text{avg-pool}(\boldsymbol{f}^{(i)}); \boldsymbol{w}) \tag{3}$$

where avg-pool: $\mathbb{R}^{C \times H \times W} \rightarrow \mathbb{R}^C$ is average pooling over the spatial dimensions and $\mathrm{MLP_c}$ is a three-layer multi-layer perceptron (MLP) with 32 feature channels per hidden layer, ReLU activations, and softmax outputs.

Location: We apply uniform quantization to each actor's position and model the quantized values with a categorical distribution. Our quantization resolution is $0.25m$, which we found sufficient to generate realistic traffic scenes while balancing memory efficiency. To predict the parameters $\boldsymbol{\pi}_{\mathrm{loc}} \in \Delta^{H \times W - 1}$, we use a three-layer CNN with 32 feature channels per hidden layer. Each hidden convolution layer consists of a 3×3 convolution kernel, Group Normalization [24], and ReLU activations. The output convolution layer uses a 1×1 kernel with softmax activations. Note that we use separate CNN weights for each class in $\mathbb{C}$; *i.e.*, vehicles, pedestrians, and bicyclists.

Bounding box: An actor's bounding box $\boldsymbol{b}_i \in \mathbb{B}$ consists of its width and height $(w_i, l_i) \in \mathbb{R}^2_{>0}$ and its heading $\theta_i \in [0, 2\pi)$. We model the distribution over bounding box sizes with a mixture of K bivariate log-normal distributions whose parameters are predicted by a three-layer MLP (with the same architecture as described earlier):

$$[\boldsymbol{\pi}_{\mathrm{box}}, \boldsymbol{\mu}_{\mathrm{box}}, \boldsymbol{\Sigma}_{\mathrm{box}}] = \mathrm{MLP_{box}}(\boldsymbol{f}^{(i)}_{x_i, y_i}; c_i, \boldsymbol{w}) \tag{4}$$

where $\boldsymbol{\pi}_{\mathrm{box}} \in \Delta^{K-1}$ are mixture weights and each $\boldsymbol{\mu}_{\mathrm{box},k} \in \mathbb{R}^2$ and $\boldsymbol{\Sigma}_{\mathrm{box},k} \in \mathbb{S}^2_+$ parameterize a component log-normal distribution. To enforce the constraint that each $\boldsymbol{\Sigma} \in \mathbb{S}^2_+$, $\mathrm{MLP_{box}}$ predicts a variance term $\boldsymbol{\sigma}^2 \in \mathbb{R}^2_{>0}$ (in log-scale) and a

[1]We use $\Delta^n = \{(x_0, x_1, \ldots, x_n) \in \mathbb{R}^{n+1} | \sum_i x_i = 1 \text{ and } x_i \geq 0 \text{ for all } i\}$ to denote the n-simplex.

correlation term $\rho \in [-1, 1]$ (using tanh) such that:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix} \in \mathbb{S}_+^2 \tag{5}$$

Similarly, we model the distribution over heading angles with a mixture of K Von-Mises distributions whose parameters are predicted by another three-layer MLP:

$$[\boldsymbol{\pi}_\theta, \mu_\theta, \kappa_\theta] = \mathrm{MLP}_\theta(\boldsymbol{f}_{x_i,y_i}^{(i)}; c_i, \boldsymbol{w}) \tag{6}$$

where $\boldsymbol{\pi}_\theta \in \Delta^{K-1}$ are mixture weights and each $\mu_{\theta,k} \in [0, 2\pi)$ and $\kappa_{\theta,k} > 0$ parameterize a component Von-Mises distribution. Following Prokudin *et al*. [17], we parameterize each μ with its Biternion representation $\mu = (\cos\mu, \sin\mu)$ and each κ is predicted in log-scale. Note that we use separate MLP weights for each class in $\mathbb{C}$ whose actors are represented by bounding boxes; *i.e*., vehicles and bicyclists. Pedestrians are represented by their center of gravity only (*i.e*., location).

Velocity: Each of MLP_v, MLP_s, and MLP_ω is a three-layer MLP with the same architecture as described above. We parameterize the mixture of K Von-Mises distributions for directions ω just as we parameterize the distribution of headings. As before, we use separate MLP weights for each class in $\mathbb{C}$.

1.3. Training Details

We train our model to maximize the log-likelihood of real traffic scenes in our training dataset:

$$\boldsymbol{w}^\star = \arg\max_{\boldsymbol{w}} \sum_{i=1}^{N} \log p(\boldsymbol{a}_{i,1}, \ldots, \boldsymbol{a}_{i,n} | \boldsymbol{m}_i, \boldsymbol{a}_{i,0}; \boldsymbol{w}) \tag{7}$$

where $\boldsymbol{w}$ are the neural network parameters and N is the number of samples in our training set. We use teacher forcing and backpropagation-through-time to train through the generation process, up to a fixed window as memory allows. On a Nvidia Quadro RTX 5000 with 16GB of GPU memory, we train through 25 generation steps with batch size of 1 per GPU. We use PyTorch [15] and Horovod [19] to distribute the training process over 16 GPUs with a total batch size of 16. During training, we also randomly rotate each traffic scene with $\theta \in [0, 2\pi)$.

Note that each summand $\log p(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n | \boldsymbol{m}; \boldsymbol{w})$ can be decomposed into a sum of the log-likelihoods for each actors; namely, we have

$$\log p(\boldsymbol{a}_i | \boldsymbol{\xi}_i) = \underbrace{\log p(c_i | \boldsymbol{\xi}_i)}_{\text{class}} + \underbrace{\log p(x_i, y_i | c_i, \boldsymbol{\xi}_i)}_{\text{location}} + \underbrace{\log p(\boldsymbol{b}_i | c_i, x_i, y_i, \boldsymbol{\xi}_i)}_{\text{bounding box}} + \underbrace{\log p(\boldsymbol{v}_i | c_i, x_i, y_i, \boldsymbol{b}_i, \boldsymbol{\xi}_i)}_{\text{velocity}} \tag{8}$$

where $\boldsymbol{\xi}_i$ encapsulates the conditions on $\boldsymbol{a}_{<i}$, $\boldsymbol{m}$, and $\boldsymbol{a}_0$, to simplify notation. Therefore, the first summand $\log p(c_i | \boldsymbol{\xi}_i)$ is the (negative) cross-entropy loss between the predicted parameters $\boldsymbol{\pi}_\mathrm{c}$ and the ground truth class $c_i \in \mathbb{C} \cup \{\perp\}$. We describe the remaining summands in detail next.

