

# Generating Useful Accident-Prone Driving Scenarios via a Learned Traffic Prior

## Supplementary Material

Davis Rempe<sup>1,2</sup>

Jonah Philion<sup>2,3,4</sup>

Leonidas J. Guibas<sup>1</sup>

Sanja Fidler<sup>2,3,4</sup>

Or Litany<sup>2</sup>

<sup>1</sup>Stanford University

<sup>2</sup>NVIDIA

<sup>3</sup>University of Toronto

<sup>4</sup>Vector Institute

[nv-tlabs.github.io/STRIVE](https://nv-tlabs.github.io/STRIVE)

## 1. Introduction

In this document, we include additional technical details and results omitted from the main paper due to space constraints. Sec. 2 gives details on the main technical methods, Sec. 3 gives details for experiments, and Sec. 4 provides additional results to supplement those in the main paper.

**Video Results.** We encourage readers to view the video results in the [supplementary webpage](#) to get a better sense of the scenarios produced by STRIVE.

## 2. Method Details

In this section, we provide additional technical and implementation details about the methods introduced in Sec 3, 4, and 5 of the main paper. Note that due to many formulated optimization problems being very similar, mathematical notation is often overloaded and may mean slightly different things depending on the section/optimization problem. This is always noted in the text.

### 2.1. Learned Traffic Model

The learned traffic model is introduced in Sec 3.1 of the main paper. Here we step through the main components of the architecture in more detail. All neural network components use ReLU activations and layer normalization [1] unless noted otherwise.

**State Representation.** In practice, trajectories used as input and supervision for the traffic model are represented as  $\mathbf{Y}^i = [\mathbf{y}_1^i, \mathbf{y}_2^i, \dots, \mathbf{y}_T^i]$  for agent  $i$  with  $T$  timesteps where the state at time  $t$  is  $\mathbf{y}_t^i = [x_t, y_t, \theta_t^x, \theta_t^y, v_t, \dot{\theta}_t] \in \mathbb{R}^6$  containing the 2D position  $(x_t, y_t)$ , heading unit vector  $(\theta_t^x, \theta_t^y)$ , speed  $v_t$ , and yaw rate  $\dot{\theta}_t$ .

**Feature Extraction.** Context features for each agent in the scene are first extracted based on: the past trajectory  $\mathbf{X}^i$ , the map  $\mathcal{M}$ , a one-hot encoding of the semantic class  $\mathbf{s}^i$  (either *car* or *truck* for the experiments in the main paper), and the agent’s bounding box length/width  $\mathbf{b}^i = (l, w)$ . The past trajectory feature for each agent  $\mathbf{p}^i \in \mathbb{R}^{64}$  is encoded from  $\mathbf{X}^i$ ,  $\mathbf{s}^i$ , and  $\mathbf{b}^i$  using a 4-layer MLP with hid-

den size 128. To handle missing frames where an agent has not been annotated in nuScenes [2], the past encoding MLP is also given a flag for each input timestep indicating whether the agent state at that step is valid, *i.e.* whether it is true nuScenes data or has been filled with dummy zeros. The map feature  $\mathbf{m}^i \in \mathbb{R}^{64}$  is extracted from a local map crop of size  $256 \times 256$  around the agent (17m behind, 60m in front, 38.5m to each side) at the last step of  $\mathbf{X}^i$ . The map input contains one channel for each layer in the input (*drivable area*, *carpark area*, *road divider*, and *lane divider* in nuScenes); each layer is a binary image rasterized at 4 pixels/m. The map extraction network is a CNN with 6 convolutional layers (stride 2, kernel sizes [7, 5, 5, 3, 3, 3], and number of filters [16, 32, 64, 64, 128, 128]) followed by a single fully-connected layer. Map extraction uses group normalization between layers [12].

**Prior Network.** The input to the prior is a fully-connected scene graph with a context feature placed at each node that is the concatenation of the previously detailed features  $\mathbf{h}^i = [\mathbf{p}^i, \mathbf{m}^i, \mathbf{s}^i]$ . Before message passing, each  $\mathbf{h}^i$  is further processed with a small 3-layer input MLP to be size 128.

The prior, posterior, and decoder are all graph neural networks (GNN) similar to a scene interaction module [3, 10]. They perform one round of message passing, which involves an *edge network*, *aggregation function*, and *update network*. Consider a single node  $i$  in the scene graph. First, interaction features are computed for every incoming edge. For an edge from node  $j \rightarrow i$ , the edge feature is computed using the *edge network*  $\mathcal{E}$  as  $\mathbf{e}^{ij} = \mathcal{E}(\mathbf{h}^i, \mathbf{h}^j, \mathcal{T}^{ij})$  where  $\mathcal{T}^{ij}$  is the relative position and heading of agent  $j$  in the local frame of agent  $i$ .  $\mathcal{E}$  is a 3-layer MLP with hidden and output size of 128. After computing all edge features, they are aggregated into a single interaction feature  $\mathbf{e}^i \in \mathbb{R}^{128}$  using maxpooling  $\mathbf{e}^i = \max(\mathbf{e}^{i1}, \mathbf{e}^{i2}, \dots)$ . The update network then gives the output at each node  $\mathbf{o}^i = \mathcal{U}(\mathbf{h}^i, \mathbf{e}^i)$ ; it is a 4-layer MLP with hidden size 128.

In the case of the prior network, the outputs at each node are the parameters of a Gaussian distribution. In particular,  $\mathbf{o}^i = [\mu^i, \sigma^i]$  where  $\mu^i \in \mathbb{R}^{32}$  gives the mean and  $\sigma^i \in$

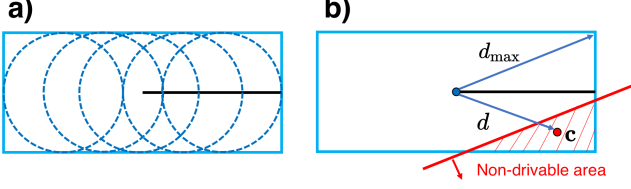


Figure 1. Collision penalties. (a) Computing  $\mathcal{L}_{\text{agent}}$  estimates each vehicle by a set of 5 discs. (b)  $\mathcal{L}_{\text{env}}$  is computed based on the distance between a collision point  $\mathbf{c}$  and the vehicle position.

$\mathbb{R}^{32}$  parameterizes the diagonal covariance matrix so that  $p_{\theta}(\mathbf{z}^i | X, \mathcal{M}) = \mathcal{N}(\mu_{\theta}^i(X, \mathcal{M}), \sigma_{\theta}^i(X, \mathcal{M}))$  where  $\theta$  are the learned parameters of the prior network.

**Posterior (Encoder) Network.** The approximate posterior network (commonly referred to as the *encoder* in CVAEs) is used (i) during training and (ii) to initialize adversarial optimization. It is nearly the same as the prior, but takes in an additional future feature extracted from the future trajectory for each agent. The future trajectory feature for each agent  $\mathbf{f}^i \in \mathbb{R}^{64}$  is encoded from  $\mathbf{Y}_{\text{gt}}^i$ ,  $\mathbf{s}^i$ , and  $\mathbf{b}^i$  using a 4-layer MLP with hidden size 128. Node  $i$  of the scene graph input to the posterior contains a concatenation of  $[\mathbf{f}^i, \mathbf{p}^i, \mathbf{m}^i, \mathbf{s}^i]$  where  $\mathbf{p}^i$ ,  $\mathbf{m}^i$  are the same past and map features given to the prior network. Input processing and message passing is performed in the same way as the prior, and the final output is also a distribution over latents  $q_{\phi}(\mathbf{z}^i | Y_{\text{gt}}, X, \mathcal{M}) = \mathcal{N}(\mu_{\phi}^i(Y_{\text{gt}}, X, \mathcal{M}), \sigma_{\phi}^i(Y_{\text{gt}}, X, \mathcal{M}))$  with  $\phi$  the learned parameters of the network.

