Scenario Dreamer: Vectorized Latent Diffusion for Generating Driving Simulation Environments (Supplementary Document)

Luke Rowe [1,2,6], Roger Girgis[1,3,6], Anthony Gosselin[1,3],
Christopher Pal[1,2,3,5], Liam Paull[1,2,5], Felix Heide[4,6]
[1]Mila, [2]Université de Montréal, [3]Polytechnique Montréal,
[4]Princeton University, [5]CIFAR AI Chair, [6]Torc Robotics
https://princeton-computational-imaging.github.io/scenario-dreamer

This supplementary document provides additional information to support the findings in the main manuscript. Specifically, we include additional quantitative and qualitative results, details of the proposed simulation framework, additional model details, and evaluation details. We encourage readers to visit our webpage for additional qualitative results and videos.

Contents

1. Additional Results

1.1. Behaviour Simulation

Method	Control?	JSD ($\times 10^{-2}$) ↓				Agent Collision (%) ↓	Planner Collision (%) ↓
		Lin. Spd.	Ang. Spd.	Acc.	Near. Dist.		
IDM [18]	✗	9.2	0.4	19.8	1.6	7.2	5.8
Trajeglish† [12]	✗	19.5	0.3	**19.7**	4.0	6.4	7.0
CtRL-Sim (Positive Tilting)	✓	**4.1**	**0.1**	20.1	**1.3**	**6.2**	**4.9**
CtRL-Sim (Negative Tilting)	✓	4.2	0.2	26.1	1.5	10.9	11.9

Table 1. Comparison of different methods for behaviour simulation over 1000 Waymo test scenes. † indicates reimplementation.

Table 1 presents the behaviour simulation results, comparing various methods on a held-out test set of simulation-compatible Waymo scenes. We evaluate against the rule-based baseline IDM [18], which is employed as the behaviour model in SLEDGE [2], as well as the competitive data-driven behaviour model Trajeglish [12]. All methods operate on the Scenario Dreamer lane graph representation, where each Waymo map is processed as a set of lane centerlines resampled to 50 points per lane segment and a corresponding lane graph. The IDM policy assigns a random route for each agent to follow by traversing the lane graph. Simulations are performed within an $80\text{m} \times 80\text{m}$ field of view (FOV) around the ego vehicle at each timestep. The ego vehicle uses an IDM planner that follows the lane centerline route closest to the logged trajectory in the dataset. We report standard JSD distributional realism metrics alongside collision rates: (*Agent Collision*), the rate of collisions among simulated agents, and (*Planner Collision*), the rate of collisions between simulated agents and the IDM planner. To showcase the controllability of CtRL-Sim, we include results for models with exponentially tilted behaviour: $\kappa = +10$ (*CtRL-Sim Positive Tilting*) and $\kappa = -50$ (*CtRL-Sim Negative Tilting*).

We observe that the CtRL-Sim behaviour model performs competitively with both data-driven and rule-based baselines across JSD and collision rate metrics. Notably, the positively-tilted CtRL-Sim model surpasses the baselines in all JSD metrics except acceleration JSD, while achieving the lowest planner collision rate (4.9%) due to the positive tilting. By contrast, the IDM and Trajeglish behaviour models exhibit higher planner collision rates, demonstrating their inability to effectively coordinate with the IDM planner. A key advantage of CtRL-Sim is its ability to intuitively control adversarial behaviours. Even with negative tilting, CtRL-Sim maintains realistic driving behaviours, evidenced by only a modest increase in JSD metrics, while the planner collision rate rises by 7 percentage points. Importantly, this version of CtRL-Sim has not been fine-tuned on adversarial driving data, unlike prior work in [15].

1.2. Qualitative Results

Figure 1 showcases samples generated by the Scenario Dreamer L model in *lane-conditioned object generation* and *initial scene generation* modes, trained on both the nuPlan and Waymo datasets. Figures 3 and 4 compare random initial scene samples from our Scenario Dreamer L model (trained on nuPlan) with those from the retrained SLEDGE DiT-XL model [2]. Despite having 100M fewer parameters, $4\times$ lower inference latency, and $\sim 4\times$ fewer GPU training hours, Scenario Dreamer L produces higher-quality scenes. Additionally, Figures 3 and 4 compares rasterized ground-truth samples processed through the DriveSceneGen vectorization pipeline [17] with random samples from Scenario Dreamer L trained on the Waymo dataset. The vectorization pipeline introduces notable errors in lane graph reconstruction, often resulting in implausible configurations. In contrast, Scenario Dreamer L generates significantly more realistic lane graphs, demonstrating the effectiveness of the proposed vectorized approach.

Figure 2 illustrates the inpainting capabilities of the Scenario Dreamer L model trained on the Waymo dataset. Scenario Dreamer is able to inpaint at the border of complex lane geometries, such as intersections. We refer readers to the Scenario Dreamer webpage, where we visualize the diffusion chain of Scenario Dreamer for the different supported generation modes. Figure 5 showcases the diversity of generated Scenario Dreamer scene generations by visualizing multiple randomly generated scenes under the same initial conditions: (Top) inpainting on the same partial scene and (Bottom) full scene generation conditioned on the same number of lanes (24) and agents (8). In both cases, the generated scenes demonstrate plausible diversity. Figure 6 shows qualitative examples of closed-loop CtRL-Sim behaviour model rollouts simulated from Scenario Dreamer generative environments.

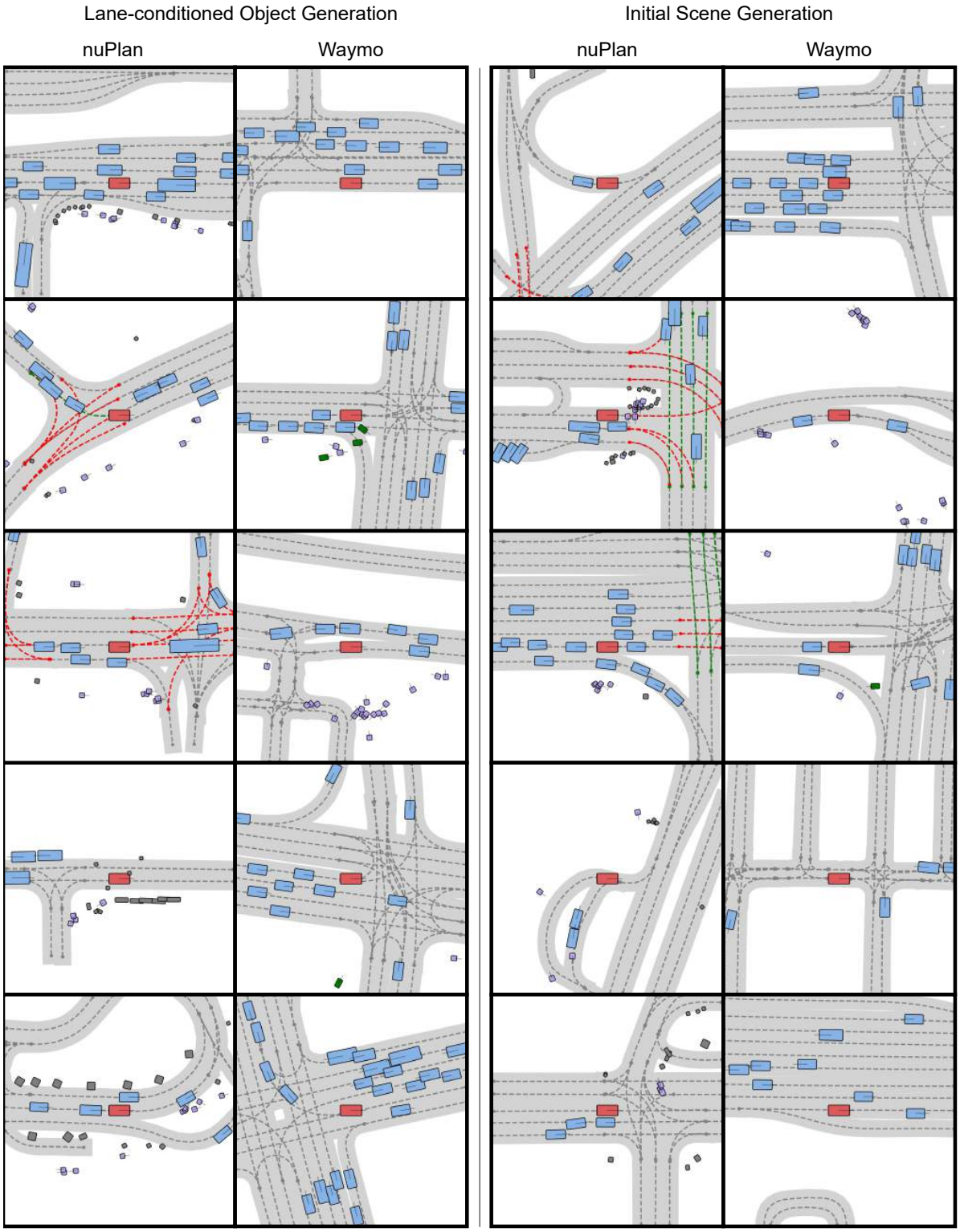


Figure 1. **Lane-conditioned object generation and initial scene generation qualitative results.** We visualize samples from the Scenario Dreamer (L) model in lane-conditioned object generation mode (columns 1 and 2) and full scene generation mode (columns 3 and 4) on both the nuPlan and Waymo datasets.

