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

Davis Rempe[1,2] Jonah Philion[2,3,4] Leonidas J. Guibas[1] Sanja Fidler[2,3,4] Or Litany[2]
[1]Stanford University [2]NVIDIA [3]University of Toronto [4]Vector Institute

nv-tlabs.github.io/STRIVE

Abstract

Evaluating and improving planning for autonomous vehicles requires scalable generation of long-tail traffic scenarios. To be useful, these scenarios must be realistic and challenging, but not impossible to drive through safely. In this work, we introduce STRIVE, a method to automatically generate challenging scenarios that cause a given planner to produce undesirable behavior, like collisions. To maintain scenario plausibility, the key idea is to leverage a learned model of traffic motion in the form of a graph-based conditional VAE. Scenario generation is formulated as an optimization in the latent space of this traffic model, effected by perturbing an initial real-world scene to produce trajectories that collide with a given planner. A subsequent optimization is used to find a "solution" to the scenario, ensuring it is useful to improve the given planner. Further analysis clusters generated scenarios based on collision type. We attack two planners and show that STRIVE successfully generates realistic, challenging scenarios in both cases. We additionally "close the loop" and use these scenarios to optimize hyperparameters of a rule-based planner.

1. Introduction

The safety of contemporary autonomous vehicles (AVs) is defined by their ability to safely handle complicated near-collision scenarios. However, these kinds of scenarios are rare in real-world driving, posing a data-scarcity problem that is detrimental to both the development and testing of data-driven models for perception, prediction, and planning. Moreover, the better models become, the more rare these events will be, making these models even harder to train.

A natural solution is to synthesize difficult scenarios in simulation, rather than relying on real-world data, making it easier and safer to evaluate and train AV systems. This approach is especially appealing for *planning*, where the appearance domain gap is not a concern. For example, one can manually design scenarios where the AV may fail by inserting adversarial actors or modifying trajectories, either from scratch or by perturbing a small set of real scenarios. Unfortunately, the manual nature of this approach quickly becomes prohibitively expensive when a large set of scenarios is necessary for training or comprehensive evaluation.

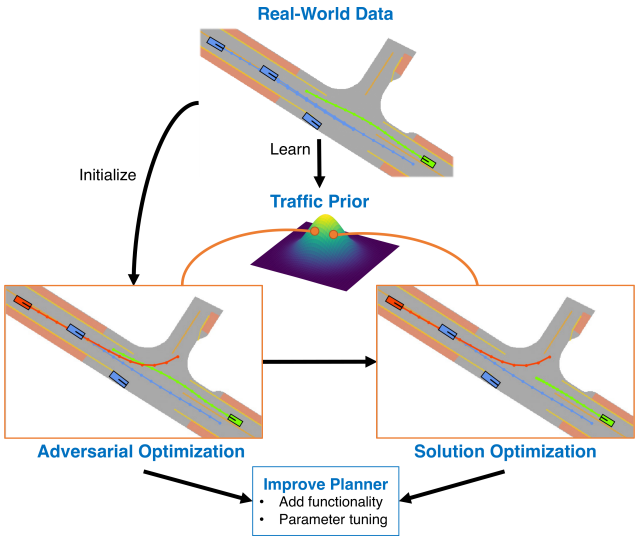

Figure 1. STRIVE generates challenging scenarios for a given planner. An *adversarial optimization* perturbs a real-world scene in the latent space of a learned traffic model, causing an adversary (red) to collide with the planner (green). A subsequent *solution optimization* finds a planner trajectory to avoid collisions, verifying a scenario is useful for identifying planner improvements.

Recent work looks to *automatically generate* challenging scenarios [1, 13, 14, 27, 38, 51, 54]. Generally, these approaches control a single or small group of "adversaries" in a scene, define an objective (*e.g.*, cause a collision with the AV), and then optimize the adversaries' behavior or trajectories to meet this objective. While most methods demonstrate generation of only 1 or 2 scenarios [1, 9, 27, 38], recent work [54] has improved scalability by starting from real-world traffic scenes and perturbing a limited set of prechosen adversaries. However, these approaches *lack expressive priors over plausible traffic motion*, which limits the realism and diversity of scenarios. In particular, adversarial entities in a scenario are a small set of agents heuristically chosen ahead of time; surrounding traffic will not be reactive and therefore perturbations must be careful to avoid implausible situations (*e.g.*, collisions with auxiliary agents). Furthermore, less attention has been given to determining if

a scenario is unsolvable or degenerate [19] (*i.e.* if even an oracle AV is not capable of avoiding the collision), in which case it will not be useful for evaluation or training.