**Decoder Network.** As described in the main paper, the decoder progresses autoregressively, predicting one future step at a time. When predicting step  $t$ , each node of the input scene graph contains a concatenation of  $[\mathbf{z}^i, \mathbf{p}_{t-1}^i, \mathbf{m}_{t-1}^i, \mathbf{s}^i, \mathbf{b}^i]$  where  $\mathbf{z}^i$  is sampled from either the prior or posterior output. The past and map features are updated throughout autoregressive rollout, and therefore notated by a time subscript; initially these are the exact same as given to the prior, *i.e.*  $\mathbf{p}_0^i = \mathbf{p}^i$  and  $\mathbf{m}_0^i = \mathbf{m}^i$ .

At step  $t$  of rollout, the concatenated input features are first processed by a 3-layer input MLP (hidden size 128) to get a single 64-dimensional feature at each node. A single round of message passing proceeds in the same way as for the prior to get the output  $\mathbf{o}_t^i = [\hat{v}_t^i, \hat{\theta}_t^i] \in \mathbb{R}^2$  at each node, which contains current linear and angular acceleration. The kinematic bicycle model [5, 8] is then used to get the actual agent state  $\mathbf{y}_t^i = \mathcal{K}(\mathbf{y}_{t-1}^i, \mathbf{o}_t^i, \mathbf{b}^i)$ . Before proceeding to the next step of rollout, the past and map context features must be updated according to the new state. In particular, the past feature is updated using a gated-recurrent unit RNN  $\mathbf{p}_t^i = \text{GRU}(\mathbf{p}_{t-1}^i, \mathbf{y}_t^i)$  with 3 layers (the GRU uses a hidden state of size 64 that is omitted here for brevity). The new map feature  $\mathbf{m}_t^i$  is extracted using the same CNN as before, but with an updated map crop in the local frame of  $\mathbf{y}_t^i$ . The feature at each node can then be updated to  $[\mathbf{z}^i, \mathbf{p}_t^i,$

$\mathbf{m}_t^i, \mathbf{s}^i, \mathbf{b}^i]$  before moving on to predict step  $t + 1$  in the exact same fashion. Note the autoregressive nature of the decoder allows us to roll out further into the future than just the 6s training horizon if desired.

**Training.** Training uses a modified CVAE objective:

$$\mathcal{L}_{\text{cvae}} = \mathcal{L}_{\text{recon}} + w_{\text{KL}} \mathcal{L}_{\text{KL}} + w_{\text{coll}} \mathcal{L}_{\text{coll}} \quad (1)$$

$$\mathcal{L}_{\text{recon}} = \sum_{i=1}^N \|\mathbf{Y}_{\text{post}}^i - \mathbf{Y}_{\text{gt}}^i\|^2 \quad (2)$$

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}(q_{\phi}(Z | Y_{\text{gt}}, X, \mathcal{M}) || p_{\theta}(Z | X, \mathcal{M})) \quad (3)$$

$$\mathcal{L}_{\text{coll}} = \mathcal{L}_{\text{agent}} + \mathcal{L}_{\text{env}}. \quad (4)$$

In practice,  $\mathcal{L}_{\text{recon}}$  only supervises the position and heading, *i.e.*  $\mathbf{Y}_{\text{post}}^i$  and  $\mathbf{Y}_{\text{gt}}^i$  in Eq. (2) contain only  $[x_t, y_t, \theta_t^x, \theta_t^y]$ . The reconstruction loss is applied on one sample from the posterior distribution, while collision losses use a sample from the prior.

$\mathcal{L}_{\text{agent}}$  is introduced in TrafficSim [10]. It uses a differentiable approximation of vehicle-vehicle collision detection, which represents all  $N$  vehicles by discs. We estimate each agent vehicle  $i$  by 5 discs with radius  $r_i$ , as shown in Fig. 1(a), and compute the loss by summing over all pairs of agents  $(i, j)$  over time as:

$$\mathcal{L}_{\text{agent}} = \frac{1}{N^2} \sum_{(i,j), i \neq j} \sum_{t=1}^T \mathcal{L}_{\text{pair}}(\mathbf{y}_t^i, \mathbf{y}_t^j) \quad (5)$$

$$\mathcal{L}_{\text{pair}}(\mathbf{y}_t^i, \mathbf{y}_t^j) = \begin{cases} 1 - \frac{d}{r_i + r_j}, & d \leq r_i + r_j \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

where  $d$  is the minimum distance over all pairs of discs representing agents  $i$  and  $j$ .

$\mathcal{L}_{\text{env}}$  uses a similar idea to penalize collisions with the non-drivable area. This penalty is only applied to the annotated *ego* vehicle in the nuScenes [2] sequences during training, since many non-ego vehicles appear off the annotated map. At each step of rollout, collisions are detected between the ego vehicle and the non-drivable area (by checking for overlap between the rasterized non-drivable layer in  $\mathcal{M}$  and the rasterized vehicle bounding box), and a collision point  $\mathbf{c}$  is determined as the mean of all vehicle pixels that overlap with non-drivable area. The loss is then calculated as:

$$\mathcal{L}_{\text{env}} = \frac{1}{T} \sum_{t=1}^T \mathcal{L}_{\text{drivable}}(\mathbf{y}_t, \mathcal{M}) \quad (7)$$

$$\mathcal{L}_{\text{drivable}}(\mathbf{y}_t, \mathcal{M}) = \begin{cases} 1 - \frac{d}{d_{\text{max}}}, & \text{if partial collision} \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

where  $d$  is the distance between the vehicle position (center of bounding box) and collision point  $\mathbf{c}$ , and  $d_{\text{max}}$  is half the ego bounding box diagonal as shown in Fig. 1(b). Note the loss is only applied if there is a partial collision, *i.e.* only

part of the bounding box overlaps with the non-drivable area – this is because if the vehicle is completely embedded in the non-drivable area, the loss will not give a useful gradient.

The traffic model is implemented in PyTorch [7] and trained using the ADAM optimizer [4] with learning rate  $1e^{-5}$  for 110 epochs. Losses are weighted with  $w_{\text{KL}} = 4e^{-3}$  and  $w_{\text{coll}}$  is split into  $w_{\text{agent}} = 0.05$  for  $\mathcal{L}_{\text{agent}}$  and  $w_{\text{env}} = 0.1$  for  $\mathcal{L}_{\text{env}}$ . The KL loss weight  $w_{\text{KL}}$  is linearly annealed over time, starting from 0.0 at the start of training up to the full value at epoch 20.

## 2.2. Initialization Optimization

As discussed in Sec 3.2 of the main paper, initialization of adversarial optimization is done by first performing inference with the learned approximate posterior  $q_{\phi}(Z_{\text{init}}|Y_{\text{init}}, X, \mathcal{M})$  and then running an *initialization optimization* to get the final  $Z_{\text{init}}$  encapsulating both non-ego agents  $Z = \{\mathbf{z}^i\}_{i=1}^N$  along with the latent planner  $\mathbf{z}^{\text{plan}}$ .

The initialization optimization objective is nearly the same as Eqn 6 in the main paper. It tries to match the initial future trajectories of non-ego agents (from the input nuScenes scenario) and the ego vehicle (from planner rollout within the initial scenario) that are contained in  $Y_{\text{init}}$ :

$$\min_{Z_{\text{init}}} w_{\text{match}} \|Y - Y_{\text{init}}\|^2 - w_{\text{prior}} \log p_{\theta}(Z_{\text{init}}|X, \mathcal{M}) \quad (9)$$

where  $Y = d_{\theta}(Z_{\text{init}}, X, \mathcal{M})$  is the decoded initial scenario but uses *only* position and heading information (*i.e.* no velocities). For initialization optimization, we use  $w_{\text{match}} = 10.0$ ,  $w_{\text{prior}} = 0.01$ , and run for 175 iterations.