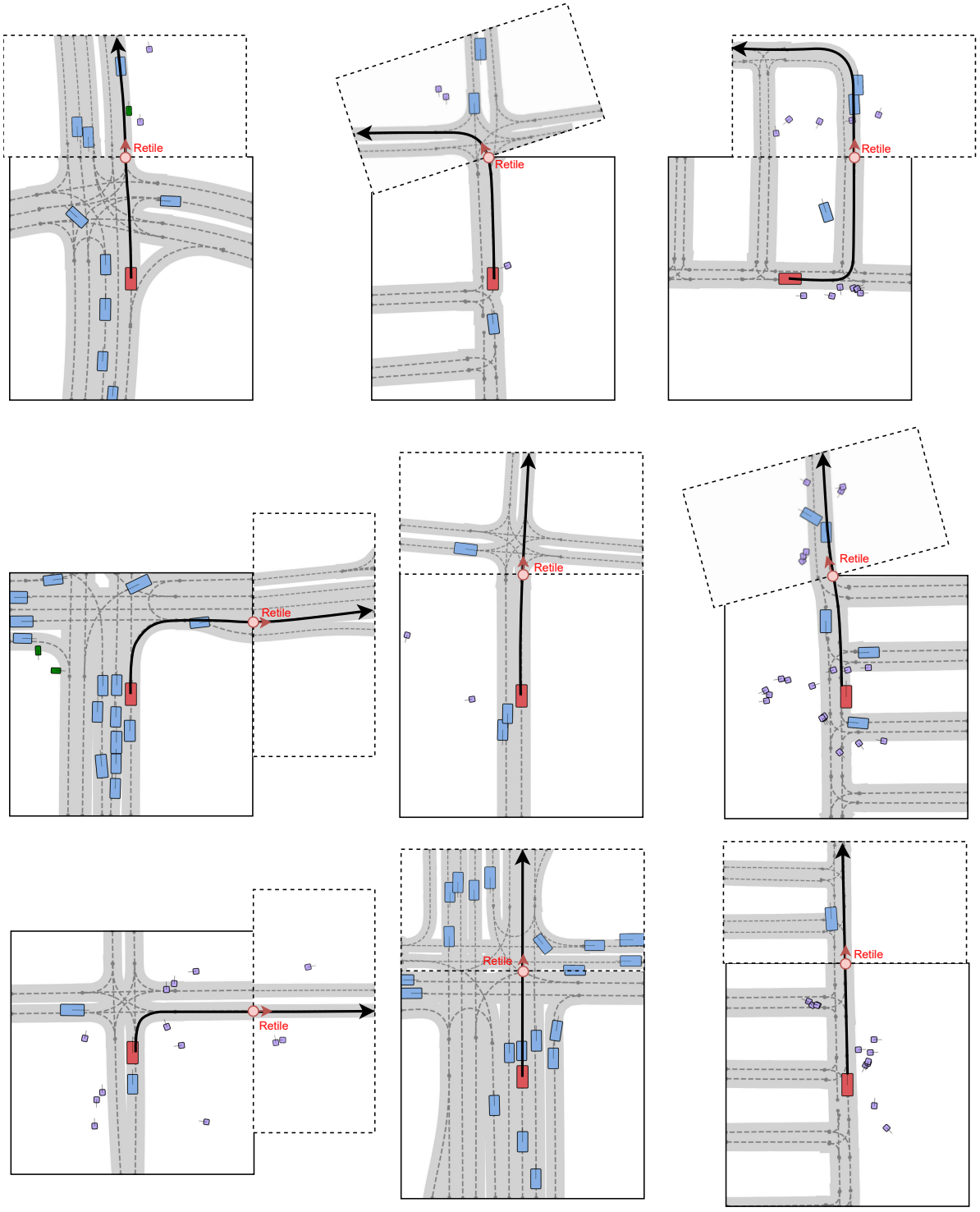


Figure 2. **Scenario Dreamer inpainting.** We visualize the inpainting capabilities of Scenario Dreamer trained on the Waymo dataset. The initial tile is outlined in a solid line, and the new inpainted tile is outlined in a dashed line. The ego route is visualized as a solid black line.

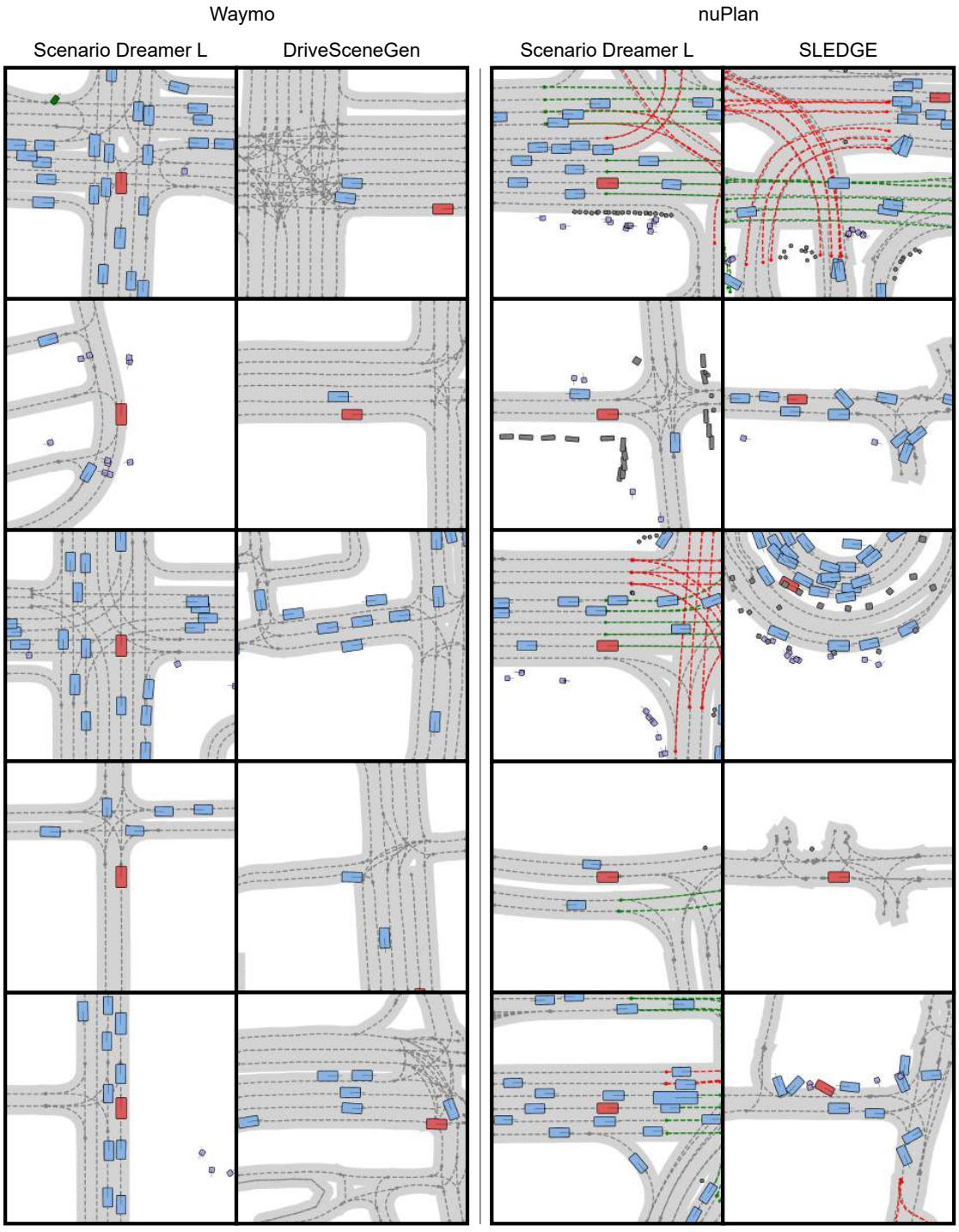


Figure 3. **Qualitative results comparison across methods (set 1).** We visualize samples from the Scenario Dreamer (L) model trained on the Waymo dataset and the privileged DriveSceneGen model (columns 1 and 2) along with samples from Scenario Dreamer (L) and SLEDGE DiT-XL both trained on nuPlan (columns 3 and 4).