Location: The second summand $\log p(x_i, y_i | c_i, \boldsymbol{\xi}_i)$ measures the log-likelihood the actor's location $(x_i, y_i) \in \mathbb{R}^2$. As discussed earlier, we uniformly quantize each actor's location and parameterize it with a categorical distribution. Therefore, $\log p(x_i, y_i | c_i, \boldsymbol{\xi}_i)$ is the (negative) cross-entropy loss between the predicted parameters $\boldsymbol{\pi}_\mathrm{loc}$ and the actor's ground truth quantized location. To address the significant imbalance of positive versus negative locations here, we use online negative hard mining. Specifically, we normalize $\boldsymbol{\pi}_\mathrm{loc}$ over the hardest 10,000 locations only (including the positive location), and compute $\log p(x_i, y_i | c_i, \boldsymbol{\xi}_i)$ based this restricted categorical distribution instead.

Bounding box: The third summand $\log p(\boldsymbol{b}_i | c_i, x_i, y_i, \boldsymbol{\xi}_i)$ is a sum of the log-likelihoods of the actor's bounding box size $(w_i, l_i) \in \mathbb{R}^2$ and heading $\theta_i \in [0, 2\pi)$:

$$\log p(\boldsymbol{b}_i | c_i, x_i, y_i, \boldsymbol{\xi}_i) = \log p(w_i, l_i | c_i, x_i, y_i, \boldsymbol{\xi}_i) + \log p(\theta_i | c_i, x_i, y_i, \boldsymbol{\xi}_i) \tag{9}$$

Since we model bounding box size with a mixture of K bivariate log-normal distributions, we have

$$\log p(w_i, l_i | c_i, x_i, y_i, \boldsymbol{\xi}_i) = \log \sum_{k=1}^{K} \pi_k \frac{1}{2\pi\sigma_{k,1}\sigma_{k,2}\sqrt{1-\rho_k^2}} e^{-\frac{1}{2(1-\rho_k^2)}\left[\left(\frac{\log w_i - \mu_{k,1}}{\sigma_{k,1}}\right)^2 + \left(\frac{\log l_i - \mu_{k,2}}{\sigma_{k,2}}\right)^2 + 2\rho_k\left(\frac{\log w_i - \mu_{k,1}}{\sigma_{k,1}}\right)\left(\frac{\log l_i - \mu_{k,2}}{\sigma_{k,2}}\right)\right]} \tag{10}$$

where $\boldsymbol{\pi} \in \Delta^{K-1}$ are mixture weights and each $\boldsymbol{\mu}_k \in \mathbb{R}^2$, $\boldsymbol{\sigma}_k \in \mathbb{R}^2_{>0}$, and $\rho_k \in [-1, 1]$ parameterize a component bivariate log-normal distribution.

Similarly, since we model heading angles with a mixture of K Von-Mises distributions, we have

$$\log p(\theta_i | c_i, x_i, y_i, \boldsymbol{\xi}_i) = \log \sum_{k=1}^{K} \pi_k \frac{e^{\kappa_k \cos(\theta_i - \mu_k)}}{2\pi I_0(\kappa_k)} \tag{11}$$

where $\boldsymbol{\pi} \in \Delta^{K-1}$ are mixture weights and each $\boldsymbol{\mu}_k \in [0, 2\pi)$ and $\kappa_k > 0$ parameterize a component Von-Mises distribution.

Velocity: The fourth summand $\log p(\boldsymbol{v}_i | c_i, x_i, y_i, \boldsymbol{b}_i, \boldsymbol{\xi}_i)$ is the log-likelihood of the actor's velocity $\boldsymbol{v}_i \in \mathbb{R}^2$, which we parameterize as $\boldsymbol{v}_i = (s_i \cos\omega_i, s_i \sin\omega_i)$ where $s_i \in \mathbb{R}_{\geq 0}$ is its speed and $\omega_i \in [0, 2\pi)$ is its direction. Recall that we model the distribution over an actor's velocity as a mixture model where one of the $K \geq 2$ components corresponds to $\boldsymbol{v}_i = 0$. Therefore, for $\boldsymbol{v}_i = 0$, we have

$$\log p(\boldsymbol{v}_i | c_i, x_i, y_i, \boldsymbol{b}_i, \boldsymbol{\xi}_i) = \log \pi_1 \tag{12}$$

and for $\boldsymbol{v}_i > 0$, we have

$$\log p(\boldsymbol{v}_i | c_i, x_i, y_i, \boldsymbol{b}_i, \boldsymbol{\xi}_i) = \log \sum_{k=2}^{K} \pi_k \underbrace{\frac{1}{\sigma_{s,k}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log s_i - \mu_{s,k}}{\sigma_{s,k}}\right)^2}}_{\text{speed}} \times \underbrace{\frac{e^{\kappa_{\omega,k}\cos(\omega_i - \mu_{\omega,k})}}{2\pi I_0(\kappa_{\omega,k})}}_{\text{direction}} \tag{13}$$

where $\boldsymbol{\pi} \in \Delta^{K-1}$ are mixture weights, each $\mu_{s,k} \in \mathbb{R}$ and $\sigma_{s,k} > 0$ parameterize a component log-normal distribution for speed s_i, and each $\mu_{\omega,k} \in [0, 2\pi)$ and $\kappa_{\omega,k} > 0$ parameterize a component Von-Mises distribution for direction ω_i.