In this work, we introduce STRIVE – a method for generating challenging scenarios to **S**tress-**T**est d**RIVE** a given AV system. STRIVE attacks the prediction, planning, and control subset of the AV stack, which we collectively refer to as the *planner*. As shown in Fig. 1, our approach perturbs an initial real-world scene through an optimization procedure to cause a collision between an arbitrary adversary and a given planner. Our core idea is to measure the plausibility of a scenario during optimization by its likelihood under a learned generative model of traffic motion, which encourages scenarios to be challenging, yet realistic. As a result, STRIVE does not choose specific adversaries ahead of time, rather it jointly optimizes all scene agents, enabling a diverse set of scenarios to arise. Moreover, in order to accommodate for non-differentiable (or inaccessible) planners, which are widely used in practice, the proposed optimization uses a differentiable proxy representation of the planner within the learned motion model, thus allowing standard gradient-based optimization to be used.

We propose to identify and characterize generated scenarios that are *useful* for improving a given planner. We first search for a "solution" to generated scenarios to determine if they are degenerate, and then cluster solvable scenarios based on collision properties. We test STRIVE on two AV planners, including a new rule-based planner, and show that it generates plausible and diverse collision scenarios in both cases. We additionally use generated scenarios to improve the rule-based planner by identifying fundamental limitations of its design and tuning hyperparameters.

In short, our contributions are: (i) a method to automatically generate plausible challenging scenarios for a given planner, (ii) a solution optimization to ensure scenario utility, and (iii) an analysis method to cluster scenarios by collision type. We will release code – including the rule-based planner we created for testing – upon publication to facilitate future work in this area.

2. Related Work

Traffic Motion Modeling. Scenario replay is insufficient for testing and developing AV planners as the motion of non-ego vehicles is strongly coupled to the actions chosen by the ego planner. Advances in deep learning have allowed us to replace traditional dynamic and kinematic models [28, 30, 53] or rule-based simulators [16, 34] with neural counterparts that better capture traffic complexity [3, 39]. Efforts to predict future trajectories from a short state history and an HD map can generally be categorized according to the encoding technique, modeling of multi-modality, multi-agent interaction, and whether the trajectory is estimated in a single step or progressively. The encoding of surrounding context of each agent is often done via a bird's-eye view (BEV) raster image [8, 10, 18], though some work [32] replaces the rasterization-based map encoding with a lane-graph representation. SimNet [4] increased the diversity of generated simulations by initializing the state using a generative model conditioned on the semantic map. To account for multi-modality, multiple futures have been estimated either directly [10] or through trajectory proposals [8, 18, 33, 41]. Modeling multi-actor interactions explicitly using dense graphs has proven effective for vehicles [6, 48], lanes [32], and pedestrians [21, 29, 46]. Finally, step-by-step prediction has performed favorably to one-shot prediction of the whole trajectory [15]. We follow these works and design a traffic model that uses an inter-agent graph network [22] to represent agent interaction and is variational, allowing us to sample multiple futures.

Our model builds on VAE-based approaches [6, 48] that provide a learned prior over a controllable latent space. Among other differences in design, we incorporate a penalty for environment collisions and structure predictions through a bicycle model to ensure physical plausibility.

Challenging Scenario Generation. Generating scenarios has the potential to exponentially increase scene coverage compared to relying exclusively on recorded drives. Advances in photo-realistic simulators like CARLA [16] and NVIDIA's DriveSim, along with the availability of large-scale datasets [5, 17, 20, 24, 47], have been instrumental to methods generating plausible scene graphs to improve perception [11, 12, 23, 43] and planning [4, 6, 25, 48]. Our work focuses on generating challenging – or "adversarial"[1] – scenarios, which are even more crucial since they are so rare in recorded data. While most works assume perfect perception and attack the planning module [9, 13, 14, 19, 27, 51], recent efforts exploit the full stack, including image or point-cloud based perception [1, 31, 38, 49, 54]. Our work focuses on attacking the planner only, though our scene parameterization as a learned traffic model could be incorporated into end-to-end methods. Unlike our approach, which uses gradient-based optimization enabled by the learned motion model, most adversarial generation works rely heavily on black-box optimization which may be slow and unreliable.