## 2.3. Adversarial Optimization

The adversarial optimization is introduced in Sec 3.2 of the main paper. Here we provide details about each objective.

**Match Planner.** In practice, the objective in Eqn 6 of the main paper only uses the position and heading information contained in  $\mathbf{Y}^{\text{plan}}$ ,  $\hat{\mathbf{Y}}^{\text{plan}}$  (*i.e.* no velocities). For all experiments,  $\alpha = 1e^{-5}$ .

**Adversarial Loss.** The  $\delta$  coefficients in the adversarial objective (Eqn 8 of the main paper) can be explicitly manipulated to discourage certain types of scenarios. For all experiments, we dynamically set  $\delta_t^i = 0$  if agent  $i$  is “behind” the planner at time  $t$  (*i.e.* we “mask out” these agents). “Behind” is determined based on the current heading of the planner: if an agent is outside of the planner’s  $180^\circ$  field of view, it is considered behind. This discourages degenerate scenarios with malicious collisions from behind.

We weight the adversarial loss in Eqn 8 of the main paper by  $w_{\text{adv}} = 2.0$ .

**Prior Loss.** In practice, we do not use the  $\gamma^i$  coefficients directly to weight the prior loss for each agent. Instead,

we use them to compute a dynamic weight for each agent by interpolating between minimum and maximum hyperparameters  $w_{\text{prior}}^{\text{min}}$ ,  $w_{\text{prior}}^{\text{max}}$ . Eqn 10 from the main paper becomes

$$\mathcal{L}_{\text{prior}} = -\frac{1}{N} \sum_{i=1}^N \log p_{\theta}(\mathbf{z}^i|X, \mathcal{M}) \cdot w_{\text{prior}}(\gamma^i) \quad (10)$$

$$w_{\text{prior}}(\gamma^i) = \gamma^i w_{\text{prior}}^{\text{max}} + (1 - \gamma^i) w_{\text{prior}}^{\text{min}} \quad (11)$$

where we use  $w_{\text{prior}}^{\text{min}} = 5e^{-3}$  and  $w_{\text{prior}}^{\text{max}} = 1$ . This gives more fine-grained control over the prior loss weight rather than leaving it to  $\gamma_i \in [0, 1]$ . In particular, if an agent is a likely adversary (*i.e.* close to colliding with the planner), its weight will be near  $w_{\text{prior}}^{\text{min}}$ , whereas agents far away will be close to  $w_{\text{prior}}^{\text{max}}$ .

**Initialization Loss.** Similar to the prior loss, the initialization loss (Eqn 12 in the main paper) actually uses  $\gamma^i$  to interpolate between max and min hyperparameters and is written:

$$\mathcal{L}_{\text{init}} = \frac{1}{N} \sum_{i=1}^N \|\mathbf{z}^i - \mathbf{z}_{\text{init}}^i\|^2 \cdot w_{\text{init}}(\gamma^i) \quad (12)$$

$$w_{\text{init}}(\gamma^i) = \gamma^i w_{\text{init}}^{\text{max}} + (1 - \gamma^i) w_{\text{init}}^{\text{min}} \quad (13)$$

where we use  $w_{\text{prior}}^{\text{min}} = 0.05$  and  $w_{\text{prior}}^{\text{max}} = 0.5$ .

**Collision Losses.** The collision term for adversarial optimization is similar to that used to train the traffic model:

$$\mathcal{L}_{\text{coll}} = w_{\text{agent}} \mathcal{L}_{\text{agent}} + w_{\text{env}} \mathcal{L}_{\text{env}} + w_{\text{plan}} \mathcal{L}_{\text{plan}}.$$

$\mathcal{L}_{\text{agent}}$  is the same as defined in Eq. (5) and is applied to all non-ego vehicles to avoid colliding with each other.  $\mathcal{L}_{\text{env}}$  is the same as defined in Eq. (7) and is applied to all non-ego vehicles (instead of the ego vehicle as in CVAE training) to encourage staying on the drivable area.  $\mathcal{L}_{\text{plan}}$  is similar to  $\mathcal{L}_{\text{agent}}$ , but instead of discouraging collisions between non-ego agents, it discourages collisions between the planner and non-ego agents with a large  $\gamma^i$  (*i.e.* unlikely adversaries). This term only affects agents that are close to the planner but have large  $\gamma^i$  because they have been “masked out” as described previously. Intuitively, we don’t want these agents to “accidentally” collide with the planner from behind.

All collision losses are computed on trajectories that are upsampled by  $\times 3$  to avoid missing collisions at the low nuScenes rate of 2 Hz. We use  $w_{\text{agent}} = w_{\text{env}} = w_{\text{plan}} = 20$ .

**Optimization.** The adversarial optimization runtime reported in the main paper uses a machine with an NVIDIA Titan RTX GPU and 12x Intel i7-7800X@3.50GHz CPUs. We optimize for 200 iterations using the ADAM [4] optimizer (we found L-BFGS is overly prone to local minima for this problem) with learning rate 0.05.

## 2.4. Solution Optimization

The solution optimization is introduced in Sec 4.1 of the main paper. It attempts to find a trajectory for the planner that avoids the collision in the adversarial scenario. Like the adversarial optimization, it optimizes  $Z$  and  $\mathbf{z}^{\text{plan}}$  in the latent space of the traffic model and uses very similar objectives:

**1. Match Adversarial Scenario.** All non-ego agents should maintain the same trajectories outputted from adversarial optimization. Let  $Y_{\text{adv}}$  be the set of non-ego trajectories from the collision scenario and  $Y$  be the current scenario during solution optimization, then the objective is:

$$\min_Z \|Y_{\text{adv}} - Y\|^2 - \alpha \log p_\theta(Z|X, \mathcal{M}) \quad (14)$$

with  $\alpha = 1e^{-4}$ . Note the reason we need to actually optimize  $Z$ , instead of simply fixing it, is because the non-ego agent trajectories may change as  $\mathbf{z}^{\text{plan}}$  is optimized due to message passing in the traffic model decoder.

**2. Avoid Collisions.** The goal for the planner is to avoid collisions with other agents and the environment while driving plausibly:

$$\min_{\mathbf{z}^{\text{plan}}} \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{coll}}. \quad (15)$$

The prior loss is

$$\mathcal{L}_{\text{prior}} = -w_{\text{prior}} \log p_\theta(\mathbf{z}^{\text{plan}}|X, \mathcal{M}) \quad (16)$$

with  $w_{\text{prior}} = 5e^{-3}$ . The collision loss is

$$\mathcal{L}_{\text{coll}} = w_{\text{agent}} \mathcal{L}_{\text{agent}} + w_{\text{env}} \mathcal{L}_{\text{env}}$$

where  $\mathcal{L}_{\text{agent}}$  is the same as defined in Eq. (5) but only discourages collisions between the planner and all non-ego agents.  $\mathcal{L}_{\text{env}}$  is the same as defined in Eq. (7) and is applied to the planner only. We use  $w_{\text{agent}} = w_{\text{env}} = 10$ . When computing collision losses, the planner is rolled out 8s into the future (instead of the 6s length of the scenario future) to ensure that it does not end up in an irrecoverable state at the end of the scenario. The planner trajectory is upsampled  $\times 3$  before collision checking.