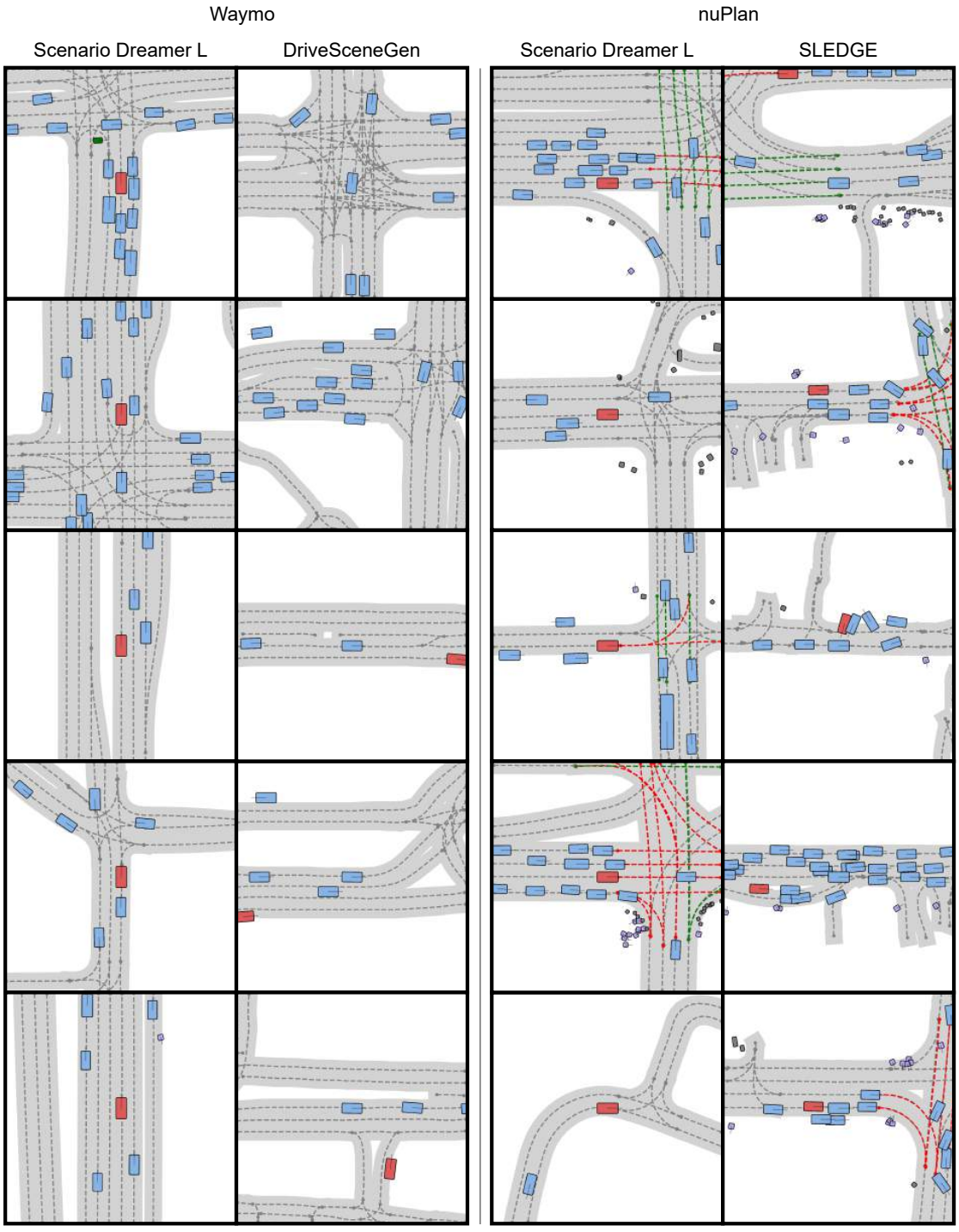


Figure 4. **Qualitative results comparison across methods (set 2).** We visualize samples from the Scenario Dreamer (L) model trained on the Waymo dataset and the privileged DriveSceneGen model (columns 1 and 2) along with samples from Scenario Dreamer (L) and SLEDGE DiT-XL both trained on nuPlan (columns 3 and 4).

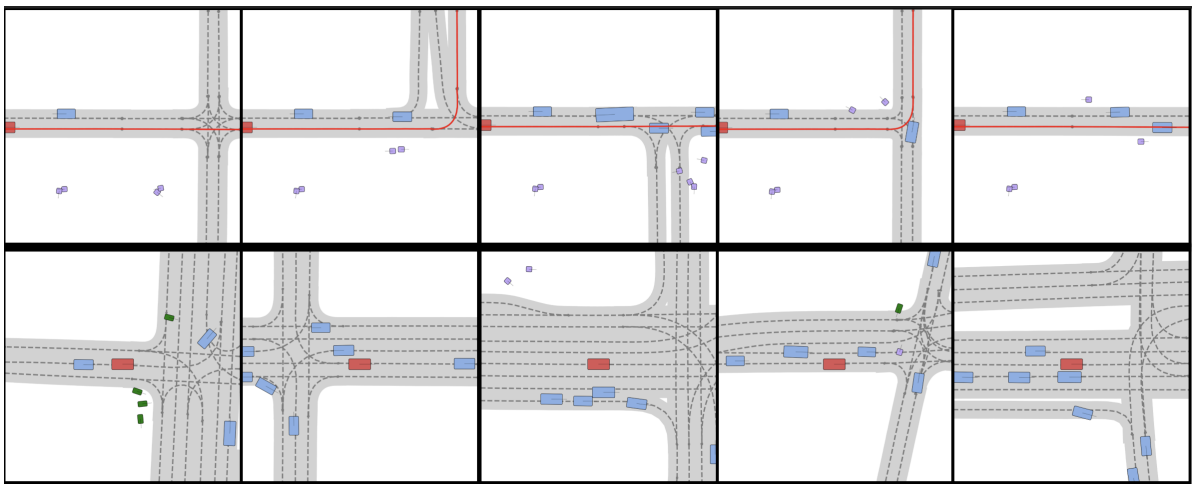


Figure 5. **Scene Diversity**: (Top) Right-half of the scene inpainted from the same left-half of the scene. (Bottom) Random samples with the same initial conditions: 8 agents and 24 lanes.

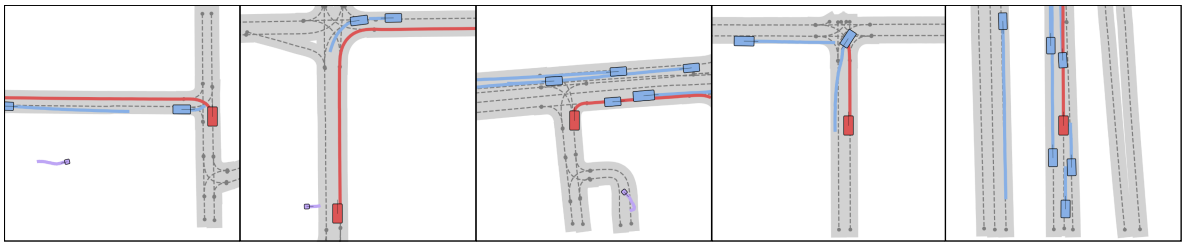


Figure 6. **Behaviour Simulation Examples**: CtRL-Sim rollouts with IDM ego planner simulated from SD initial scenes.

2. Simulation Framework Details

2.1. Simulator Properties

Scenario Dreamer supports the evaluation of AV planners within its generative simulation environments over arbitrarily long simulation lengths. Specifically, it makes it possible to define a 500-meter route for the AV planner to follow. To construct such a simulation environment, an initial scene $\mathcal{I}_F$ is generated from the initial scene generator, and a route is selected. Then, an SE(2) transformation is applied to the generated scene $\mathcal{I}_F$ such that $\mathcal{I}_F$ is renormalized to the end of the driving route in $\mathcal{I}_F$. The Scenario Dreamer diffusion model subsequently inpaints $\mathcal{I}_{F_P}$ based on the existing map $\mathcal{I}_{F_N}$, iterating this process of route selection and inpainting until the desired route length is achieved. Routes are represented by lane centerlines, resampled at 1-meter intervals, and determined by a depth-first traversal of the lane graph starting from the origin.

The Scenario Dreamer model trained on the Waymo dataset serves as the foundation for deriving simulation environments used for AV planner evaluation. Maps are represented using the compact centerline-based format proposed in SLEDGE [2]. Each lane segment consists of 50 points, along with lane connectivity information. The current version of Scenario Dreamer does not include additional map elements such as crosswalks, stop signs, and road edges; however, we plan to incorporate these elements in future work. The simulator supports vehicle, pedestrian, and bicyclist agent types. Agents are controlled using the trained CtRL-Sim model, which operates in the $\Delta x, \Delta y, \Delta\theta$ action space with actions selected from the k-disks vocabulary, visualized in Figure 9. The AV planner outputs steering and acceleration commands, which are passed through a forward bicycle model [6] to update the AV's state, ensuring physical realism in the trajectory. The AV planner's context is defined by a 64m radius around the planner's position at each timestep. The AV planner's objective is to complete the route within a reasonable time limit without colliding or deviating from the path by more than 2.5 meters. The simulation is deemed a failure if the AV collides with an agent, exceeds the deviation threshold, or surpasses the time limit.