Our scenario generation approach is most similar to AdvSim [54], however instead of optimizing acceleration profiles of a simplistic bicycle model we use a more expressive data-driven motion prior. This remedies the previous difficulty of controlling many adversarial agents simultaneously in a plausible manner. Moreover, we avoid constraining the attack trajectory to not collide with the playback AV by proposing a "solution" optimization stage to filter worthwhile scenarios. Prior work [9] clusters lane-change scenarios based on trajectories of agents, while we cluster features

[1] we use "challenging" to address generation procedures that do not explicitly attack a specific module in the perception or planning stack

of the collision between the adversary and planner.

AV Planners. Despite the recent academic interest in end-to-end learning-based planners and AVs [3, 7, 44, 45, 58], rule-based planners remain the norm in practical AV systems [52]. Therefore, we evaluate our approach on a rule-based planner similar to the lane-graph-based planners used by contestants in the 2007 DARPA Urban Challenge [36, 50] detailed in Sec. 4.2.

3. Challenging Scenario Generation

STRIVE aims to generate high-risk traffic situations for a given *planner*, which can subsequently be used to improve that planner (Fig. 1). For our purpose, the planner encapsulates prediction, planning, and control, *i.e.* we're interested in scenarios where the system misbehaves even with perfect perception. The planner takes as input past trajectories of other agents in a scene and outputs the future trajectory of the vehicle it controls (termed the *ego* vehicle). It is assumed to be black-box: STRIVE has no knowledge of the planner's internals and cannot compute gradients through it. Undesirable behavior includes collisions with other vehicles and non-drivable terrain, uncomfortable driving (*e.g.* high accelerations), and breaking traffic laws. We focus on generating **accident-prone scenarios** involving vehicle-vehicle collisions with the planner, though our formulation is general and in principle can handle alternative objectives.

Similar to prior work [54], scenario generation is formulated as an optimization problem that perturbs agent trajectories in an initial scenario from real-world data. The input is a planner f, map $\mathcal{M}$ containing semantic layers for drivable area and lanes, and a snapshot from a pre-recorded real-world scene that serves as initialization for optimization. This initial scenario contains N agents with trajectories represented in 2D BEV as $Y = \{\mathbf{Y}^i\}_{i=1}^N$, where $\mathbf{Y}^i = [\mathbf{y}_1^i, \mathbf{y}_2^i, \dots, \mathbf{y}_T^i]$ is the sequence of states for agent i. We let $\mathbf{Y}_t = [\mathbf{y}_t^1, \mathbf{y}_t^2, \dots, \mathbf{y}_t^N]$ be the state of all agents at a single timestep. An agent state $\mathbf{y}_t^i = [x_t, y_t, \theta_t, v_t, \dot{\theta}_t]$ at time t contains the 2D position (x_t, y_t), heading θ_t, speed v_t, and yaw rate $\dot{\theta}_t$. When rolled out within a scenario, at each timestep the planner outputs the next **ego** state $\mathbf{y}_t^{\text{plan}} = f(\mathbf{y}_{<t}^{\text{plan}}, \mathbf{Y}_{<t}, \mathcal{M})$ based on the *past* motion of itself and other agents. For simplicity, we will write the rolled out planner trajectory as $\mathbf{Y}^{\text{plan}} = f(Y, \mathcal{M})$ where $\mathbf{Y}^{\text{plan}} = [\mathbf{y}_1^{\text{plan}}, \mathbf{y}_2^{\text{plan}}, \dots, \mathbf{y}_T^{\text{plan}}]$ for the remainder of this paper. Scenario generation perturbs trajectories for all non-ego agents to best meet an adversarial objective $\mathcal{L}_{\text{adv}}$ (*e.g.* cause a collision with the planner):

$$\min_{Y} \mathcal{L}_{\text{adv}}(Y, \mathbf{Y}^{\text{plan}}), \quad \mathbf{Y}^{\text{plan}} = f(Y, \mathcal{M}). \tag{1}$$

One may optimize a single or small set of "adversaries" in Y explicitly, *e.g.* through the kinematic bicycle model parameterization [28, 42, 54]. While this enforces plausible single-agent dynamics, **interactions** must be constrained to avoid collisions between non-ego agents and, even then, the resulting traffic patterns may be unrealistic. We propose to instead *learn* to model traffic motion using a VAE-type neural network and then use it at optimization time **(i) to parameterize** all trajectories in a scenario as vectors in the latent space, and **(ii) as a prior** over scenario plausibility. Next, we describe this traffic model, followed by the "adversarial" optimization that produces collision scenarios.