**Optimization.** Solution optimization uses the ADAM [4] optimizer for 200 iterations with a learning rate of 0.05.

## 2.5. Pre-Filtering Potential Scenarios

As discussed in Sec 5.1 of the main paper, before performing adversarial optimization, initial 8s scenarios from nuScenes are filtered to remove those that will be difficult or impossible to cause a collision. For each potential initialization, 20 futures are sampled from the traffic model conditioned on the past trajectories. A scenario is considered *feasible* if any of the sampled futures meets the following heuristic conditions, which are designed to find an agent that could be made to collide with the planner:

- There is a non-ego agent that passes within 10m of the planner at some timestep  $t$ .
- That agent is not behind the planner (where “behind” is determined in the same way as in Sec. 2.3) at  $t$ .
- That agent is not separated from the planner by non-drivable map area at  $t$ .
- The ego vehicle must move  $> 1$  m/s at some point, avoiding situations where the planner is a “sitting duck” with no hope of avoiding a collision.

If no samples meet these conditions, the initial scenario is not used for scenario generation. After doing this feasibility check, we end up with a candidate non-ego agent that could reasonably collide with planner at time  $t$ . Note, STRIVE does not use this information in any way – the adversarial optimization can and does cause collisions with a different agent than the initial candidate. However, this information is useful for the *Bicycle* baseline which requires the adversary to be chosen before optimization.

## 2.6. Bicycle Baseline Scenario Generation

The *Bicycle* baseline is introduced in Sec 5.1 of the main paper. This approach does not use the learned traffic model to parameterize scenarios, instead explicitly optimizing the acceleration profile of an adversary. As it uses no strong priors on holistic traffic motion, only a single pre-chosen “attacker” is optimized (similar to prior work [11]). For this, we use the candidate adversary determined by the feasibility check in Sec. 2.5 (in practice, using this feasibility check for *Bicycle* would not be possible since it leverages samples from the traffic model, however we use it here to ensure a fair comparison to STRIVE).

Optimization is performed over the future acceleration profile of the adversary  $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_T]$  with  $\mathbf{a}_t = [\dot{v}_t, \ddot{\theta}_t]$ . Let  $\mathbf{b}$  be the length and width of the adversary vehicle, then to get the adversary trajectory  $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T]$  when needed, the kinematic bicycle model is recursively applied  $\mathbf{y}_t = \mathcal{K}(\mathbf{y}_{t-1}, \mathbf{a}_t, \mathbf{b})$  where  $\mathbf{y}_0$  is the state at the last timestep of the fixed *past*.

**Initialization Optimization.** Before optimizing to cause a collision, the acceleration profile of the adversary is fit to its corresponding trajectory in the initial nuScenes scenario  $\mathbf{Y}_{\text{init}}$  with

$$\min_{\mathbf{A}} \|\mathbf{Y} - \mathbf{Y}_{\text{init}}\|^2. \quad (17)$$

This loss is applied only to the position and heading part of the trajectories (*i.e.* not velocities or accelerations).

**Adversarial Optimization.** Next, the main adversarial optimization is performed, which encourages the adversary to collide with the planner using

$$\min_{\mathbf{A}} w_{\text{adv}} \mathcal{L}_{\text{adv}} + w_{\text{accel}} \mathcal{L}_{\text{accel}} + \mathcal{L}_{\text{coll}}. \quad (18)$$



The adversarial term encourages a collision similar to STRIVE by minimizing positional distance:

$$\mathcal{L}_{\text{adv}} = \sum_{t=1}^T \|\mathbf{y}_t - \mathbf{y}_t^{\text{plan}}\| \cdot \delta_t \quad (19)$$

$$\delta_t = \frac{\exp(-\|\mathbf{y}_t - \mathbf{y}_t^{\text{plan}}\|)}{\sum_t \exp(-\|\mathbf{y}_t - \mathbf{y}_t^{\text{plan}}\|)}. \quad (20)$$

The *softmin* in equation Eq. (20) is only choosing a candidate timestep to cause a collision (as opposed to choosing both a candidate agent *and* timestep as in STRIVE) since the colliding agent is fixed ahead of time. Also note that  $\mathbf{y}_t^{\text{plan}}$  is the actual planner trajectory, not a differentiable approximation from the traffic model as used in STRIVE. This means that during optimization, it is necessary to compute gradients through the true planner if the setting is closed-loop (e.g. the *Rule-based* planner). As discussed in Sec 5.1 of the main paper, we implemented an explicit gradient estimation using finite differences to be able to backpropagate through the planner. However, this requires substantially more queries to the planner and is too slow to practically generate many scenarios. Finite differences, while easy to implement, is inefficient compared to black-box approaches explored in prior work [11] – however, it was out of our current scope to explore these options as well since STRIVE does not require them.

The acceleration term regularizes the optimized profile to prefer small accelerations:

$$\mathcal{L}_{\text{accel}} = \frac{1}{T} \sum_{t=1}^T \|\mathbf{a}_t\|^2. \quad (21)$$

The collision loss is

$$\mathcal{L}_{\text{coll}} = w_{\text{agent}} \mathcal{L}_{\text{agent}} + w_{\text{env}} \mathcal{L}_{\text{env}}$$

where  $\mathcal{L}_{\text{agent}}$  is the same as defined in Eq. (5) but only discourages collisions between the adversary and other (fixed) non-ego agents.  $\mathcal{L}_{\text{env}}$  is the same as defined in Eq. (7) and is applied to the adversary only.

**Optimization Details.** We use  $w_{\text{adv}} = 1$ ,  $w_{\text{accel}} = 1$ , and  $w_{\text{agent}} = w_{\text{env}} = 20$ . Initialization optimization uses L-BFGS (which was superior to ADAM for the simple objective) for 50 iterations. Adversarial optimization uses ADAM for 300 iterations.

To fairly compare to STRIVE and evaluate certain metrics, after *Bicycle* adversarial optimization we fit the output scenario within our learned traffic model using the procedure described in Sec. 2.2. We can then compute the likelihood of the adversary’s  $\mathbf{z}$  under the learned prior, and use the exact same solution optimization described in Sec. 2.4.

## 2.7. Rule-based Planner

Our rule-based planner is introduced in Sec 4.2 of the main paper and has the following structure:

1. Extract from the lane graph a finite set of splines that each vehicle might follow.
2. Generate predictions for the future motion of non-ego vehicles along each of the splines from (1).
3. Generate candidate trajectories for the ego vehicle and use the predictions from (2) to estimate the “probability of collision”  $p_{\text{col}}(\tau)$  for each candidate  $\tau$ .
4. Among trajectories that are unlikely to collide  $\{\tau \mid p_{\text{col}}(\tau) < p_{\text{max}}\}$ , choose the trajectory that covers the most distance. If no trajectories are unlikely to collide, choose the trajectory that is least likely to collide.
5. Repeat every  $\Delta t$  seconds.

Note the “intent” of the planner is deterministic, *i.e.* it will always follow the same lane graph path (e.g. choosing whether to turn left or right) when rollout starts from the same initialization. As discussed in the main paper, nuScenes lane graphs do not contain information to switch lanes and since the rule-based planner strictly follows the lane graph, it cannot change lanes. Planner behavior is affected by hyperparameters such as how  $p_{\text{col}}(\tau)$  is computed,  $p_{\text{max}}$ , and the maximum speed and forward acceleration.

## 3. Experimental Details

In this section, we provide details for the experiments performed in Sec 5 of the main paper.

### 3.1. Data and Metrics

**Dataset.** The nuScenes [2] dataset is used for all experiments; we use the scene splits and settings from the nuScenes prediction challenge. All vehicle trajectories in the dataset are pre-processed to remove frames where the vehicle bounding box has a  $> 30\%$  overlap with either the non-drivable area or a car park area. This avoids scenarios with many auxiliary agents that are off of the annotated map or not moving. Trajectories are additionally annotated with velocity and yaw rate using finite differences on the provided positions/headings. Finally, we flip the maps and trajectories for “Singapore” data about the  $x$  axis so that vehicles across the whole dataset (both in Boston and Singapore) are consistently driving on the right-hand side of the road.

For training the learned traffic model, we use scenes from the training split of the nuScenes prediction challenge. Note we use all available trajectory data in the training scenes for *cars* and *trucks*, including the ego trajectories and all agents not removed in our own pre-processing.