2.2. Fully Data-Driven Simulation

Unlike Chitta et al. [2], which employ a rule-based IDM model for behaviour simulation, Scenario Dreamer leverages a data-driven and controllable CtRL-Sim behaviour model, offering enhanced scenario diversity and realism. The AV planner operates within a simulation radius of 64m centered on the AV's current position. To accommodate this, CtRL-Sim simulates agents within a slightly larger $80\text{m}\times80\text{m}$ field of view (FOV) around the AV's position. This restricted FOV minimizes simulation latency, while agents outside the radius remain in a constant state until they enter the FOV. To ensure smooth transitions, agents that exit the FOV are not allowed to reenter. By limiting behaviour modeling to this restricted FOV, we achieve efficiency using a lightweight 8M parameter behaviour model—at least $5\times$ smaller than several state-of-the-art behaviour models trained on the Waymo dataset [5, 12, 21].

2.3. Adversarial Simulation Environments

Scenario Dreamer supports the generation of adversarial simulation environments by leveraging the controllability of CtRL-Sim through exponential tilting. In an adversarial simulation environment, we ensure at least one agent within the simulation radius is sampling from a negatively tilted return distribution, which encourages adversarial collision behaviours with the ego vehicle. We show systematically that these adversarial tilting produces more collisions with an IDM planner in Table 1.

2.4. Training Scenario Dreamer-compatible RL Agents in GPUDrive

We train PPO [16] planning agents on 100 simulation-compatible Waymo scenarios for 100 million steps using the GPU-accelerated GPUDrive simulator [6]. Training requires approximately 24 hours on a single A100-L GPU. To ensure compatibility with Scenario Dreamer environments, we make four key modifications to the GPUDrive simulator:

1. **Route-conditioning**: We replace goal-conditioning with route-conditioning, as Scenario Dreamer provides a route for the planner to follow. The route is defined by the sequence of lane centerlines nearest to the logged ground-truth trajectory of the ego.
2. **Map simplification**: We remove road edges and crosswalks from the map, retaining only lane centerlines that are preprocessed in the same way as in Scenario Dreamer.
3. **Route deviation penalty**: To encourage on-road driving in the absence of road edges, we penalize the planner for lateral deviations exceeding $2.5 + \text{width} = 4.83$m from the lane centerline.
4. **Episode termination**: Episodes end only when the ego either reaches the end of the prescribed route (i.e., the goal) or when the episode times out after 9 seconds.

The reward function consists of three components: a collision penalty with weight 0.1, a route-deviation penalty with weight 0.05, and a goal-achievement reward with weight 1.

3. Model Details

3.1. Vectorized Latent Diffusion Model

3.1.1 Autoencoder

Encoder The autoencoder encoder $\mathcal{E}\phi$ takes as input a set of N_l lanes $\{\mathbf{l}_i\}_{i=1}^{N_l}$, N_o objects $\{\mathbf{o}_i\}_{i=1}^{N_l}$, and the lane connectivity $\mathbf{A} \in \{0,1\}^{N_l \times N_l \times 4}$. Separate 2-layer MLPs f_l and f_o process the lane and object vectors, respectively, where the lane type and object type are concatenated prior to the MLP

$$\begin{aligned}\mathbf{l}_i^0 &= f_l([\mathbf{l}_i, c_i^l]),\\ \mathbf{o}_i^0 &= f_o([\mathbf{o}_i, c_i^o]),\end{aligned}$$

where c_i^l is the lane type of lane i and c_i^o is the object type of object i. A 2-layer MLP f_a additionally processes the lane connectivity type $a_{ij} := \mathbf{A}[i,j]$ between all (directed) edges (i,j) connecting lane i to lane j

$$\mathbf{a}_{ij}^0 = f_a(a_{ij}).$$

The lane, object, and lane connectivity embeddings are then processed by a sequence of N_E factorized attention blocks (FABs), where each FAB consists of sequential lane-to-lane (L2L), lane-to-object (L2O), and object-to-object (O2O) attention layers. First, we define $\mathbb{L}^k := \{\mathbf{l}_i^k\}_{i=1}^{N_l}$, $\mathbb{O}^k := \{\mathbf{o}_i^k\}_{i=1}^{N_o}$, and $\mathbb{A}^k := \{\mathbf{a}_{ij}^k\}_{(i,j)\in[N_l]\times[N_l]}$ as the set of embeddings for the lanes, objects, and lane connectivities following the output of the k'th FAB. The k'th FAB updates the k'th layer lane, object, and lane connectivity embeddings as

$$(\mathbb{L}^{k+1}, \mathbb{O}^{k+1}, \mathbb{A}^{k+1}) = \text{FAB}_k(\mathbb{L}^k, \mathbb{O}^k, \mathbb{A}^k),$$

where FAB_k is decomposed as follows

$$\begin{aligned}\mathbb{L}^{k+1} &= \text{L2L}_k(\mathbb{L}^k, \mathbb{A}^k),\\ \mathbb{O}^{k+1} &= \text{L2O}_k(\text{proj}^k(\mathbb{L}^{k+1}), \mathbb{O}^k),\\ \mathbb{O}^{k+1} &= \text{O2O}_k(\mathbb{O}^{k+1}),\\ \mathbb{A}^{k+1} &= \text{EdgeUpdate}_k(\mathbb{L}^{k+1}, \mathbb{A}^k)\end{aligned}$$

where L2L is a Transformer encoder block [19] where the multi-head self-attention operation fuses the edge features $\mathbb{A}^k$ into the keys and values, as in [22]. L2O is a Transformer decoder block where the lane embeddings are projected to the object hidden dimension with a linear layer proj_k, and O2O is a Transformer encoder block. As we are processing sets of elements and wish to preserve permutation equivariance, each of the L2L, L2O, and O2O blocks do not have positional encodings. EdgeUpdate_k updates each lane connectivity embedding $\mathbf{a}_{ij}^k$ as

$$\mathbf{a}_{ij}^{k+1} = f_k^2([\mathbf{a}_{ij}^k, f_k^1([\mathbf{l}_i^{k+1}, \mathbf{l}_j^{k+1}])]),$$

where $[\cdot,\cdot]$ denotes concatenation along the feature dimension, and f_k^1, f_k^2 are each 2-layer MLPs. For the partitioned scenes, to predict the number of lanes in $\mathcal{I}_{F_P}$, we additionally define a learnable query vector $\mathbf{q}$ and each FAB_k in the encoder is augmented with a lane-to-query (L2Q) attention layer following the L2L layer

$$\mathbb{Q}^{k+1} = \text{L2Q}_k(\mathbb{L}_{F_N}^{k+1}, \mathbb{Q}^k),$$

where $\mathbb{Q}^0 = \{\mathbf{q}\}$ and $\mathbb{L}_{F_N}^{k+1}$ denotes the embeddings of the lanes in $\mathcal{I}_{F_N}$ of a partitioned scene.

After N_E FABs, the lane and object embeddings are projected to mean and variance parameters with latent dimension K_l and K_o, respectively, as in the standard VAE, and the resulting transformed query vector $\mathbf{q}^{N_E}$ is passed through a 3-layer MLP f_{num} to predict the number of lanes in $\mathcal{I}_{F_P}$

$$\begin{aligned}\boldsymbol{\mu}_{\mathcal{L}}^i &= f_{\text{mean}}^{\mathcal{L}}(\mathbf{l}_i^{N_E}),\\ \boldsymbol{\sigma}_{\mathcal{L}}^i &= f_{\text{std}}^{\mathcal{L}}(\mathbf{l}_i^{N_E}),\\ \boldsymbol{\mu}_{\mathcal{O}}^i &= f_{\text{mean}}^{\mathcal{O}}(\mathbf{o}_i^{N_E}),\\ \boldsymbol{\sigma}_{\mathcal{O}}^i &= f_{\text{std}}^{\mathcal{O}}(\mathbf{o}_i^{N_E}),\\ \hat{N}_l^{F_P} &= f_{\text{num}}(\mathbf{q}^{N_E})\end{aligned}$$

where $f^{\mathcal{L}}_{\text{mean}}, f^{\mathcal{L}}_{\text{std}}, f^{\mathcal{O}}_{\text{mean}}, f^{\mathcal{O}}_{\text{std}}$ are linear layers.