3.1. Modeling "Realism": Learned Traffic Model

We wish to generate accident-prone scenarios that are assumed to develop over short time periods (<10 sec) [37]. Therefore, traffic modeling is formulated as *future forecasting*, which predicts future trajectories for all agents in a scene based on their past motion. We learn $p_\theta(Y|X, \mathcal{M})$ to enable sampling a future scenario Y conditioned on the fixed past $X = \{\mathbf{X}^i\}_{i=1}^N$ (defined similar to Y) and the map $\mathcal{M}$. Two properties of the traffic model make it particularly amenable to downstream optimization: a low-dimensional latent space for efficient optimization, and a prior distribution over this latent space to determine the plausibility of a given scenario. Inspired by recent work [6, 48], we design a conditional variational autoencoder (CVAE), shown in Fig. 2, that meets these criteria while learning accurate and scene-consistent joint future predictions. We briefly introduce the architecture and training procedure here, and refer to the supplement for specific details.

Architecture. To sample future motions at test time, the *conditional prior* and *decoder* are used; both are graph neural networks (GNN) operating on a fully-connected scene graph of all agents. The *prior* models $p_\theta(Z|X, \mathcal{M})$ where $Z = \{\mathbf{z}^i\}_{i=1}^N$ is a set of agent latent vectors. Each node in the input scene graph contains a context feature $\mathbf{h}^i$ extracted from that agent's past trajectory, local rasterized map, bounding box size, and semantic class. After message passing, the *prior* outputs parameters of a Gaussian $p_\theta(\mathbf{z}^i|X, \mathcal{M}) = \mathcal{N}(\mu_\theta^i(X, \mathcal{M}), \sigma_\theta^i(X, \mathcal{M}))$ for each agent in the scene, forming a "distributed" latent representation that captures the variation in possible futures.

The deterministic *decoder* $Y = d_\theta(Z, X, \mathcal{M})$ operates on the scene graph with both a sampled latent $\mathbf{z}^i$ and past context $\mathbf{h}^i$ at each node. Decoding is performed autoregressively: at timestep t, one round of message passing resolves interactions before predicting accelerations $\dot{v}_t, \ddot{\theta}_t$ for each agent. Accelerations immediately go through the kinematic bicycle model [28, 42] to obtain the next state $\mathbf{y}_{t+1}^i$, which updates $\mathbf{h}^i$ before continuing rollout. The determinism and graph structure of the decoder encourages scene-consistent futures even when agent $\mathbf{z}$'s are independently sampled. Importantly for latent optimization, the decoder ensures plausible vehicle dynamics by using the kinematic bicycle model, even when input Z is unlikely.

Training. Training is performed on (X, Y_{gt}) pairs using a

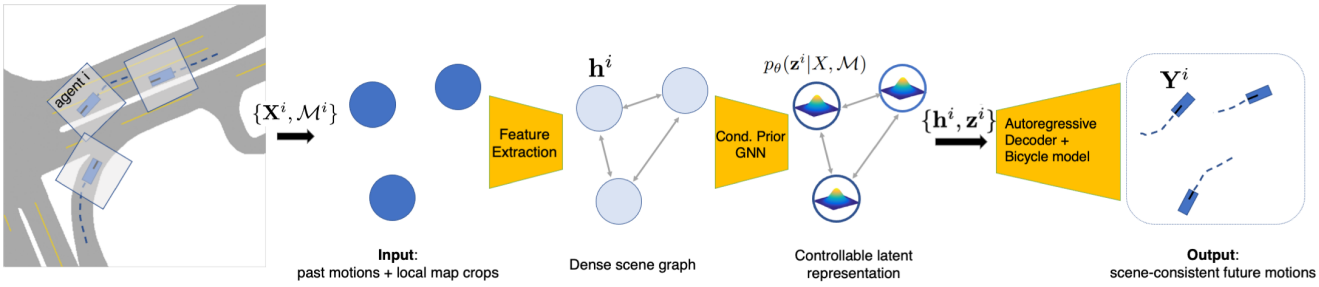

Figure 2. Test-time architecture of the learned traffic model. To jointly sample future trajectories for all agents in a scene, past motion and local map context is first processed individually for each agent. The *conditional prior*, then outputs a latent distribution at each node that can be sampled and fed through the *autoregressive decoder* to predict future agent trajectories.

modified CVAE objective:

$$\mathcal{L}_{\text{cvae}} = \mathcal{L}_{\text{recon}} + w_{\text{KL}}\mathcal{L}_{\text{KL}} + w_{\text{coll}}\mathcal{L}_{\text{coll}}. \tag{2}$$