2. Additional Experiment Details

2.1. Baselines

Prob. Grammar: Our Prob. Grammar baseline is inspired by recent work on probabilistic scene grammars [16, 9, 3]. Here, traffic scenes are composed by placing actors onto lane segments in the HD map, and initializing their classes, sizes, headings, velocities according to a hand-crafted prior. In our experiments, we use the following scene grammar:

$$\langle \text{Scene} \rangle \rightarrow \langle \text{Lanes} \rangle \tag{14}$$

$$\langle \text{Lanes} \rangle \rightarrow \langle \text{Lane} \rangle \langle \text{Lanes} \rangle | \epsilon \tag{15}$$

$$\langle \text{Lane} \rangle \rightarrow \langle \text{Actors} \rangle \tag{16}$$

$$\langle \text{Actors} \rangle \rightarrow \langle \text{Actor} \rangle \langle \text{Actors} \rangle | \epsilon \tag{17}$$

where Actor and ϵ are terminal symbols. Sampling from this scene grammar yields a *scene graph*, which defines the *scene structure*—where lane segments are and which actors are positioned on top of them—and *scene parameters*—the attributes of each lane segment and actor. In our setting, we are given the lane nodes (and the SDV actor's node) of the scene graph as a condition, and our goal is to insert/modify the actor nodes.

Drawing inspiration from MetaSim's probabilistic scene grammar [9], we first uniformly sample the maximum number of actors per lane segment and then place them along the lane centerline, with a random clearance between successive actors drawn from the exponential distribution. The class of each actor is determined by the lane segment under consideration (*i.e.*, car lane *vs*. bike lane); its lateral offset from the lane centerline is given by uniform noise; its bounding box size is sampled from a uniform distribution; its heading and the direction of its velocity is given by the direction of the lane segment plus some uniform noise; and its speed is the minimum of a sample from a uniform distribution and the lane segment's speed limit. The parameters of every distribution are tuned by hand.

	ATG4D			Argoverse		
Method	Size	Speed	Head	Size	Speed	Head
Prob. Grammar	0.49	0.42	0.30	0.41	0.57	0.38
MetaSim	0.49	0.33	0.14	0.50	0.53	**0.18**
Procedural	0.15	0.41	**0.07**	0.23	0.59	**0.17**
Lane Graph	0.33	0.28	0.16	0.31	0.34	0.38
LayoutVAE	0.16	0.40	0.29	0.21	0.46	0.29
SceneGen	**0.06**	**0.19**	0.08	**0.15**	**0.20**	0.22

Table 1: Vehicle-only maximum mean discrepency (MMD) results on ATG4D and Argoverse.

MetaSim: Our next baseline (MetaSim) uses a graph neural network (GNN) to transform the attributes of each actor node in the given scene graph. We use the implementation of Kar *et al*. [9] for this purpose. Specifically, given a scene graph drawn from Prob. Grammar, MetaSim deterministically modifies each actor's distance along its lane centerline, lateral offset, bounding box size, heading, and velocity. The inputs to MetaSim is a scene graph where each node's features are its attributes (normalized between 0 and 1 based on their respective minimum/maximum values under the prior), and the outputs of MetaSim are each node's new attributes (again normalized between 0 and 1). We use the GNN architecture of Kar *et al*. [9]: a three-layer GNN encoder with $32 \rightarrow 64 \rightarrow 128$ features channels and a three-layer GNN decoder with $128 \rightarrow 64 \rightarrow 32$ feature channels. Additionally, we use linear layers to encode and decode the per-node attributes.

Note that we train MetaSim using a supervised algorithm with heuristically generated ground truth scene graphs. In particular, given a real traffic scene, we first associate each actor to a lane segment; if this is not possible, the actor is not included in the scene graph. Next, we modify the attributes of each actor according to Prob. Grammar's prior. Finally, this modified scene graph is given as input to MetaSim, and we train MetaSim to transform the modified attributes back to their original ones. In our setting, this training process was both faster and more stable than the original unsupervised algorithm.

Procedural: Our Procedural baseline is inspired by methods that operate directly on the road topology of the traffic scene [22, 21, 7, 14]. Specifically, given a *lane graph* of the scene [11], Procedural uses a set of rules to place actors onto lane centerlines. First, we determine a set of valid routes traversing the entire lane graph. Each valid route is a sequence of successive lane centerlines along which actors can traverse without violating traffic rules; *e.g*., running red lights, merging onto an oncoming lane, *etc*. Next, we place actors onto each route such that successive actors maintain a random clearance (drawn from an exponential distribution) and no two actors collide. Each actor's bounding box size is sampled form a Gaussian KDE fitted to the training dataset, and its heading is determined by the tangent vector along its lane centerline at its location. Finally, we initialize the speed of each actor such that successive actors maintain a random time gap (drawn from an exponential distribution). Procedural is similar to the heuristics underlying [22, 21, 7] but generalized to handle arbitrary road topologies. Similar to Prob. Grammar, Procedural can generate only vehicles and bicyclists since existing HD maps do not provide sidewalks. We believe this limitation highlights the difficulty of using a heuristics-based approach.

Lane Graph: Inspired by MetaSim, we also consider a learning-based version of Procedural. Specifically, given a traffic scene generated by Procedural, we use a lane graph neural network to transform the attributes of each actor; *i.e*., location, bounding box size, heading, and velocity. Our lane graph neural network follows the design of the state-of-the-art motion forecasting model by Liang *et al*. [11]. It consists of MapNet for extracting map topology features and four fusion modules: actor-to-lane, lane-to-lane, lane-to-actor, and actor-to-actor. We train Lane Graph using heuristically generated ground truth, as in our MetaSim baseline.

LayoutVAE: Our implementation of LayoutVAE largely follows that of Jyothi *et al*. [8]. To adapt LayoutVAE to traffic scene generation, we first augment the original model with an additional CNN to extract map features. In particular, given a bird's eye view multi-channel image of the HD map, we use the backbone architecture of Liang *et al*. [12] to extract multi-scale map features, which we subsequently average-pool into a feature vector. This is then given to LayoutVAE as input in place of the label set encoding used in the original setting[2]. Our second modification enables LayoutVAE to output oriented

[2]The label set in our setting is fixed to be vehicles, pedestrians, and bicyclists.