Scenario generation in all experiments is initialized from a set of 1200 8s nuScenes scenarios (before pre-filtering in Sec. 2.5) extracted from the train and val splits. Since the original nuScenes data sequences are 20s long, some of these extracted 8s scenarios partially overlap.

**Metrics.** Acceleration and collision velocity metrics are computed using finite differences. The accelerations reported in Tab 1 and Tab 3 of the main paper are *forward accelerations* (calculated using the change in speed) while those in Tab 2 are the full acceleration (encompassing both forward and lateral). This is because Tab 1 and 3 focus on trajectories from the *Rule-based* planner, which follows the lane graph, cannot change lanes, and has a deterministic route as discussed in Sec. 2.7. As a result, generated scenarios cannot make the planner swerve suddenly (which would require leaving the lane graph, changing lanes, or changing route), they can only cause a harsher slow down or speed up, which is captured by measuring forward acceleration.

For all experiments, acceleration (*Accel*), environment collision rate (*Env Coll*), and nearest-neighbor distance (*NN Dist*) are only reported up to the time of collision, since any continuing motion is merely the result of not physically simulating the collision. For the environment collision rate reported in Tab 2, a vehicle is considered in collision if there is more than 5% overlap between its bounding box and the non-drivable area.

### 3.2. Planner-Specific Scenario Generation

This experiment is presented in Sec 5.1 of the main paper. During scenario generation, the *Rule-based* planner uses default manually-set hyperparameters (*i.e.* no large-scale tuning was done beforehand, as described in Sec 5.3 of the main paper). These default hyperparameters were set by observing rollouts on a small subset of nuScenes.

In Tab 1, collision rate is reported over all scenarios for which adversarial optimization was performed (*i.e.* the roughly 500 pre-filtered scenarios). Solution rate is with respect to all scenarios where a collision was successfully caused, while planner trajectory and match planner metrics are computed over all useful scenarios (those where both a collision and solution were found).

### 3.3. Baseline Comparison

This experiment is presented in Sec 5.1 of the main paper, and compares STRIVE to the baseline scenario generation detailed in Sec. 2.6. Metrics reported in Tab 2 are for a set of 139 scenarios where both methods were able to cause a collision from the same nuScenes initialization. Please see Sec. 2.6 for details on how solution rate and NLL are measured for *Bicycle*.

### 3.4. Scenario Analysis

This experiment is presented in Sec 5.2 of the main paper. The results shown in Fig 6 are collision labels assigned to the scenarios generated in Sec 5.1. Note that  $k$ -means clustering was not performed directly on the scenarios generated in Sec 5.1, instead clustering was done beforehand

with a large set of over 400 scenarios generated from various subsets of nuScenes and using many versions of our *Rule-based* planner, giving a wide variety of collisions. Clusters were assigned semantic labels by visual inspection of the collisions in each cluster. Then, after generating new scenarios in Sec 5.1, collision types are assigned to each new scenario by simply associating their collision feature with the closest cluster (*i.e.* clustering is *not* done over again, scenarios are assigned to extant clusters). Videos of representative scenarios assigned to each cluster are shown on the **supplementary webpage**.

We use  $k = 10$  for clustering. Finer-grained classification is possible, if necessary, using larger  $k$  or additional features like collision velocity, however we found the described approach sufficient to analyze the *Rule-based* planner’s performance while being easily interpretable.

### 3.5. Improving Rule-based Planner

This experiment is presented in Sec 5.3 of the main paper. Before any tuning is performed, the planner starts with the same set of “default” hyperparameters described in Sec. 3.2. When tuning the planner hyperparameters, the optimal set is chosen by lowest collision rate with ties broken by lowest acceleration. The planner is first tuned on 800 8s nuScenes scenarios (from the train/val splits), which gives initial optimal hyperparameters for “regular” driving scenarios. Adversarial optimization is performed on the planner with both the *default* and *regular-tuned* hyperparameters to create a set of challenging scenarios to guide further improvements. When tuning on challenging scenarios, “Behind” collisions are removed, which we found unrealistic as discussed in Sec 5.2 of the main paper.

The hyperparameters for the *Rule-based* planner are described in Sec. 2.7. Tuning searches over  $p_{\max}$  in the range of  $[0.05, 0.2]$ , max speeds in the range  $[12.5, 20.0]$  m/s, max accelerations in the range  $[3.0, 4.5]$  m/s<sup>2</sup>, and parameters related to computing  $p_{\text{col}}(\tau)$ . In total, tuning sweeps over 432 hyperparameter combinations.

Results in Tab 3 of the main paper are on scenarios from the held-out nuScenes test set. Metrics over collision scenarios (*Coll*) are reported for scenarios where some hyperparameter setting succeeded in avoiding a collision, *i.e.* scenarios where all hyperparameter settings collide are considered impossible and discarded. The best possible collision rate on regular scenarios is only 3.2% (achieved by choosing a *different* set of parameters for every scenario), making the 4.6% of the regular-tuned planner version close to optimal. The reason this collision rate cannot be 0 is the use of log replay (*i.e.* rolling out the planner in pre-recorded scenarios), which results in some unavoidable collisions: (i) pre-recorded traffic is not reactive to the planner and (ii) the ego vehicle is sometimes initialized off the lane graph which it cannot robustly handle.

Model	ADE (m) ↓	FDE (m) ↓
LDS-AF [6]	1.66	3.58
DLow-AF [13]	1.78	3.77
Trajectron++ [9]	1.51	-
AgentFormer [14]	<b>1.45</b>	<b>2.86</b>
Ours, Full	1.75	3.57
Ours, No Bicycle	1.60	3.17

Table 1. Learned traffic model future prediction accuracy on all nuScenes prediction categories compared to current state of art. ADE/FDE is reported using 10 samples.

Model	ADE (m)	FDE (m)	Env Coll (%)	Veh Coll (%)
Full	1.74	3.54	10.6	5.6
No Bicycle	<b>1.72</b>	<b>3.45</b>	<b>7.2</b>	<b>3.7</b>
No $\mathcal{L}_{\text{env}}$	1.91	3.92	13.2	5.0
No Autoregress	3.68	8.00	16.2	5.4

Table 2. Traffic model ablation study on *cars* and *truck* nuScenes categories only (same as used for scenario generation). Though not using the bicycle model gives better performance, it gives less realistic single-agent vehicle dynamics which is very undesirable for scenario generation.

**Learned Mode Classifier.** The multi-mode version of the planner uses a binary classifier that decides whether the ego vehicle is currently in a “regular” or “accident-prone” situation. In the *learned* version, this classifier is a neural network that has a very similar architecture to the learned traffic model described in Sec. 2.1. In particular, it takes in the past  $2s$  trajectories for all agents in a scene, along with local map crops around each, and processes them in the same way as the traffic model. These features are then placed into a scene graph and message passing is performed in the same way as done for the prior. The output feature (size 64) at the ego node is given to 2-layer MLP that makes a binary classification for the scene. This network is trained on regular nuScenes scenarios from the training split in addition to a diverse set of over 1000 collision scenarios generated from train/val scenes using variations of both the *Replay* and *Rule-based* planners. Training uses a typical binary cross entropy loss that is weighted to account for the data imbalance between regular and collision scenarios.

## 4. Supplemental Experiments

In this section, we provide additional results and experiments omitted from the main paper for brevity.

### 4.1. Traffic Motion Model

We first evaluate the learned traffic model’s ability to accurately predict future motion in a scene.

**Data.** We evaluate on the test split of the nuScenes [2] prediction challenge using  $2s$  (4 steps) of past motion to predict  $6s$  (12 steps) of future. This data contains vehicles from the *bus*, *car*, *truck*, *construction*, and *emergency* categories.