To optimize this loss, a *posterior* network $q_\phi(Z|Y_{\text{gt}}, X, \mathcal{M})$ is introduced similar to the prior, but operating jointly on past and future motion. Future trajectory features are extracted separately, while past features are the same as used in the *prior*. The full training loss uses trajectory samples from both the posterior Y_{post} and prior Y_{prior}:

$$\mathcal{L}_{\text{recon}} = \sum_{i=1}^{N} ||\mathbf{Y}^i_{\text{post}} - \mathbf{Y}^i_{\text{gt}}||^2 \tag{3}$$

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}(q_\phi(Z|Y_{\text{gt}}, X, \mathcal{M})||p_\theta(Z|X,\mathcal{M})) \tag{4}$$

$$\mathcal{L}_{\text{coll}} = \mathcal{L}_{\text{agent}} + \mathcal{L}_{\text{env}} \tag{5}$$

with $\mathbf{Y}^i_{\text{post}} \in Y_{\text{post}}$, $\mathbf{Y}^i_{\text{gt}} \in Y_{\text{gt}}$. The collision penalties $\mathcal{L}_{\text{agent}}$ and $\mathcal{L}_{\text{env}}$ use a differentiable approximation of collision detection as in [48], which represents vehicles by sets of discs to penalize Y_{prior} for collisions between agents or with the non-drivable map area.

3.2. Adversarial Optimization

To leverage the learned traffic model, the real-world scenario used to initialize optimization is split into the past X and future Y_{init}. Throughout optimization, past trajectories in X (including that of the planner) are *fixed* while the future is perturbed to cause a collision with the given planner f. This perturbation is done in the learned latent space of the traffic model – as described below, we optimize the set of latents for all N non-ego agents $Z = \{\mathbf{z}^i\}_{i=1}^N$ along with a latent representation of the planner $\mathbf{z}^{\text{plan}}$.

Latent scenario parameterization encourages plausibility in two ways. First, since the decoder is trained on real-world data, it will output realistic traffic patterns if Z stays near the learned manifold. Second, the learned prior network gives a distribution over latents, which is used to penalize unlikely Z. This strong prior on behavioral plausibility enables jointly optimizing *all agents* in the scene rather than choosing a small set of specific "adversaries" in advance.

At each step of optimization (see Fig. 3), the perturbed

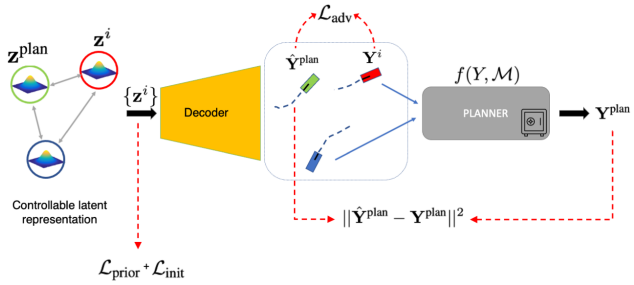

Figure 3. At each step of adversarial optimization, latent representations of both the planner and non-ego agents are decoded with the learned decoder, and non-ego trajectories are given to the planner for rollout within the scenario. Finally, losses are computed.

scenario is decoded with $d_\theta(Z, \mathbf{z}^{\text{plan}}, X, \mathcal{M})$ and non-ego trajectories Y are given to the (black-box) planner which is rolled out in the scene before losses are computed. Adversarial optimization seeks two simultaneous objectives:

1. Match Planner. Although optimization has no direct control over the planner's behavior – it is an external function that is queried when required – it is still necessary to represent the planner within the traffic model (*i.e.* include it in the scene graph with an associated latent $\mathbf{z}^{\text{plan}}$) so that interactions with other agents are realistic. Future predictions from the decoder will then include an estimate of the planner trajectory $\hat{\mathbf{Y}}^{\text{plan}}$ that, ideally, is close to the true planner trajectory $\mathbf{Y}^{\text{plan}} = f(Y, \mathcal{M})$. At the same time, this gives a *differentiable approximation* of the planner, which enables typical gradient-based optimization to be used for the second objective described below. To match the real planner output with this "internal" approximation, we use

$$\min_{\mathbf{z}^{\text{plan}}} ||\hat{\mathbf{Y}}^{\text{plan}} - \mathbf{Y}^{\text{plan}}||^2 - \alpha \log p_\theta(\mathbf{z}^{\text{plan}}|X,\mathcal{M}) \tag{6}$$

where the right term lightly regularizes $\mathbf{z}^{\text{plan}}$ to stay likely under the learned prior and α balances the two terms.