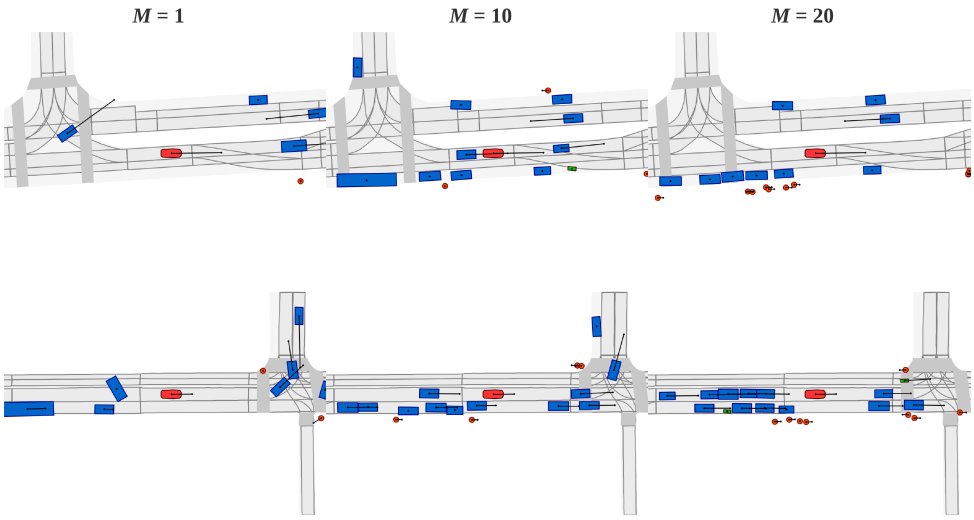

Figure 2: Traffic scenes generated by SceneGen using $M = 1, 10, 20$ sample proposals for ATG4D.

bounding boxes and velocities. Specifically, we replace the spherical quadrivariate Gaussian distribution of its BBoxVAE with a bivariate Gaussian distribution for location, a bivariate log-normal distribution for bounding box size, and a bivariate Gaussian distribution for velocity. To evaluate the log-likelihood of a scene, we use Monte-Carlo approximation with 1000 samples from the conditional prior [8].

2.2. MMD Metrics

To complement our likelihood-based metric, we compute a sample-based metric as well: maximum mean discrepancy (MMD) [4]. As we discussed in the main text, MMD measures a distance between two distributions p and q as

$$\text{MMD}^2(p, q) = \mathbb{E}_{x,x' \sim p}[k(x, x')] + \mathbb{E}_{y,y' \sim q}[k(y, y')] - 2\mathbb{E}_{x \sim p, y \sim q}[k(x, y)] \tag{18}$$

for some kernel k. Following [26, 13], we compute MMD using Gaussian kernels (with bandwidth $\sigma = 1$) with the total variation distance to compare scene statistics between generated and real traffic scenes. In particular, we first sample a set P of real traffic scenes from the evaluation dataset. Conditioned the SDV state and HD map of the scenes in P, we generate a set Q of synthetic scenes using the method under evaluation. Then, we approximate MMD as:

$$\text{MMD}^2(p, q) \approx \frac{1}{|P|^2} \sum_{x \in P} \sum_{x' \in P} k(x, x') + \frac{1}{|Q|^2} \sum_{y \in Q} \sum_{y' \in Q} k(y, y') - \frac{2}{|P||Q|} \sum_{x \in P} \sum_{y \in Q} k(x, y) \tag{19}$$

Our scene statistics measure the distribution of locations, classes, bounding box sizes (in m^2), speeds (in m/s), and heading angles (relative to that of the SDV) for each scene. Each distribution of location is a histogram of bird's eye view locations over an $80m \times 80m$ grid at a $5m$ resolution. Empty scenes are discarded since these scene statistics are undefined. Since MMD is expensive to compute, in ATG4D, we form P by sampling the evaluation dataset by every 25th scene, yielding approximately 5000 scenes. We compute MMD over the full Argoverse validation set as it contains 5015 scenes only.

We also compute MMD in the feature space of a pre-trained motion forecasting model. This is similar to some popular metrics for evaluating generative models such IS [18], FID [5], and KID [2], except we use a motion forecasting model as our feature extractor. Here, our motion forecasting model takes a bird's eye view multi-channel image of the actors in the scene and regresses the future locations of each actor over the next 3 seconds in 0.5s increments. We use the actor rasterization procedure described in Sec. 1.1 and the model architecture from [23], and we train the model using 4000 training log from the ATG4D training set. To obtain a feature vector summarizing the scene, we average pool the model's backbone features along its spatial dimensions. Then, to compute MMD, we use the RBF kernel with bandwidth $\sigma = 1$.

M	Feat.	Class	Size	Speed	Head
1	0.13	0.05	0.05	0.10	0.10
10	0.11	0.20	0.06	0.33	0.08
20	0.11	0.30	0.07	0.41	0.08

Table 2: Analysis of the number of sample proposals M on ATG4D. The reported numbers are the MMD metrics computed between distributions of features extracted by a motion forecasting model and various scene statistics (see main text).

	Vehicle		Pedestrian		Bicyclist	
Training Dataset	0.5	0.7	0.3	0.5	0.3	0.5
Real 250K	95.3	90.4	77.3	72.0	69.3	62.8
+ Sim 250K	95.7	90.7	78.2	72.9	70.4	64.3
+ Sim 500K	95.6	90.8	78.0	72.6	68.6	61.7
+ Sim 750K	95.6	90.6	77.5	72.2	67.7	59.1
+ Sim 1000K	95.6	90.6	77.3	72.0	66.4	58.6
+ Sim 1250K	95.4	90.4	76.9	71.6	65.4	57.2

Table 3: Data augmentation results on ATG4D. Starting from 250,000 real training LiDAR frames, we progressively add 250,000 SceneGen-simulated LiDAR frames to the training dataset. We evaluate detection AP at 0.5/0.7IoU for vehicles and 0.3/0.5IoU for pedestrians and bicyclists.

3. Additional Experiment Results

3.1. Vehicle MMD Metrics

In Tab. 1, we report vehicle-only MMD metrics for ATG4D and Argoverse. Specifically, we compute scene statistics for generated and real traffic scenes using vehicle actors only. As before, scenes with no vehicles are discarded during evaluation. This allows for an alternative comparison that controls for the class most easily handled by heuristics; *i.e.*, vehicles. Overall, we see that SceneGen still achieves the best results among the competing methods. This result reaffirms our claim that heuristics-based methods are insufficient to model the full complexity and diversity of real world traffic scenes.