Decoder Given sampled latents $\{\{\mathbf{h}^{\mathcal{O}}_i\}_{i=1}^{N_o}, \{\mathbf{h}^{\mathcal{L}}_i\}_{i=1}^{N_l}\} \sim \mathcal{E}_\phi$ from the encoder, separate 2-layer MLPs f_{h_l} and f_{h_o} process the lane and object latents, respectively

$$\begin{aligned}\mathbf{l}^0_i &= f_{h_l}(\mathbf{h}^{\mathcal{L}}_i),\\ \mathbf{o}^0_i &= f_{h_o}(\mathbf{h}^{\mathcal{O}}_i).\end{aligned}$$

We additionally derive lane connectivity features $\mathbf{a}^0_{ij}$ by processing the incident lane embeddings $\mathbf{l}^0_i, \mathbf{l}^0_j$ through a 2-layer MLP f_{h_a} for all i, j

$$\mathbf{a}^0_{ij} = f_{h_a}([\mathbf{l}^0_i, \mathbf{l}^0_j])$$

Then, as in the encoder, we process the lane, object, and lane connectivity embeddings with a stack of N_D FABs. The resulting embeddings are used to predict the lane positions, lane type, object features, object type, and lane graph connectivity, for all i, j

$$\begin{aligned}\hat{\mathbf{l}}_i &= f_{\text{lane}}(\mathbf{l}^{N_D}_i),\\ \hat{c}^l_i &= f_{\text{lane-type}}(\mathbf{l}^{N_D}_i),\\ \hat{\mathbf{o}}_i &= f_{\text{object}}(\mathbf{o}^{N_D}_i),\\ \hat{c}^o_i &= f_{\text{object-type}}(\mathbf{o}^{N_D}_i),\\ \hat{a}_{ij} &= f_{\text{connectivity}}(\mathbf{a}^{N_D}_{ij}),\end{aligned}$$

where $f_{\text{lane}}, f_{\text{object}}$ are 4-layer MLPs and $f_{\text{lane-type}}, f_{\text{object-type}}, f_{\text{connectivity}}$ are 3-layer MLPs.

The autoencoder loss function is defined by

$$\begin{aligned}L_{\text{ae}} = &\underbrace{\lambda_{\text{lane}}\frac{1}{N_l}\sum_{i=1}^{N_l}\Big(\ell_2(\mathbf{l}_i, \hat{\mathbf{l}}_i) + \text{ce}(\hat{c}^l_i, c^l_i)\Big) + \frac{1}{N_o}\sum_{i=1}^{N_o}\Big(\ell_2(\mathbf{o}_i, \hat{\mathbf{o}}_i) + \text{ce}(\hat{c}^o_i, c^o_i)\Big) + \lambda_{\text{conn}}\frac{1}{N_l^2}\sum_{i=1}^{N_l}\sum_{j=1}^{N_l}\text{ce}(\hat{a}_{ij}, a_{ij})}_{\text{reconstruction loss}}\\ &\underbrace{-\frac{\beta}{2}\Big(\frac{1}{N_l}\sum_{i=1}^{N_l}\Big[1 + \log(\boldsymbol{\sigma}^{\mathcal{L}}_i)^2 - (\boldsymbol{\mu}^{\mathcal{L}}_i)^2 - (\boldsymbol{\sigma}^{\mathcal{L}}_i)^2\Big] + \frac{1}{N_o}\sum_{i=1}^{N_o}\Big[1 + \log(\boldsymbol{\sigma}^{\mathcal{O}}_i)^2 - (\boldsymbol{\mu}^{\mathcal{O}}_i)^2 - (\boldsymbol{\sigma}^{\mathcal{O}}_i)^2\Big]\Big)}_{\text{KL loss}} + \lambda_{\text{num}}\text{ce}(\hat{N}_l^{F_P}, N_l^{F_P}),\end{aligned}$$

where $\lambda_{\text{lane}}, \lambda_{\text{conn}}, \lambda_{\text{num}}$ are coefficients to scale the respective lanes, $\text{ce}(\cdot,\cdot)$ denotes the cross entropy loss, and β scales the KL loss. For non-partitioned scenes, $\lambda_{\text{num}} = 0$.

3.1.2 Latent Diffusion Model

The latent diffusion model models the joint distribution over lane and object latents: $p_\theta(\{\mathbf{h}^{\mathcal{L}}_i\}_{i=1}^{N_l}, \{\mathbf{h}^{\mathcal{O}}_i{}_{i=1}^{N_o}\}|N_o, N_l)$. We let $\mathbf{H}_0$ denote the matrix of stacked lane and object latents. Then, the diffusion model models the distribution: $p_\theta(\mathbf{H}_0|N_o, N_l)$.

Forward Diffusion Process We define a forward noising process over T steps for the latents $\mathbf{H}_0$. Following DDPM [4], starting from data $\mathbf{H}_0 \sim q(\mathbf{H}_0)$, we can define a chain of noisy latents with a variance schedule $\beta_1, \ldots, \beta_T$

$$\begin{aligned}q(\mathbf{H}_{1:T}|\mathbf{H}_0) &= \prod_{t=1}^{T} q(\mathbf{H}_t|\mathbf{H}_{t-1}),\\ q(\mathbf{H}_t|\mathbf{H}_{t-1}) &:= \mathcal{N}(\mathbf{H}_t; \sqrt{1-\beta_t}\mathbf{H}_{t-1}, \beta_t\mathbf{I}),\end{aligned}$$

Leveraging the properties of Gaussians, we have that

$$q(\mathbf{H}_t|\mathbf{H}_0) = \mathcal{N}(\mathbf{H}_t; \sqrt{\bar{\alpha}_t}\mathbf{H}_0, (1-\bar{\alpha}_t)\mathbf{I}), \tag{1}$$

where $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t}\alpha_s$.

Reverse Diffusion Process We define a corresponding reverse diffusion process for the latents. Given a sufficiently large T, the distribution of latents is approximately distributed as $\mathbf{H}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. If we can sample from $q(\mathbf{H}_{t-1}|\mathbf{H}_t)$, then we can sample from $q(\mathbf{H}_0)$ by first sampling noise $\mathbf{H}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and iteratively sampling from $\mathbf{H}_{t-1} \sim q(\mathbf{H}_{t-1}|\mathbf{H}_t)$ for T steps. Unfortunately, $q(\mathbf{H}_{t-1}|\mathbf{H}_t)$ is intractable and thus we approximate it with a neural network. Concretely, we define a Markov chain starting from $p(\mathbf{H}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

$$p_\theta(\mathbf{H}_{0:T}) := p(\mathbf{H}_T) \prod_{t=1}^{T} p_\theta(\mathbf{H}_{t-1}|\mathbf{H}_t),$$
$$p_\theta(\mathbf{H}_{t-1}|\mathbf{H}_t) = \mathcal{N}(\mathbf{H}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{H}_t, t), \Sigma_t).$$

Following [4], we let $\Sigma_t := \tilde{\beta}_t \mathbf{I} := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$, and we utilize the ϵ-parameterization of $\boldsymbol{\mu}_\theta(\mathbf{H}_t, t)$:

$$\boldsymbol{\mu}_\theta(\mathbf{H}_t, t) := \frac{1}{\sqrt{\alpha_t}}(\mathbf{H}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{H}_t, t)).$$

Training Objective To train ϵ_θ, we optimize the variational lower bound to the log-likelihood (ELBO)

$$-\mathbb{E}_{q(\mathbf{H}_0)} \log p_\theta(\mathbf{H}_0) \leq \mathbb{E}_{q(\mathbf{H}_{0:T})}[\log \frac{q(\mathbf{H}_{1:T}|\mathbf{H}_0)}{p_\theta(\mathbf{H}_{0:T})}]. \tag{2}$$

Up to reweighting, optimizing Equation 2 is equivalent to optimizing the simple DDPM objective

$$L_{\text{dm}} = \mathbb{E}_{t, \mathbf{H}_t, \boldsymbol{\epsilon}_t}\left[||\boldsymbol{\epsilon}_t - \epsilon_\theta(\mathbf{H}_t, t)||_2^2\right] \tag{3}$$
$$= \mathbb{E}_{t, \mathbf{H}_0, \boldsymbol{\epsilon}_t}\left[||\boldsymbol{\epsilon}_t - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{H}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)||_2^2\right], \tag{4}$$

where (4) is derived from an application of (1).