2. Collide with Planner. The goal for non-ego agents is to cause the planner to collide with another vehicle:

$$\min_{Z} \mathcal{L}_{\text{adv}} + \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{init}} + \mathcal{L}_{\text{coll}}. \tag{7}$$

The *adversarial* term encourages a collision by minimizing the positional distance between controlled agents and the current traffic model approximation of the planner:

$$\mathcal{L}_{\text{adv}} = \sum_{i=1}^{N} \sum_{t=1}^{T} ||\mathbf{y}_t^i - \hat{\mathbf{y}}_t^{\text{plan}}||^2 \cdot \delta_t^i \tag{8}$$

$$\delta_t^i = \frac{\exp(-||\mathbf{y}_t^i - \hat{\mathbf{y}}_t^{\text{plan}}||)}{\sum_j \sum_t \exp(-||\mathbf{y}_t^j - \hat{\mathbf{y}}_t^{\text{plan}}||)} \tag{9}$$

where $\mathbf{y}_t$ here only includes the 2D position. Intuitively, the δ_t^i coefficients defined by the *softmin* in Eq. (9) are finding a candidate agent and timestep to collide with the planner. The agent with the largest δ_t^i is the most likely "adversary" based on distance, and Eq. (8) prioritizes causing a collision between this adversary and the planner while still allowing gradients to reach other agents. This weighting helps $\mathcal{L}_{\text{prior}}$ to avoid all agents unrealistically colliding with the planner.

The *prior* term encourages latents to stay likely under the learned prior network:

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

$$\gamma^i = 1 - \sum_t \delta_t^i. \tag{11}$$

The γ^i coefficient will weight likely adversaries near zero, *i.e.* agents close to colliding with the planner are allowed to deviate from the learned traffic manifold to exhibit rare and challenging behavior. Because the traffic model training data does not contain collisions, we found it difficult for an agent to collide with the planner using a large prior loss, thus motivating the weighting in Eq. (11). Note that even when γ^i is small, agents will maintain physical plausibility since the decoder uses the kinematic bicycle model.

$\mathcal{L}_{\text{init}}$ encourages staying close to the initialization in latent space, since it is already known to be realistic:

$$\mathcal{L}_{\text{init}} = \frac{1}{N} \sum_{i=1}^{N} \gamma^i ||\mathbf{z}^i - \mathbf{z}_{\text{init}}^i||^2 \tag{12}$$

where $\mathbf{z}_{\text{init}}^i \in Z_{\text{init}}$ are the latents that initialize optimization. Finally, similar to CVAE training, $\mathcal{L}_{\text{coll}}$ discourages non-ego agents from colliding with each other and the non-drivable area. In practice, all loss terms are balanced by manual inspection of a small set of generated scenarios.

Initialization and Optimization. Given a real-world scene, Z_{init} is obtained through the posterior network q_ϕ, then further refined with an initialization optimization that fits to the input future trajectories of all agents (similar to Eq. (6)), including the initial planner rollout. Optimization is implemented in PyTorch [40] using ADAM [26] with a learning rate of 0.05. Runtime depends on the planner and number of agents; for our rule-based planner (see Sec. 4.2), a 10-agent scenario takes 6-7 minutes.

4. Analyzing and Using Generated Scenarios

4.1. Filtering and Collision Classification

Solution Optimization. Though adversarial optimization produces plausible scenarios, it cannot guarantee they are "solvable": for instance a scenario in which the ego car is squeezed by n other cars produces an unavoidable collision. Therefore, an additional optimization is performed to identify one possible solution to avoid the collision; if this optimization fails, the scenario is discarded for downstream tasks. The solution optimization is initialized from the output of the adversarial optimization and essentially inverts the objectives described in Sec. 3.2: non-ego latent Z are tuned to maintain the adversarial trajectories, while $\mathbf{z}^{\text{plan}}$ is optimized to avoid collisions and stay likely under the prior. Please see the supplementary for more details.

Clustering and Labeling. To gain insight into the distribution of collision scenarios and inform their downstream use, we propose a simple approach to cluster and label them. Scenarios are characterized by the relationship between the planner and adversary at collision time: the relative position and heading of the adversary are computed in the local frame of the planner, and the position is normalized to give the direction of collision. These collision features are clustered with k-means to form semantically similar groups of accidents that are labeled by visual inspection. Their distribution can then be visualized as in Fig. 6.