3.2. Sampling Strategy Analysis

As discussed in the main text, SceneGen uses a sampling strategy inspired by *nucleus sampling* [6]. Specifically, at each generation step, we sample each of SceneGen's position, heading, and velocity distributions M times and return the most likely sample as output. In Tab. 2 and Fig. 2, we analyze the effects of using different numbers of sample proposals $M = 1, 10, 20$. We see that using $M > 1$ decreases MMD on deep features, indicating that scene-level realism is improved. This improvement is even more evident in Fig. 2, where we see vehicles disregarding the rules of traffic when $M = 1$. With more fine-grained tuning of M, we expect to see improvements in the actor-level statistics as well; *i.e.*, class, size, and speed.

3.3. Sim2Real Data Augmentation

In Sec. 4.4 of the main text, we demonstrated that SceneGen coupled with sensor simulation can generate realistic labeled data for training perception models. Our next experiment studies whether this simulated data can be used to *improve* the performance of the perception models via data augmentation. To this end, we train several 3D object detectors [25] with varying amounts of simulated LiDAR frames added to their training datasets: starting from 250,000 real LiDAR frames, we progressively add 250,000 simulated LiDAR frames. Note that each set of 250,000 LiDAR frames have the same underlying SDV state and HD map. Each detector is trained from scratch until convergence using the Adam optimizer [10] with a learning rate of $1\mathrm{e}{-4}$ with a batch size of 32, distributed over 4 GPUs.

From Tab. 3, we see that adding 250,000 SceneGen simulated LiDAR frames yields iprovements of 0.4% in AP@0.5IoU for vehicles, 0.9% in AP@0.3IoU for pedestrians, and 1.1% in AP@0.3IoU for bicyclists. Additional augmentation gave no further gains, which we hypothesize is due to a Sim2Real gap in sensor simulation and performance saturation.

4. Additional Qualitative Results

In Fig. 3 and 4, we present an array of additional qualitative results for ATG4D and Argoverse respectively. Here, we compare traffic scenes generated by SceneGen, MetaSim, Lane Graph, and LayoutVAE. From these visualizations, we see that SceneGen generates traffic scenes that best reflect the complexity and diversity of real world traffic scenes. For example, in the second-to-last row of Fig. 3, we show a traffic scene generated by SceneGen in which a vehicle performs a three-point turn. In the bottom row of Fig. 3, we also show a scene in which two bicyclists perform an left turn using the car lane. These scenes highlight SceneGen's ability to model rare but plausible traffic scenes that could occur in the real world.

In Fig. 5 and 6, we also showcase the diversity of traffic scenes that SceneGen is able to generate. Each row in the figures show four samples from our model when given the same SDV state and HD map as inputs. From these visualizations, we see that SceneGen captures the multi-modality of real world traffic scenes well. For example, the top row of Fig. 5 shows four traffic scenes generated for a four-way intersection. Here, we see samples in which pedestrians cross the intersection, vehicles perform an unprotected left turn, and a large bus goes straight.

Finally, in Fig. 7, we visualize the quantized location heatmaps for steps $t = 0, 5, 10, 15, 20$ of the generation process. Each row shows the categorical distribution from which we sample the next actor's location. From these visualizations, we see that SceneGen is able to model the distribution over actor locations (and the corresponding uncertainties) quite precisely. For example, the distribution over vehicle locations are concentrated around lane centerlines and the distribution over pedestrian locations are diffused over crosswalks and sidewalks.