**Metrics.** Evaluation is done with standard future prediction metrics including minimum average displacement error (ADE) and minimum final displacement error (FDE), which are measured over  $K$  samples from the traffic model. For a single agent being evaluated, these metrics are

$$\text{ADE} = \min_k \frac{1}{T} \sum_{t=1}^T \|\hat{\mathbf{y}}_t^{(k)} - \mathbf{y}_t\|_2 \quad (22)$$

$$\text{FDE} = \min_k \|\hat{\mathbf{y}}_T^{(k)} - \mathbf{y}_T\|_2 \quad (23)$$

where  $\hat{\mathbf{y}}_t^{(k)}$  is the predicted *position* of the agent in the  $k$ th sample at time  $t$  and  $\mathbf{y}_t$  is the ground truth. In our experiments, we use  $K = 10$  samples.

For the ablation study, we also measure the **environment and vehicle collision rates**. Environment collision rate is the fraction of predicted future trajectories where more than 5% of the vehicle bounding box overlaps with the non-drivable area. This is measured over all  $K$  samples. The vehicle collision rate is measured over all agents in each scene (rather than only the single one specified at each data point in the prediction challenge test split) and all  $K$  samples. It is the same as used in TrafficSim [10], which counts the number of agents in collision (*i.e.* have a bounding box overlap more than IoU 0.02 with another agent).

#### 4.1.1 Baseline Comparison

Prediction performance is compared to reported results for recent state-of-the-art models AgentFormer [14], Trajectron++ [9], DLow-AF [13], and LDS-AF [6]. Results are shown in Tab. 1. Our full model is trained only on *car* and *truck* vehicles to be used for scenario generation, so to evaluate on the prediction challenge test split, we modify the category of input vehicles to our model to be one of these (*e.g.* *bus*  $\rightarrow$  *truck*). Our learned traffic model makes accurate predictions and is competitive with current SOTA methods as shown in Tab. 1. We also train an ablation of our model that does not use the kinematic bicycle model, instead the decoder directly predicts output position and headings. This version is trained on all categories in the challenge dataset, and makes more accurate predictions according to ADE/FDE. Note, however, that using the bicycle model is very important for adversarial and solution optimization to ensure output trajectories have reasonable dynamics even when the optimized  $Z$  is off-manifold.

#### 4.1.2 Ablation Study

To evaluate key design differences from TrafficSim [10] and ILVM [3], which our model is based on, we ablate various

Scenarios	Objective	Plausibility of Adversary Trajectory ↓						Plausibility of Other Trajectories ↓		
		Col (%)	Sol (%)	Accel ( $m/s^2$ )	Env Coll (%)	NN Dist (m)	NLL	Accel ( $m/s^2$ )	NN Dist (m)	NLL
All methods collide	Full	-	<b>87.9</b>	<b>1.11</b>	<b>9.4</b>	0.90	517.0	<b>0.45</b>	0.39	341.4
	No $\mathcal{L}_{prior}$	-	78.8	1.26	<b>9.4</b>	1.05	502.9	0.47	0.39	474.2
	No $\mathcal{L}_{coll}$	-	81.8	1.43	15.6	1.11	487.0	<b>0.45</b>	0.39	351.5
	No $\mathcal{L}_{init}$	-	78.8	1.19	<b>9.4</b>	0.83	579.2	0.51	<b>0.36</b>	21.8
	No $\mathcal{L}_{init}, \gamma$	-	84.8	1.13	<b>9.4</b>	<b>0.76</b>	<b>89.2</b>	0.49	0.40	<b>-2.9</b>
	No $\mathcal{L}_{init}, \gamma, \delta$	-	57.6	1.38	15.6	0.99	154.6	1.04	0.65	104.3
All collision scenarios for each method	Full	27.4	86.8	1.30	13.3	0.95	555.9	0.43	0.35	365.2
	No $\mathcal{L}_{prior}$	27.2	83.9	1.29	19.1	0.95	571.8	0.39	0.31	486.0
	No $\mathcal{L}_{coll}$	38.0	82.1	1.42	19.6	0.91	540.8	0.38	0.27	357.0
	No $\mathcal{L}_{init}$	34.6	80.1	1.40	18.3	1.02	611.6	0.41	0.29	6.8
	No $\mathcal{L}_{init}, \gamma$	30.5	76.1	1.34	14.2	0.97	110.4	0.40	0.30	3.0
	No $\mathcal{L}_{init}, \gamma, \delta$	45.9	58.1	2.01	26.4	1.38	230.8	1.07	0.71	127.9

Table 3. Ablation study on adversarial optimization objective for scenario generation on the *Rule-based* planner. In the top section, metrics are reported only for scenarios where all methods were able to cause a collision. In the bottom section, metrics for each method are computed over all collision scenarios generated by that method only, *i.e.* are not directly comparable. Therefore, collision rate is also reported for reference and numbers are not bolded.

components of our traffic model design. Results are shown in Tab. 2, where all models are trained and evaluated only on the *car* and *truck* categories, since this is what we use in the main paper for scenario generation. Same as Tab. 1, *No Bicycle* directly predicts the position and heading from the decoder rather than acceleration profiles that go through the kinematic bicycle model; again, this gives slightly improved performance but less realistic per-agent dynamics. *No  $\mathcal{L}_{env}$*  only uses vehicle collision penalties while training, similar to prior work [10]. Removing the environment collision penalty results in a higher collision rate and lower predictive accuracy. *No Autoregress* uses a GNN decoder that predicts the entire future trajectory in one shot rather than as an autoregressive rollout. This makes the future prediction task more difficult, substantially reducing accuracy.

## 4.2. Adversarial Optimization Ablation Study

Next, we study how various components of the adversarial optimization objective function (introduced in Sec 3.2 of the main paper and detailed in Sec. 2.3) affect the generated scenarios. We consider the following variations:

- No  $\mathcal{L}_{prior}$  – removes the prior loss which keeps agents likely under the learned prior.
- No  $\mathcal{L}_{coll}$  – removes both environment and vehicle collision penalties.
- No  $\mathcal{L}_{init}$  – removes the initialization loss which keeps agents near the initial (realistic) scenario.
- No  $\mathcal{L}_{init}, \gamma$  – removes the  $\gamma$  weights from  $\mathcal{L}_{prior}$ , meaning likely adversaries will need to stay just as likely as all other agents in the scenario.
- No  $\mathcal{L}_{init}, \gamma, \delta$  – additionally removes the  $\delta$  weighting scheme from  $\mathcal{L}_{adv}$ , meaning all agents will be simultaneously trying to collide with the planner at all timesteps,

and it is left entirely up to  $\mathcal{L}_{prior}$  to avoid unrealistic many-vehicle pileups.

Note that when ablating the  $\delta$  and  $\gamma$  weightings, we also remove  $\mathcal{L}_{init}$  because adversaries should not be expected to stay very close to initialization when needing to collide.

Results are shown in Tab. 3. We use the same metrics as for the baseline comparison in Sec 5.1 of the main paper. In addition to computing metrics for the adversary (colliding agent), results for all other non-planner agents are also reported (except for environment collision since nuScenes contains many agents driving off the annotated drivable area). These metrics are not perfect, and it can be hard to evaluate which optimization objective produces “better” scenarios (or even impossible since desired characteristics are dependent on downstream use), however they do give insight into the kinds of scenarios being produced.

The top section of Tab. 3 computes metrics over scenarios where all methods caused a collision (same protocol as Tab 2 in the main paper). However, since there are many variations, this is only 33 scenarios in total. To give a more complete picture of each variation, the bottom section of the table computes metrics over all generated collision scenarios for each method. It also reports the collision rate to contextualize the solution rate and other metrics. Because metrics are computed over a different set of scenarios for each method in the bottom section, numbers are not directly comparable, however trends often mirror those in the top part.