4.2. Improving the Planner

With a large set of labeled collision scenarios, the planner can be improved in two main ways. First, discrete improvements to functionality may be needed if many scenarios of the same type are generated. For example, a planner that strictly follows lanes is subject to collisions from head on or behind as it fails to swerve, indicating necessary new functionality to leave the lane graph. Second, scenarios provide data for tuning hyperparameters or learned parameters.

Rule-based Planner. To demonstrate how STRIVE scenarios are used for these kinds of improvements, we introduce a simple, yet competent, rule-based planner that we use as a proxy for a real-world planner. Our rule-based planner mimics the structure of motion planners used by successful teams from the 2007 DARPA Urban Challenge [36, 50]. In short, it relies entirely on the lane graph to both predict future trajectories of non-ego vehicles and generate candidate trajectories for the ego vehicle. Among these candidates, it chooses that which covers the most distance with a low "probability of collision." Planner behavior is affected by hyperparameters such as maximum speed/acceleration and how collision probability is computed. This planner has the additional limitation that it cannot change lanes, a limitation that the scenarios generated by STRIVE exposes. The full details of this planner are included in the supplementary.

5. Experiments

We next highlight the new capabilities that STRIVE enables. Sec. 5.1 demonstrates the ability to generate challenging and useful scenarios on two different planners; these scenarios contain a diverse set of collisions, as shown through analysis in Sec. 5.2. Generated scenarios are used to improve our rule-based planner in Sec. 5.3.

Dataset. The nuScenes dataset [5] is used both to train the traffic model and to initialize adversarial optimization. It contains $20s$ traffic clips annotated at 2 Hz, which we split into $8s$ scenarios. Only *car* and *truck* vehicles are used and the traffic model operates on the rasterized *drivable area*, *carpark area*, *road divider*, and *lane divider* map layers. We use the splits and settings of the nuScenes prediction challenge which is $2s$ (4 steps) of past motion to predict $6s$ (12 steps) of future, meaning collision scenarios are $8s$ long, but only the future $6s$ trajectories are optimized.

Planners. Scenario generation is evaluated on two different planners. The *Replay* planner simply plays back the ground truth ego trajectory from nuScenes data. This is an open-loop setting where the planner's $6s$ future is fully rolled out without re-planning. The *Rule-based* planner, described in Sec. 4.2, allows a more realistic closed-loop setting where the planner reacts to the surrounding agents during future rollout by re-planning at 5 Hz.

Metrics. The **collision rate** is the fraction of optimized initial scenarios from nuScenes that succeed in causing a planner collision, which indicates the sample efficiency of scenario generation. **Solution rate** is the fraction of these colliding scenarios for which a solution was found, which measures how often scenarios are *useful*. **Acceleration** indicates how comfortable a driven trajectory is; challenging scenarios should generally increase acceleration for the planner, while the adversary's acceleration should be reasonably low to maintain plausibility. If a scenario contains a collision, acceleration (and other trajectory metrics) is only calculated up to the time of collision. **Collision velocity** is the relative speed between the planner and adversary at the time of collision; it points to the severity of a collision.

5.1. Scenario Generation Evaluation

First, we demonstrate that STRIVE generates challenging, yet solvable, scenarios causing planners to collide and drive uncomfortably. Moreover, compared to an alternative generation approach that *does not* leverage the learned traffic prior, STRIVE scenarios are more plausible. Scenario generation is initialized from 1200 $8s$ sequences from nuScenes. Before adversarial optimization, scenes are prefiltered heuristically by how likely they are to produce a useful collision, leaving $<$500 scenarios to optimize.

Planner-Specific Scenarios. Tab. 1 shows that compared to rolling out a given planner on "regular" (unmodified) nuScenes scenarios, challenging scenarios from STRIVE produce more collisions and less comfortable driving. For the *Rule-based* planner, metrics on challenging scenarios are compared to the corresponding set of regular scenarios from which they originated (regular scenarios for *Replay* are omitted since nuScenes data contains no collisions and, by definition, planner behavior does not change).