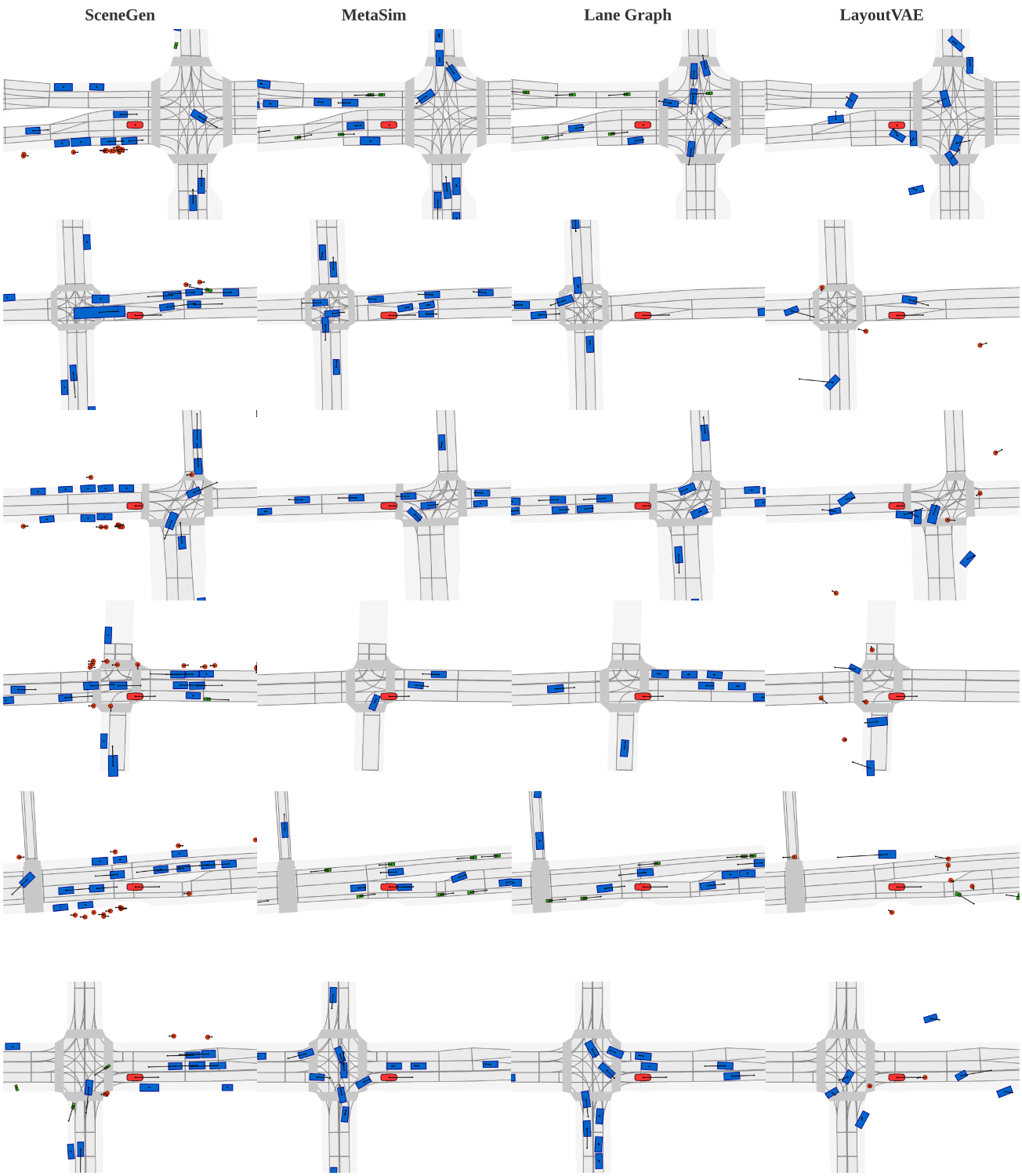

Figure 3: Qualitative comparison of traffic scenes generated by SceneGen and various baselines on ATG4D. The ego SDV is shown in red; vehicles in blue; pedestrians in orange; and bicyclists in green. We visualize lane segments and drivable surfaces in light grey and crosswalks in dark grey.

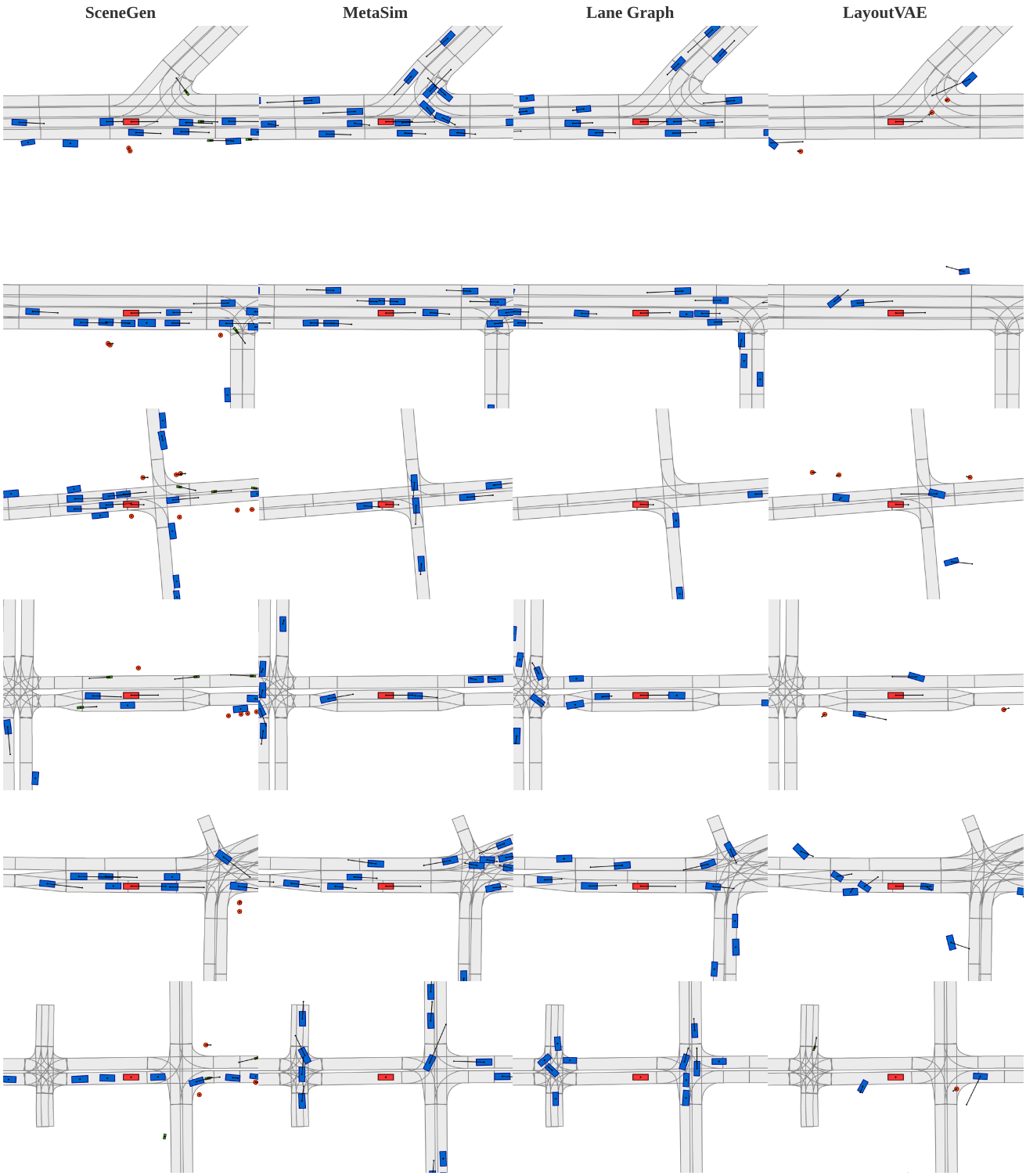

Figure 4: Qualitative comparison of traffic scenes generated by SceneGen and various baselines on Argoverse. The ego SDV is shown in red; vehicles in blue; pedestrians in orange; and bicyclists in green. We visualize lane segments in light grey.