Architecture We decompose $\mathbf{H}_t$ into lane latents $\mathbf{H}_t^{\mathcal{L}}$ and object latents $\mathbf{H}_t^{\mathcal{O}}$. The latent diffusion model ϵ_θ takes as input a set of N_l noised lane latents and N_o noised object latents $(\mathbf{H}_t^{\mathcal{L}}, \mathbf{H}_t^{\mathcal{O}}) := \{\{\mathbf{h}_{i,t}^{\mathcal{L}}\}_{i=1}^{N_l}, \{\mathbf{h}_{i,t}^{\mathcal{O}}\}_{i=1}^{N_o}\}$ and a timestep t, and predicts the noise $\boldsymbol{\epsilon}_t$ such that $\mathbf{H}_t = \sqrt{\bar{\alpha}_t}\mathbf{H}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t$. We first embed the noisy lane and object latents with separate 2-layer MLPs for all i, where $f_{\text{emb},l}$ embeds the lane latents into hidden dimension d_l and $f_{\text{emb},o}$ embeds the object latents into hidden dimension $d_o < d_l$. We additionally apply an additive sinusoidal positional encoding to each embedded latent vector to mitigate the effects of permutation ambiguity

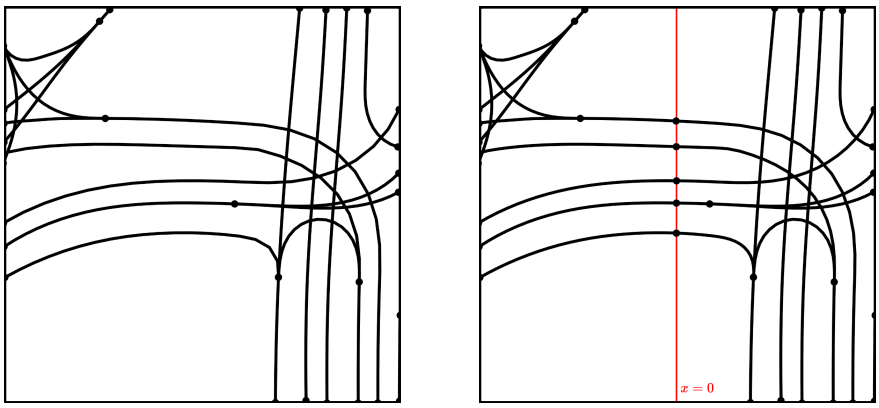
$$\mathbf{l}_i^0 = f_{\text{emb},l}(\mathbf{h}_{i,t}^{\mathcal{L}}) + \mathbf{p}_i^{\mathcal{L}},$$
$$\mathbf{o}_i^0 = f_{\text{emb},o}(\mathbf{h}_{i,t}^{\mathcal{O}}) + \mathbf{p}_i^{\mathcal{O}},$$

where $\mathbf{p}_i^{\mathcal{L}}, \mathbf{p}_i^{\mathcal{O}}$ correspond to the i'th lane and object positional encodings, respectively. We then process the embedded latents through a sequence of N_{DM} FABs with AdaLN-Zero conditioning [11]. Concretely, each FAB in the latent diffusion model consists of the following operations

$$\mathbb{L}^{k+1} = \text{O2L}_k(\text{proj}_1^k(\mathbb{O}^k), \mathbb{L}^k, \mathbf{C}),$$
$$\mathbb{L}^{k+1} = \text{L2L}_k(\mathbb{L}^{k+1}, \mathbf{C}),$$
$$\mathbb{O}^{k+1} = \text{L2O}_k(\text{proj}_2^k(\mathbb{L}^{k+1}), \mathbb{O}^{k+1}, \mathbf{C}),$$
$$\mathbb{O}^{k+1} = \text{O2O}_k(\mathbb{O}^{k+1}, \mathbf{C}),$$

where $\text{O2L}_k, \text{L2L}_k, \text{L2O}_k, \text{O2O}_k$ are DiT blocks [11] with AdaLN-Zero conditioning on vector $\mathbf{C}$, where $\mathbf{C}$ consists of the summation of a sinusoidal positional encoding of the timestep t, a learnable embedding encoding the scene type identity (*i.e.,* partitioned or non-partitioned scene), and a learnable embedding encoding the city (*i.e.,* Singapore, Las Vegas, Boston, or Pittsburgh for the nuPlan dataset, and simulation-compatible or simulation-incompatible for Waymo). proj_1^k projects the embedded object latents into dimension d_l, and proj_2^k projects the embedded lane latents into dimension d_o. Following N_{DM} FABs, the resulting lane and object embeddings are decoded with 4-layer MLPs

$$\hat{\boldsymbol{\epsilon}}_{\mathcal{L},t}^i = f_\epsilon^l(\mathbf{l}_i^{N_{\text{DM}}}),$$
$$\hat{\boldsymbol{\epsilon}}_{\mathcal{O},t}^i = f_\epsilon^o(\mathbf{o}_i^{N_{\text{DM}}}).$$



(a) **Non-partitioned scene** (b) **Partitioned scene**

Figure 7. **Partitioned vs. Non-partitioned Scenes.** In Figure 7a, we show a non-partitioned scene where lanes can arbitrarily cross the $x = 0$ boundary. In Figure 7b, we split the lanes that cross the $x = 0$ boundary (shown in red).

3.1.3 Training and Implementation Details

Dataset The nuPlan dataset includes 1,300 hours of logged data across four cities. Following Chitta et al. [2], we sample 450,000 frames for training and 50,000 for validation, with sampling intervals of 30s, 1s, 2s, and 2s for Las Vegas, Boston, Pittsburgh, and Singapore, respectively. The nuPlan dataset comprises logged scenarios from four cities: Singapore, Las Vegas, Boston, and Pittsburgh. It includes three object types—vehicles, pedestrians, and static objects—and three lane types: centerline, green light, and red light. In the Scenario Dreamer model, traffic light configurations are represented as lane vectors overlapping with the lane centerlines. The Waymo Open Motion Dataset provides $487,002$ training scenarios and $44,097$ validation scenarios, covering 1,750 km of unique roadways. To construct initial scenes for training, we sample each scenario at a random timestep. The Waymo Open Motion dataset contains logged scenarios from six U.S. cities: San Francisco, Phoenix, Mountain View, Los Angeles, Detroit, and Seattle. Each Waymo scene is labeled as either simulation-compatible (*i.e.,* no traffic light signals) or simulation-incompatible following the Nocturne [20] filtering process. This classification allows us to generate simulation-compatible scenes at inference time via classifier-free guidance while leveraging the simulation-incompatible scenes during training. For the Waymo dataset, the Scenario Dreamer model accounts for three object types—vehicles, pedestrians, and bicycles—and a single lane type representing lane centerlines.

We preprocess both the nuPlan and Waymo datasets into partitioned and non-partitioned scenes. Figure 7 illustrates examples of both formats. For partitioned scenes, we add successor/predecessor connections to lanes split by the $x = 0$ border. Additionally, we follow the lane graph preprocessing approach introduced in SLEDGE [2], merging adjacent lanes that permit only a single traversable path. This step effectively removes all key points in the lane graph with degree $= 2$. Each traffic scene is processed within a 64m$\times$64m field of view (FOV) centered on the ego agent. For the Waymo dataset, we further exclude off-road vehicles, defined as those located more than 1.5m from a lane centerline.

Autoencoder The architectural configurations for the autoencoder are consistent across datasets, with some dataset-specific adjustments. For the nuPlan dataset, the autoencoder includes an additional 1M parameters to predict the lane type (centerline red or green traffic light), a feature not present in the Waymo dataset as global traffic light configurations are not modelled in the Waymo dataset. The autoencoder is trained for 50 epochs on a single A100 GPU, which takes approximately 36 hours. Dataset-specific limits are set as follows: $N_o = 30$ and $N_l = 100$ for Waymo, and $N_o = 61$ and $N_l = 100$ for nuPlan. The model uses $N_E = 2$ encoder and $N_D = 2$ decoder factorized Transformer blocks. Hidden dimensions are set to 1024 for lanes, 512 for agents, and 64 for lane connectivities. The autoencoder also defines lane and agent latent dimensions as 24 and 8, respectively. Training is conducted using the AdamW optimizer with a learning rate of $1e^{-4}$, weight decay of $1e^{-4}$, and a total batch size of 128. The learning rate follows a linear decay schedule with 1000 warmup steps. Hyperparameters include $\lambda_{\text{lane}} = \lambda_{\text{conn}} = 10$, $\lambda_{\text{num}} = 0.1$, and $\beta = 0.01$. The autoencoder features are normalized to $[-1, 1]$ by computing the minimum and maximum values of each lane/object attribute in the training dataset. Dropout is not applied during training.