In the top of Tab. 3 we see the full objective gives the highest solution rate, *i.e.* it generates *useful* scenarios at the highest frequency. *No  $\mathcal{L}_{prior}$*  tends to cause less likely trajectories for other agents in the scene since they are no longer constrained by the learned prior, and make environment collisions more common for the adversary as seen in the bottom section. *No  $\mathcal{L}_{coll}$*  similarly causes far more



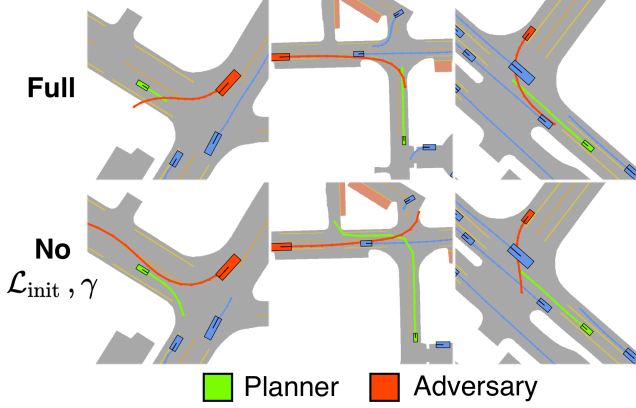


Figure 2. Comparison of generated adversarial scenarios for the *Rule-based* planner using the full objective function and a modified objective with no initialization loss or per-agent weighting in  $\mathcal{L}_{\text{prior}}$ . Removing per-agent weighting requires adversaries to be more likely under the prior, which can cause collisions that are more aligned with “usual” traffic. STRIVE gives the flexibility to modify this objective to generate scenarios best suited for downstream applications.

collisions in addition to adversaries with higher accelerations. Despite this, trajectories maintain reasonable likelihoods since  $\mathcal{L}_{\text{prior}}$  is still used and the traffic model does allow for some collisions as seen in Sec. 4.1.2. *No  $\mathcal{L}_{\text{init}}$*  allows trajectories to stray far from the nuScenes initialization, which produces solvable scenarios at a lower rate and less plausible adversary motion in the bottom part of the table. Note that *NLL* for “others” is trivially very low since the only remaining regularization is  $\mathcal{L}_{\text{prior}}$ . *No  $\mathcal{L}_{\text{init}}, \gamma$*  forces adversaries to stay more likely under the prior resulting in a low *NLL*, but lowered solution rate. The reasonable results produced by this variation indicate the flexibility of our formulation to produce different kinds of scenarios. By removing  $\gamma$ , we encourage scenarios where collisions happen within a more “typical” setting, rather than as a result of out-of-distribution, adversarial behavior. This is shown qualitatively in Fig. 2. *No  $\mathcal{L}_{\text{init}}, \gamma, \delta$*  enables many adversaries to attack simultaneously producing many unsolvable scenarios with unrealistic trajectories.

### 4.3. Performance on Many-agent Scenes

To evaluate the ability of STRIVE to effectively optimize large scenarios, we look specifically at scenes that contain many agents. The mean number of agents in nuScenes is 11.4, so it does not contain many massive scenes: only 2/496 optimized scenarios in Tab 1 of the main paper for the *Rule-based* planner contain  $> 50$  agents. For scenarios with  $\geq 20$  agents, the collision rate is 21.2%, which is consistent with 27.4% in Tab 1, indicating performance on large-scale scenes is similar to smaller ones. This is because even when a scenario has many agents, only a handful near the planner will greatly affect the outcome (due to the op-

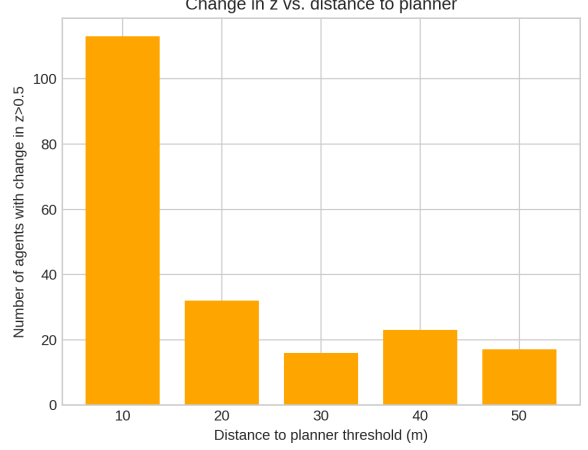


Figure 3. Change in per-agent latent vectors resulting from adversarial optimization as a function of distance to the planner. The bottom axis indicates distance intervals from the planner (e.g. 10 indicates 0-10m, 20 from 10-20m, etc.). For each interval, the number of agents whose latent distance from before to after adversarial optimization is  $> 0.5$  is reported (see text for details). We observe that adversarial optimization tends to greatly affect agents that are close to the planner.

timization objective in Eq 9 of the main paper). As shown in Fig. 3, adversarial optimization makes large changes to the latent vectors of agents within 10m of the planner much more frequently than for distant agents. The change in latent  $z^i$  for each agent is measured as the Euclidean distance between the initialized latent and the final latent (after adversarial optimization), normalized by the maximum difference observed in the scene.

### 4.4. Modeling Second-order Effects

One interesting ability of adversarial optimization is to produce scenarios that cause collisions with the planner using so-called “second-order effects.” In these scenarios, an adversary may not directly collide with the planner in an obviously malicious way – instead, it performs some action that causes the planner or other vehicles to react, thereby causing a collision between the planner and some other agent. For example, we have observed several instances of an adversary in front of the planner suddenly braking, causing the planner to brake and get hit by another agent from behind, rather than the initial adversary in front. This prompted the regularizer  $\mathcal{L}_{\text{plan}}$  introduced in Sec. 2.3. The second-order behavior is possible since STRIVE optimizes *all* agents in a scene rather than choosing specific adversaries ahead of time. Example videos of these second-order scenarios are shown in the supplementary webpage.

### 4.5. Solution Optimization Evaluation

To confirm the solution optimization is filtering out overly-difficult scenarios, we evaluate using the hyperparameter tuning procedure and generated challenging sce-

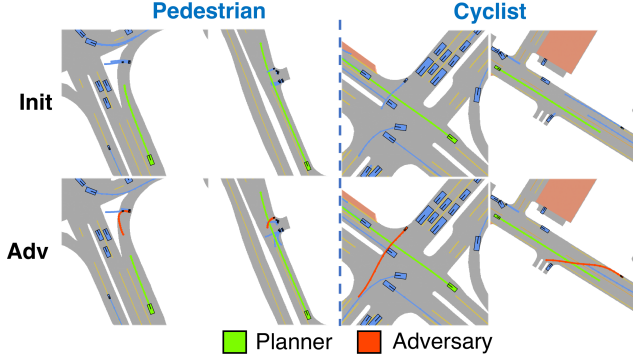


Figure 4. Generated scenarios for the *Replay* planner using a traffic model trained on *all* categories. Top row shows the initial scene and bottom is the output of adversarial optimization. When choosing the adversary ahead of time, STRIVE can cause collisions with both pedestrians (left) and cyclists (right).

narios introduced in Sec 5.3 of the main paper. In particular, we take the vanilla *Rule-based* planner and perform two hyperparameter tuning sweeps: one on generated collision scenarios where the solution optimization found a solution, and one on collision scenarios where no solution could be found. For each sweep, we measure the mean fraction of hyperparameter settings that succeed (*i.e.* don't collide) per scenario in the tuning set. When tuning on *solution-failed* scenarios, **only 11.8% of hyperparameter combinations succeed** in avoiding collisions for each scenario on average. However for tuning on *solution-found* scenarios, 29.8% of hyperparameters succeed for each. This indicates that finding a sufficient hyperparameter setting for scenarios where our solution optimization failed is more difficult than for those where a solution could be found.