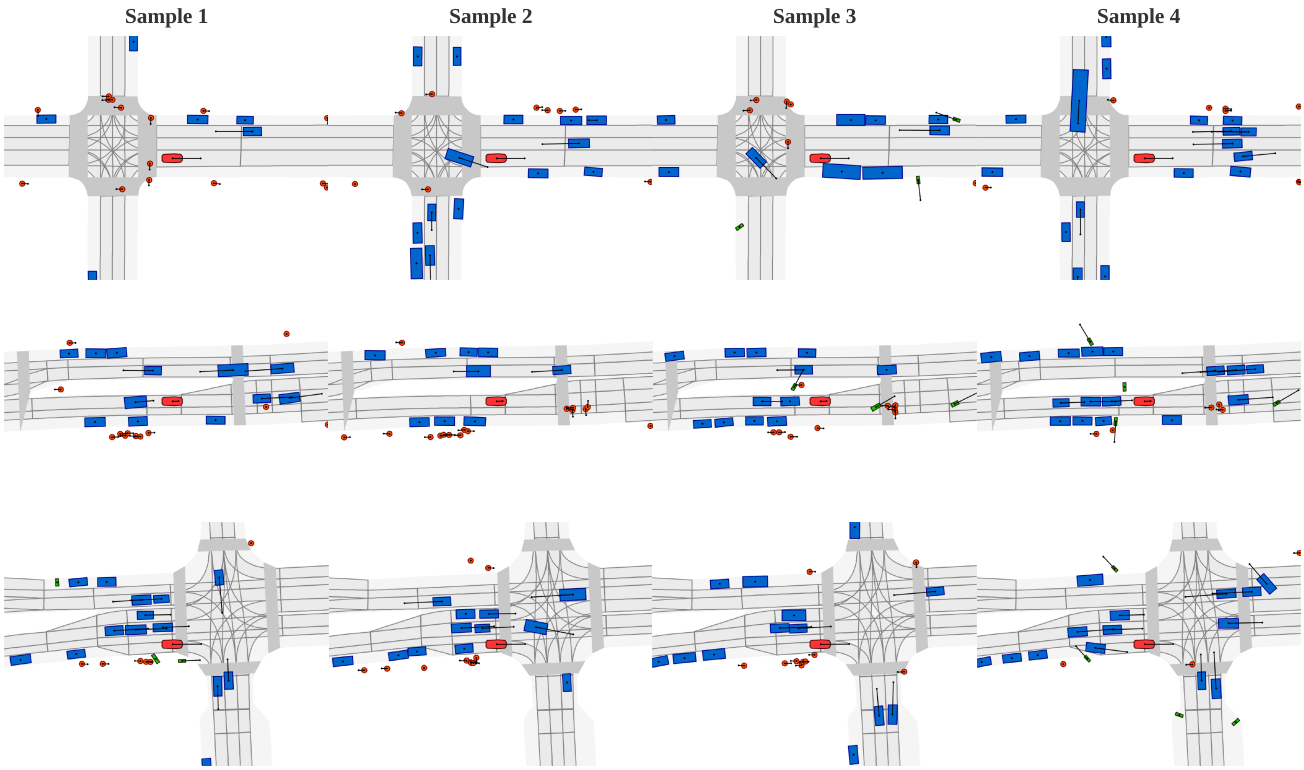

Figure 5: Traffic scenes generated by SceneGen on ATG4D. The traffic scenes in each row are generated from the same SDV state and HD map inputs. Each traffic scene is a distinct sample drawn from our model.

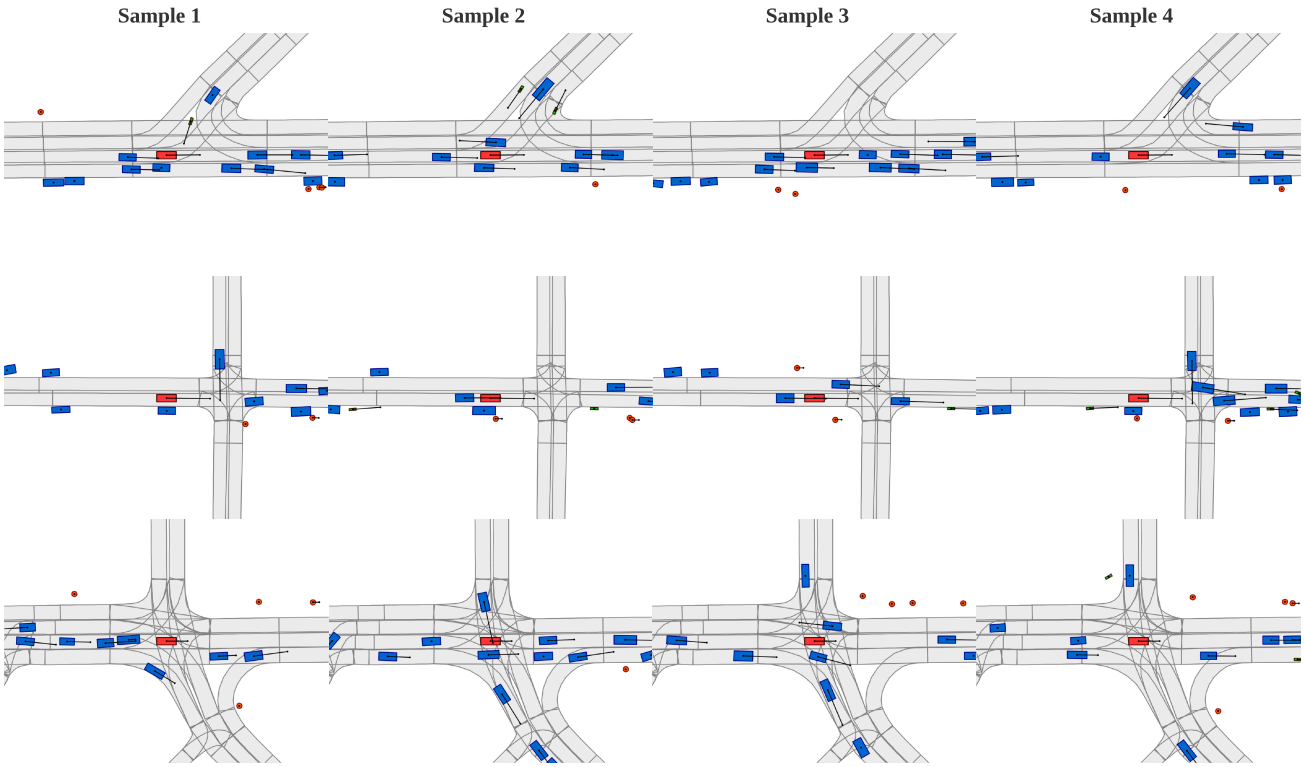

Figure 6: Traffic scenes generated by SceneGen on Argoverse. The traffic scenes in each row are generated from the same SDV state and HD map inputs. Each traffic scene is a distinct sample drawn from our model.