Latent Diffusion Model The latent diffusion model employs $N_{\text{DM}} = 2$ factorized Transformer blocks, with hidden dimensions of 2048 for lanes and 512 for agents. Following by [13], we normalize the latents by calculating the mean and standard deviation from a subset of sampled latents in the training set. During training, we sample these normalized latents to construct each batch, rather than directly regressing on the mean latent as is done in SLEDGE [2], which we found

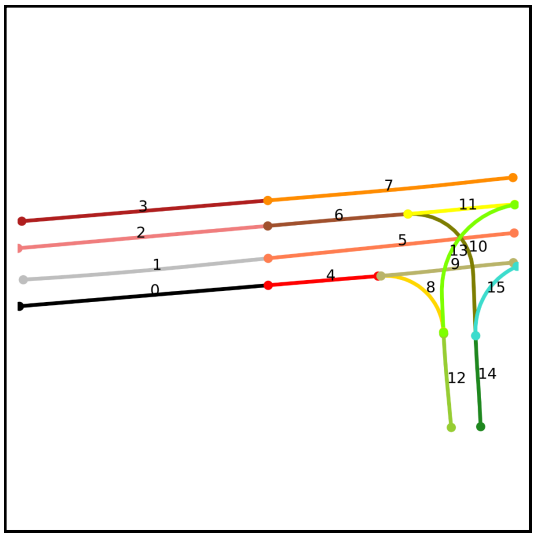


Figure 8. **Proposed ordering scheme.** We illustrate the proposed ordering scheme for the lanes of a partitioned scene, where agent ordering follows a similar approach. We order first by minimum x-value, and if the difference is less than ϵ, we then order by minimum y-value, then maximum x-value, then maximum y-value. Importantly, all lanes before the $x = 0$ boundary (in $\mathcal{I}_{F_N}$) are assigned an ordering strictly lower than the lanes after the $x = 0$ boundary (in $\mathcal{I}_{F_P}$).

introduces a beneficial regularization effect. The base model (B) includes one lane-to-lane (L2L) DiT block within each factorized attention block, while the large model (L) incorporates three L2L blocks, allocating additional capacity to lane generation, which requires greater model capacity than agent generation. Both models use the same autoencoder latents for training. The base and large models are trained with the AdamW optimizer, using a constant learning rate of $1e^{-4}$, weight decay of $1e^{-5}$, and a total batch size of 1024. We apply an exponential moving average (EMA) of 0.9999. Lane noise prediction loss is scaled by a factor of 10. The diffusion model employs 100 diffusion steps with a cosine variance schedule (β_t) [10]. During sampling, low-temperature sampling [1] is applied with $\alpha = 0.75$ to the lane latents in the reverse diffusion chain. Additionally, noisy lane and object latents $\mathbf{H}_t$ are clipped to the range $(-5, 5)$ after each denoising step. We dropout the conditioning information during training with probability 0.1 and apply a classifier guidance scale $s = 4.0$ at inference. No dropout is applied.

To address the challenges associated with permutation ambiguity during training, we order the lane latents and object latents by minimum x-value, and if x-values differ by less than $\epsilon = 0.5$ meters, they are subsequently ordered by minimum y-value, then maximum x-value, and finally maximum y-value. Figure 8 illustrates the proposed ordering for the lanes and agents within a given scene. Ordered as above, a sinusoidal positional encoding is then applied to the embedded lane and object latents to imbue the noisy latents with relative spatial information.

3.2. CtRL-Sim Behaviour Model

3.2.1 Architectural Details

The Scenario Dreamer behaviour model builds upon the architectural details of CtRL-Sim [15], with modifications to support arbitrarily long rollouts without relying on log-replay trajectories. In CtRL-Sim, the goal state for each agent is defined as the final state in the log-replay trajectory and used as conditioning. However, since the Scenario Dreamer initial scene generator provides only the initial agent states, we remove goal conditioning and instead model agent behaviours directly from these initial states. CtRL-Sim employs an encoder-decoder Transformer architecture for multi-agent behaviour simulation. The initial scene generated by Scenario Dreamer is encoded using an encoder with E Transformer encoder blocks. The decoder then processes tokenized trajectory sequences $x = \langle \dots, s_t^1, G_t^1, a_t^1, \dots, s_t^N, G_t^N, a_t^N, \dots \rangle$ with D Transformer decoder blocks, utilizing a temporally causal mask. Here, s_t^i, G_t^i, and a_t^i represent the state, return, and action of agent i at timestep t, with N denoting the number of agents in the scene.

To accommodate multiple agent types (e.g., vehicles, pedestrians, cyclists), we adopt the k-disks tokenization scheme [12], utilizing a discrete vocabulary of size 384, as visualized in Figure 9. To enable extended rollouts, we also remove absolute timestep conditioning, ensuring that the model can generalize over varying time horizons. Since CtRL-Sim simulates agent

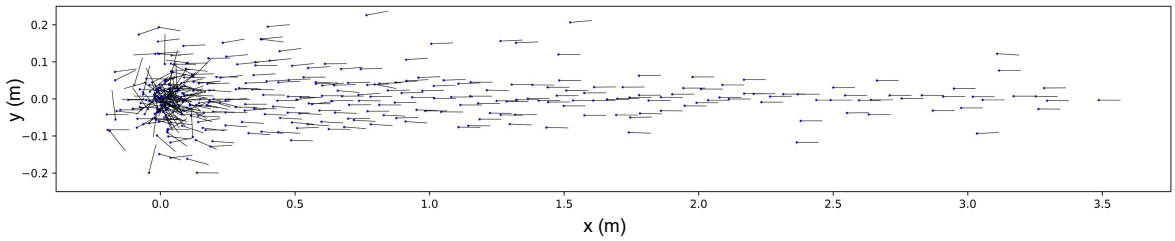


Figure 9. **Visualization of the k-disks tokenization scheme.** Each point denotes the Δx, Δy offset and $\Delta\theta$ offset denoted by the extending line.

behaviours on top of Scenario Dreamer-generated maps, we adapt its map encoder to process Scenario Dreamer maps in a compact representation. In this representation, adjacent lanes are merged when there is only one traversable path, simplifying the lane structure. Specifically, CtRL-Sim operates on a map consisting solely of lane centerlines, with additional map elements such as crosswalks, stop signs, and road edges excluded.

The original CtRL-Sim models the undiscounted return over fixed trajectory lengths T, defined as $G_t := \sum_{i=t}^{T} r_i$. However, this approach does not generalize to trajectories longer than T, limiting its applicability to rollouts of arbitrary length. To enable CtRL-Sim to support rollouts longer than those seen during training, we instead model the *discounted return* over a fixed horizon of $H = 2$ seconds, defined as $G_t := \sum_{i=t}^{t+H} r_i$. This adjustment enhances controllability in practice, as the discounted return focuses on a shorter, more actionable horizon of 2 seconds, and this horizon better reflects the decision-making timescales relevant to driving.

Reward function As we wish to explicitly aim to produce agent behaviours that challenge the planner, we define the reward function as follows

$$r_t(s_i^t, s_{\text{ego}}^t) = -10 \times \mathbb{1}_{\text{veh-ego coll}}(s_i^t) + \frac{||s_i^t - s_{\text{ego}}^t||_2, 10)}{10},$$

where s_{ego}^t is the state of the ego vehicle, or planner, at timestep t, and $\mathbb{1}_{\text{veh-ego coll}}(s_i^t)$ is an indicator function for if vehicle i at timestep t is colliding with the ego.

Controllability The primary advantage of CtRL-Sim over a purely imitation-learning based approach to behaviour simulation is that we can flexibly control agent behaviours at test-time via exponential tilting. Concretely, rather than sampling returns from the learned return distribution $p_\theta(G_t|s_t)$, we instead can sample the returns from the exponentially tilted distribution $G_t' \sim p_\theta(G_t|s_t)\exp(\kappa G_t)$, where G_t' is the tilted return-to-go and where κ represents the inverse temperature. Higher values of κ increase the return density around the best outcomes or higher returns, while negative values of κ increase the return density around less favourable outcomes or lower returns.

3.2.2 Training and Inference Details

Training We train the CtRL-Sim behaviour model on the full Waymo Open Motion dataset, consisting of $487,002$ training scenarios and $44,097$ validation scenarios; however, we omit a subset of simulation-compatible validation scenarios for testing. The CtRL-Sim behaviour simulation model trains on sequences of length $H \times N \times 3$, where $H = 32$ represents the horizon length, and $N = 24$ is the number of agents. Return-to-go components (G_t^i) are discretized into 350 bins. Following prior work [9, 12, 14], the agents and map context are encoded in a global frame by centering and rotating the scene to the ego agent. The agent states are formatted in the same way as object states in Scenario Dreamer; however, we additionally include an binary indicator for the ego vehicle in the scene. The map context is represented by road segments, $\mathcal{M} := \{\mathbf{l}_i\}_{i=1}^{N_l}$, where each segment $\mathbf{l}_i := (p_l^1, \ldots, p_l^P)$ comprises a sequence of 50 points. We note that as Scenario Dreamer outputs sequences of 20 points, we upsample the Scenario dreamer lane segments to 50 points so that it can be processed by CtRL-Sim. Up to 100 lane segments within an $125\text{m}\times125\text{m}$ FOV centered on the ego at a random timestep t in the context are selected as map context, while the social context includes up to 24 agents within a $80\text{m}\times80\text{m}$ FOV centered on the ego at timestep t. The model employs a hidden dimension $d = 256$, $E = 2$ Transformer encoder layers, and $D = 4$ Transformer decoder layers. We process the trajectories of all dynamic agents, including vehicles, pedestrians, and cyclists. Embeddings for missing states, actions, and returns tokens are set to zero. Additionally, return tokens where the computed return is truncated to horizon $H' < H$ are set to zero. The training process uses the AdamW optimizer with a total batch size of 64, starting with a learning

rate of 5×10^{-4}, which linearly decays over 500,000 steps. The CtRL-Sim architecture comprises 7.6M parameters and trains in 48 hours on 4 NVIDIA A100 GPUs.