#### 4.6. Additional Qualitative Results

Additional qualitative results of STRIVE on the *Rule-based* planner are shown in Fig. 5. Video examples are also included in the supplementary webpage.

#### 4.7. Pedestrian and Cyclist Adversaries

Though our main focus in this work is generating scenarios involving vehicles, as a proof-of-concept we train the learned traffic model on all categories in the nuScenes dataset and generate scenarios for the *Replay* planner where the adversary is a pedestrian or cyclist. For this experiment, a pedestrian/cyclist adversary is specifically chosen before optimization using the procedure in Sec. 2.5. Results are shown in Fig. 4 and on the webpage.

#### 4.8. Failure Cases and Limitations

In addition to the limitations discussed in Sec 6 of the main paper, Fig. 6 shows examples of other STRIVE limitations. First, our proposed solution optimization is iterative

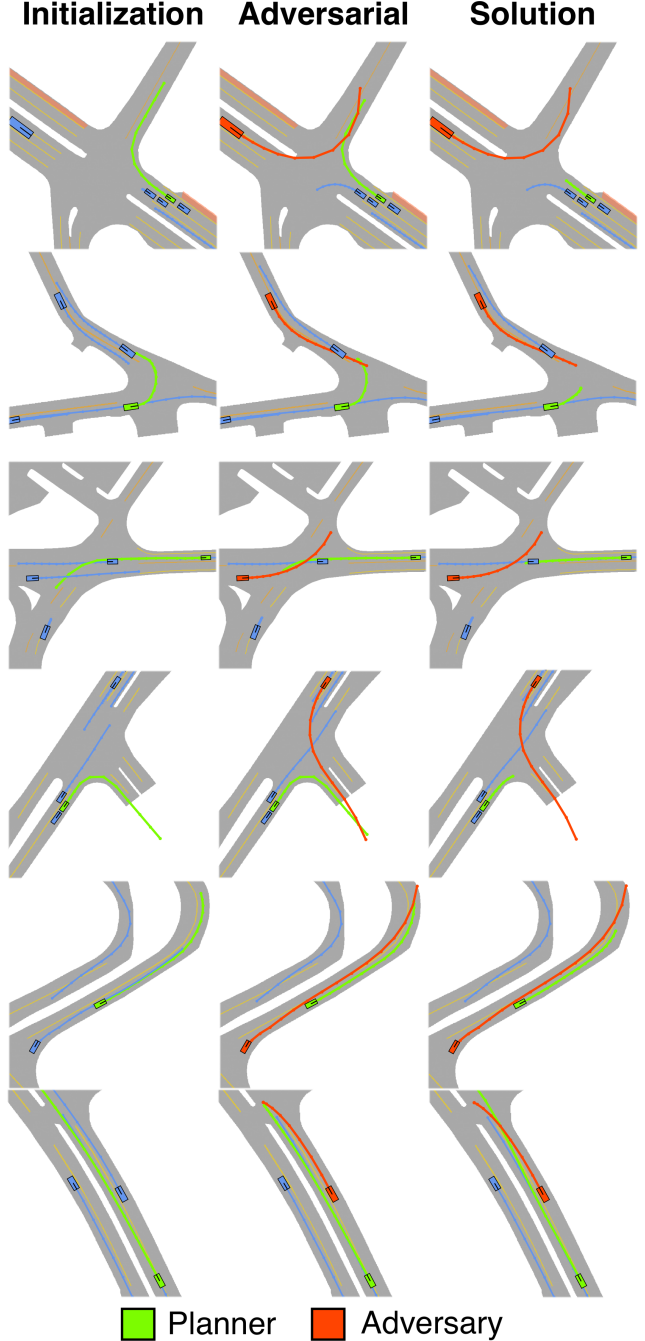


Figure 5. Additional qualitative results of STRIVE for the closed-loop *Rule-based* planner. Adversarial optimization makes large changes to the initial scenario from nuScenes, *e.g.* changing the intent of the adversary or moving stationary vehicles, to cause useful collision scenarios.

and operates on the full temporal planner trajectory, therefore it has access to future information that sometimes allows performing evasive maneuvers even before an “attack” is apparent. An example is in Fig. 6(a) where the optimized solution simply does not pull into the roundabout where

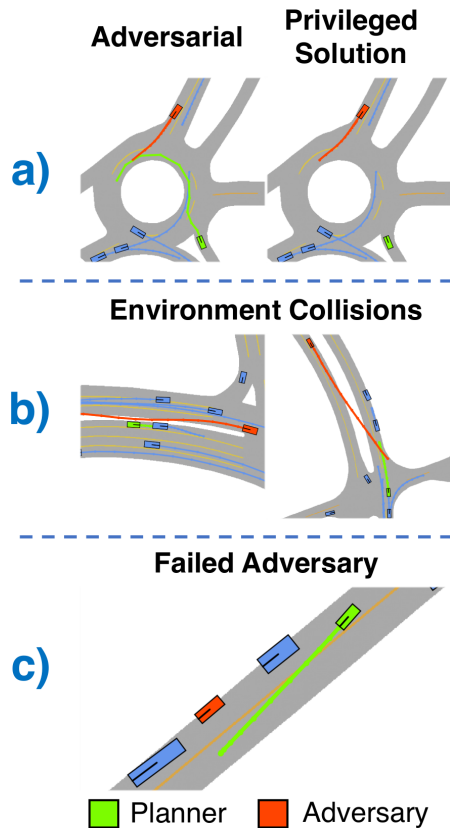


Figure 6. Example failure cases of STRIVE. (a) The solution optimization has access to privileged information, sometimes resulting in unrealistic “solutions”. (b) Adversaries sometimes drive on non-drivable area to cause a collisions. (c) Attacks that require unlikely motion under the learned prior (e.g. a parked car pulling out) can be difficult to produce.

the collision occurs. Fig. 6(b) shows that the adversary sometimes crosses non-drivable areas in order to collide with the planner. Though this scenario is technically possible, it is extreme behavior that may not be desired. However, these situations can be easily detected and discarded, and usually occur only when there is no other feasible adversary near the planner. Finally, adversarial optimization can have difficulty exhibiting behavior that is very unlikely under the prior even when it is realistic, e.g. a parked car pulling out as shown in Fig. 6(c), since these motions are rare in the traffic model training data.

## References

- [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 1
- [2] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multi-modal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 1, 2, 5, 7
- [3] Sergio Casas, Cole Gulino, Simon Suo, Katie Luo, Renjie Liao, and Raquel Urtasun. Implicit latent variable model for scene-consistent motion forecasting. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*, pages 624–641. Springer, 2020. 1, 7
- [4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 3, 4
- [5] Jason Kong, Mark Pfeiffer, Georg Schilb, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 1094–1099. IEEE, 2015. 2
- [6] Yecheng Jason Ma, Jeevana Priya Inala, Dinesh Jayaraman, and Osbert Bastani. Likelihood-based diverse sampling for trajectory forecasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13279–13288, 2021. 7
- [7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems*, 2017. 3
- [8] Philip Polack, Florent Althé, Brigitte d’Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 812–818, 2017. 2
- [9] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, pages 683–700. Springer, 2020. 7
- [10] Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10400–10409, 2021. 1, 2, 7, 8
- [11] Jingkan Wang, Ava Pun, James Tu, Sivabalan Manivasagam, Abbas Sadat, Sergio Casas, Mengye Ren, and Raquel Urtasun. Advsim: Generating safety-critical scenarios for self-driving vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9909–9918, 2021. 4, 5
- [12] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 1
- [13] Ye Yuan and Kris Kitani. Dlow: Diversifying latent flows for diverse human motion prediction. In *European Conference on Computer Vision*, pages 346–364. Springer, 2020. 7
- [14] Ye Yuan, Xinshuo Weng, Yanglan Ou, and Kris Kitani. Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 7