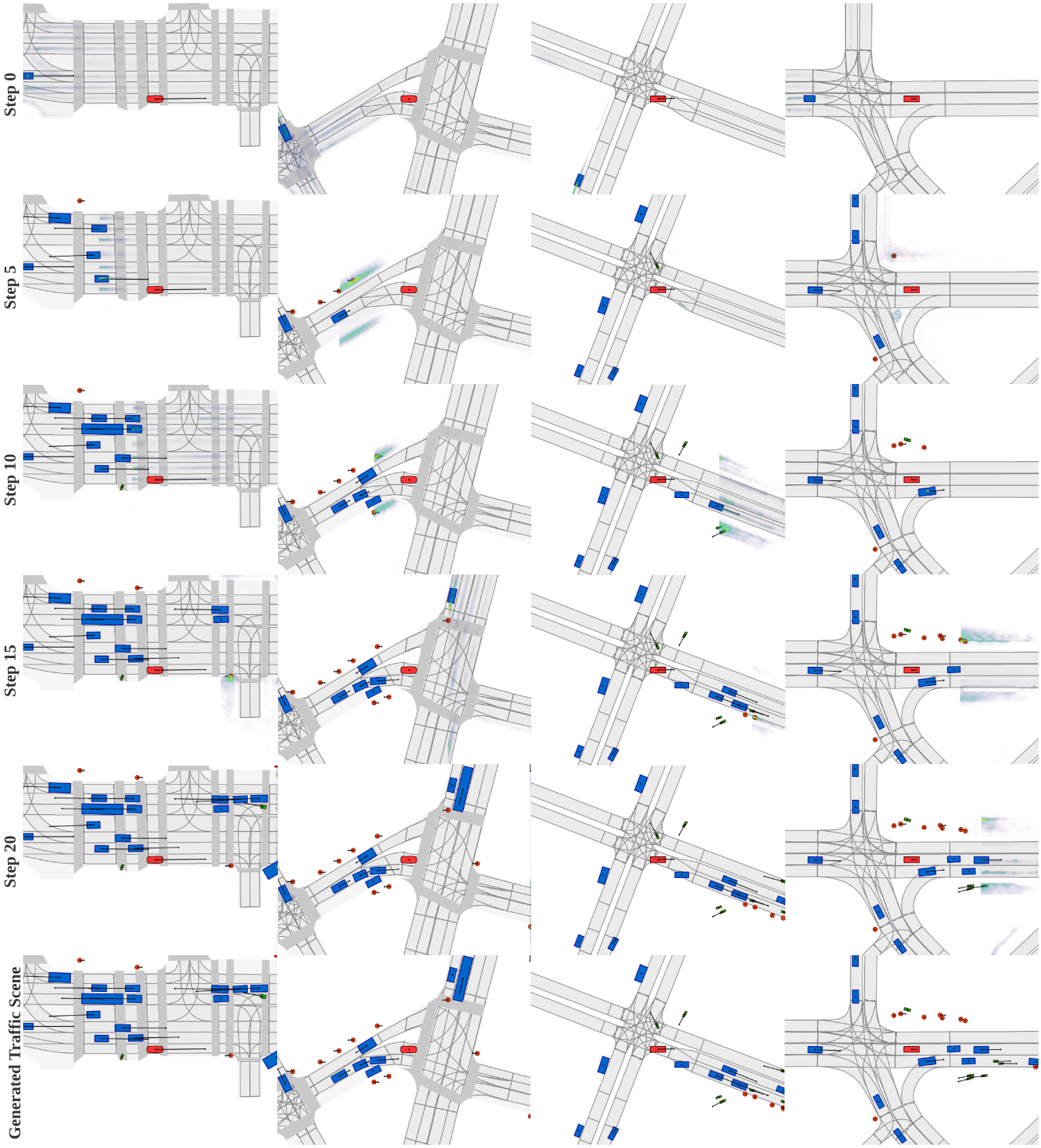

Figure 7: Traffic scenes generated by SceneGen on ATG4D (first two columns) and Argoverse (last two columns). We visualize the quantized location heatmap for steps $t = 0, 5, 10, 15, 20$ of the generation process. Each column represents the generation process for one traffic scene. Bright yellow means higher likelihood.

References

[1] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *RSS*, 2019.

[2] Mikolaj Binkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD gans. In *ICLR*, 2018.

[3] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. Meta-sim2: Unsupervised learning of scene structure for synthetic data generation. 2020.

[4] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel two-sample test. *JMLR*, 2012.

[5] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

[6] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *ICLR*, 2020.

[7] Stefan Jesenski, Jan Erik Stellet, Florian A. Schiegg, and J. Marius Zöllner. Generation of scenes in intersections for the validation of highly automated driving functions. In *IV*, 2019.

[8] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. Layoutvae: Stochastic scene layout generation from a label set. In *ICCV*, 2019.

[9] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *ICCV*, 2019.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[11] Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. Learning lane graph representations for motion forecasting. In *ECCV*, 2020.

[12] Ming Liang, Bin Yang, Wenyuan Zeng, Yun Chen, Rui Hu, Sergio Casas, and Raquel Urtasun. Pnpnet: End-to-end perception and prediction with tracking in the loop. In *CVPR*, 2020.

[13] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. Efficient graph generation with graph recurrent attention networks. In *NeurIPS*, 2019.

[14] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world. In *CVPR*, 2020.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[16] Aayush Prakash, Shaad Boochoon, Mark Brophy, David Acuna, Eric Cameracci, Gavriel State, Omer Shapira, and Stan Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. In *ICRA*, 2019.

[17] Sergey Prokudin, Peter V. Gehler, and Sebastian Nowozin. Deep directional statistics: Pose estimation with uncertainty quantification. In *ECCV*, 2018.

[18] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NeurIPS*, 2016.

[19] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv*, 2018.

[20] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NeurIPS*, 2015.

[21] Tim Allan Wheeler and Mykel J. Kochenderfer. Factor graph scene distributions for automotive safety analysis. In *ITSC*, 2016.

[22] Tim Allan Wheeler, Mykel J. Kochenderfer, and Philipp Robbel. Initial scene configurations for highway traffic propagation. In *ITSC*, 2015.

[23] Kelvin Wong, Qiang Zhang, Ming Liang, Bin Yang, Renjie Liao, Abbas Sadat, and Raquel Urtasun. Testing the safety of self-driving vehicles by simulating perception and prediction. *ECCV*, 2020.

[24] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

[25] Bin Yang, Ming Liang, and Raquel Urtasun. HDNET: exploiting HD maps for 3d object detection. In *CoRL*, 2018.

[26] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018.