Inference Although CtRL-Sim is trained with a maximum of $N = 24$ agents, it can handle larger scenes by processing agents in subsets of 24 at each timestep. This process begins by normalizing the scene around the ego agent, and identifying 23 additional agents to form a subset. The first subset includes the 23 agents closest to the ego. The next subset includes the ego and the next 23 agents closest to the ego, and this process repeats iteratively until all agents have been processed. Importantly, we always include the ego vehicle in each 24-agent subset as the return is defined relative to the ego vehicle's position. During inference, the context length is set to $H = 32$ timesteps, consistent with the training configuration. At each timestep, the 32 most recent timesteps are used as context, with the scene normalized to the ego agent's position and orientation at the latest timestep in the context. Unlike the original CtRL-Sim, which employs a physics-enhanced dynamics model for vehicle behaviour, we utilize a simpler delta-based forward model. This approach applies the predicted $\Delta x, \Delta y, \Delta\theta$ offsets directly to the agent's state at each timestep, as derived from the predicted action in the k-disks vocabulary. While this forward model does not guarantee physical realism, unlike the physics model it is applicable to multiple agent types.

4. Evaluation Details

4.1. Metrics

4.1.1 Urban Planning Metrics

The Urban Planning metrics measure the distributional realism of the generated lane graph connectivity by computing 1-dimensional Frechet distances on node features for nodes with degree $\neq 2$, referred to as *key points* [2, 3, 8]. The key points are visualized as black dots in Figure 7a. The following features are computed over $50,000$ ground-truth test lane graphs and generated lane graphs, following the definitions outlined in [2]:

- **Connectivity**: computes the degrees of all key points across the lane graph. The length of the 1-d feature list is length $50,000 \times$ avg-num-keypoints-per-lane-graph.
- **Density**: computes the number of key points in each lane graph for a total feature list length of $50,000$.
- **Reach** computes, for each keypoint, the number of paths to other keypoints, for a total feature list length of $50,000 \times$ avg-num-keypoints-per-lane-graph.
- **Convenience** computes the Dijsktra path lengths for all valid paths between key points for a total feature list length of $50,000 \times$ avg-num-paths-per-lane-graph.

Following SLEDGE [2], we compute the Frechet Distance and *not* the squared Frechet distance. Furthermore, we multiply the Connectivity, Density, Reach, and Convenience Frechet distances by 10, 1, 1, 10, respectively, for readability.

4.1.2 Agent JSD Metrics

Following SceneControl [7], we compute the Jensen-Shannon Divergence (JSD) metrics, which evaluate the distributional realism of the initial vehicle bounding box configurations by comparing $50,000$ real and generated scenes. The Jensen Shannon Divergence between two normalized histograms p and q is computed as

$$\frac{D_{\text{KL}}(p||m) + D_{\text{KL}}(q||m)}{2},$$

where m is the pointwise mean of p and q and D_{KL} is the KL-divergence. We compute the following features to compute JSDs:

- **Nearest Distance** computes the nearest distance between each vehicle and its neighbours. We clip values between $(0, 50)$m with a bin size of 1m. We scale this JSD metric by 10 for readability.
- **Lateral Deviation** computes the lateral deviation to the closest centerline, computed only over vehicles within 1.5m of a lane centerline. We clip values between $(0, 1.5)$m with a bin size of 0.1m. We scale this JSD metric by 10.
- **Angular Deviation** computes the angular deviation from the closest centerline, computed only over vehicles within 1.5m of a lane centerline. We clip values between $(-200, 200)$ degrees with a bin size of 5 degrees. We scale this JSD metric by 100.
- **Length** computes the lengths of all vehicles. We clip values between $(0, 25)$m with a bin size of 0.1m. We scale this JSD metric by 100.
- **Width** computes the widths of all vehicles. We clip values between $(0, 5)$m with a bin size of 0.1m. We scale this JSD metric by 100.

- **Speed** computes the speed of all vehicles. We clip values between $(0, 50)$m with a bin size of 1m. We scale this JSD metric by 100.

The total feature list length for computing each above JSD metric is $50,000 \times$ avg-num-vehicles-per-scene.

4.1.3 Behaviour Simulation JSD Metrics

Following CtRL-Sim [15], we compute JSD metrics for the following features:

- **Linear speed** computes the speed of all agents at each timestep along the trajectory rollout. We use 200 uniformly spaced bins between 0 and 30 m/s.
- **Angular speed** computes the angular speed (change in heading over time) of all agents at each timestep. We use 200 uniformly spaced bins between -50 and 50 degrees.
- **Acceleration** computes the acceleration of all agents at each timestep. We use 200 uniformly spaced bins between -10 and 10.
- **Nearest distance** computes the nearest distance between each vehicle and its neighbours at each timestep. We use 200 uniformly spaced bins between 0 and 40.

References

[1] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *ICLR*, 2023. 13

[2] Kashyap Chitta, Daniel Dauner, and Andreas Geiger. SLEDGE: synthesizing driving environments with generative models and rule-based traffic. In *ECCV*, 2024. 2, 8, 12, 15

[3] Hang Chu, Daiqing Li, David Acuna, Amlan Kar, Maria Shugrina, Xinkai Wei, Ming-Yu Liu, Antonio Torralba, and Sanja Fidler. Neural turtle graphics for modeling city road layouts. In *ICCV*, 2019. 15

[4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 10, 11

[5] Yihan Hu, Siqi Chai, Zhening Yang, Jingyu Qian, Kun Li, Wenxin Shao, Haichao Zhang, Wei Xu, and Qiang Liu. Solving motion planning tasks with a scalable generative model. In *ECCV*, 2024. 8

[6] Saman Kazemkhani, Aarav Pandya, Daphne Cornelisse, Brennan Shacklett, and Eugene Vinitsky. GPUDrive: Data-driven, multi-agent driving simulation at 1 million FPS. In *ICLR*, 2025. 8

[7] Jack Lu, Kelvin Wong, Chris Zhang, Simon Suo, and Raquel Urtasun. Scenecontrol: Diffusion for controllable traffic scene generation. In *ICRA*, 2024. 15

[8] Lu Mi, Hang Zhao, Charlie Nash, Xiaohan Jin, Jiyang Gao, Chen Sun, Cordelia Schmid, Nir Shavit, Yuning Chai, and Dragomir Anguelov. Hdmapgen: A hierarchical graph generative model of high definition maps. In *CVPR*, 2021. 15

[9] Jiquan Ngiam, Vijay Vasudevan, Benjamin Caine, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, David J. Weiss, Ben Sapp, Zhifeng Chen, and Jonathon Shlens. Scene transformer: A unified architecture for predicting future trajectories of multiple agents. In *ICLR*, 2022. 14

[10] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *ICML*, 2021. 13

[11] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*. IEEE, 2023. 11

[12] Jonah Philion, Xue Bin Peng, and Sanja Fidler. Trajeglish: Traffic modeling as next-token prediction. In *ICLR*, 2024. 2, 8, 13, 14

[13] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 12

[14] Luke Rowe, Martin Ethier, Eli-Henry Dykhne, and Krzysztof Czarnecki. FJMP: factorized joint multi-agent motion prediction over learned directed acyclic interaction graphs. In *CVPR*, 2023. 14

[15] Luke Rowe, Roger Girgis, Anthony Gosselin, Bruno Carrez, Florian Golemo, Felix Heide, Liam Paull, and Christopher Pal. CtRL-sim: Reactive and controllable driving agents with offline reinforcement learning. In *CoRL*, 2024. 2, 13, 16

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017. 8

[17] Shuo Sun, Zekai Gu, Tianchen Sun, Jiawei Sun, Chengran Yuan, Yuhang Han, Dongen Li, and Marcelo H. Ang. Drivescenegen: Generating diverse and realistic driving scenarios from scratch. *IEEE RA-L*, 2024. 2

[18] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000. 2

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 9

[20] Eugene Vinitsky, Nathan Lichtlé, Xiaomeng Yang, Brandon Amos, and Jakob Foerster. Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world. In *NeurIPS*, 2022. 12

[21] Yu Wang, Tiebiao Zhao, and Fan Yi. Multiverse transformer: 1st place solution for waymo open sim agents challenge 2023. *arXiv*, 2023. 8

[22] Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-centric trajectory prediction. In *CVPR*, 2023. 9