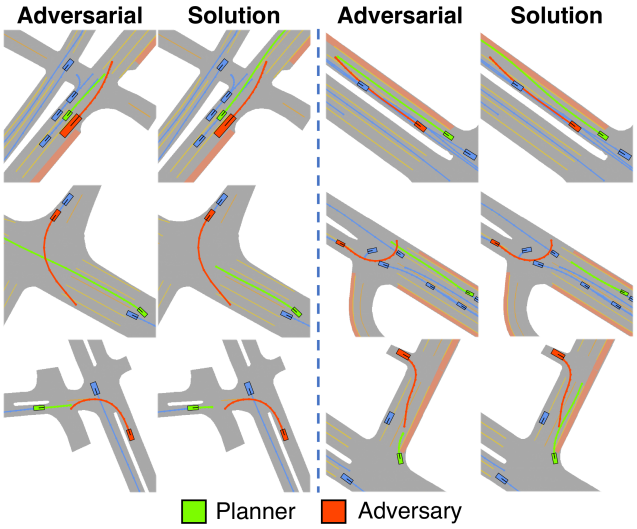

Figure 4. Qualitative results on the *Rule-based* planner. Adversarial and solution optimization results are shown. STRIVE produces diverse collision scenarios including lane changes, (u-)turning in front of the planner, and pulling into oncoming traffic.

Collision and solution rates indicate that generated scenarios are accident-prone and *useful* (solvable). For the *Rule-based* planner, adversarial optimization causes collisions in 27.4% of scenarios compared to only 1.2% in the regular scenarios. Generated scenarios also contain more severe collisions in terms of velocity, and elicit larger accelerations, *i.e*. less comfortable driving. The position and angle errors between approximate ($\hat{\mathbf{Y}}^{\text{plan}}$) and true ($\mathbf{Y}^{\text{plan}}$) planner trajectories at the end of adversarial optimization are shown on the right (see 3.2). The largest position error of $1.23m$ is reasonable relative to the $4.084m$ length of the planner vehicle. Qualitative results for the *Rule-based* planner visualize 2D waypoint trajectories (Fig. 4); though not shown, STRIVE also generates speed and heading.

Baseline Comparison. STRIVE is next compared to a baseline approach to demonstrate that leveraging a learned traffic model is key to plausible and useful scenarios. Previous works are not directly comparable as they focus on small-scale scenario generation (*e.g*. [1,9]) and/or attack the full AV stack rather than just the planner [38, 54]. Therefore, in the spirit of AdvSim [54] we implement the *Bicycle* baseline, which explicitly optimizes the kinematic bicycle model parameters (acceleration profile) of a single prechosen adversary in the scenario to cause a collision. Rather than using the learned traffic model, it relies on the bicycle

Planner	Scenarios	Collision (%)	Solution (%)	Planner Trajectory Accel (m/s^2)	Planner Trajectory Coll Vel (m/s)	Match Planner Err Pos (m)	Match Planner Err Ang (deg)
Replay	Challenging	43.7 (**+43.7**)	82.4	0.85	7.82	0.28	1.32
Rule-based	Regular	1.2	–	1.63	8.48	–	–
Rule-based	Challenging	27.4 (**+26.2**)	86.8	1.91 (**+0.28**)	9.65 (**+1.17**)	1.23	3.79

Table 1. Evaluation of generated *challenging* scenarios. Generated scenarios contain far more collisions compared to the corresponding *regular* (unmodified) scenes, as well as higher acceleration and collision speeds. Acceleration is measured in the forward direction (*i.e.* change in speed), since the *Rule-based* planner cannot change lanes. Rightmost columns show small errors between $\hat{\mathbf{Y}}^{\text{plan}}$ and $\mathbf{Y}^{\text{plan}}$.

Scenarios	Plausibility of Adversary Trajectory ↓ Accel (m/s^2)	Env Coll (%)	NN Dist (m)	NLL	Usefulness ↑ Solution (%)
Bicycle	2.00	16.5	0.97	962.9	73.4
STRIVE	**0.98**	**10.8**	**0.72**	**323.4**	**83.5**

Table 2. Scenario generation for *Replay* planner compared to the *Bicycle* baseline, which does not leverage a learned traffic model.

model, collision penalties, and acceleration regularization to maintain plausibility. This precludes using the differentiable planner approximation from the traffic model, thus requiring gradient estimation (*e.g.* finite differences) for the closed-loop setting, which we found is $\approx 40\times$ slower and requires several hours to generate a single scenario. Therefore, comparison is done only on the *Replay* planner.

Tab. 2 shows that scenarios generated by *Bicycle* exhibit more unrealistic adversarial driving, and are more difficult to find a solution for. All metrics are reported for scenarios where both methods successfully caused a collision. In addition to higher accelerations, the *Bicycle* adversary collides with the non-drivable area more often (*Env Coll*), and exhibits less typical trajectories as measured by the distance to the nearest-neighbor ego trajectory in the nuScenes training split (*NN Dist*). After fitting the *Bicycle*-generated scenarios with our traffic model, we see the adversary's behavior is also less realistic as measured by the negative log-likelihood (NLL) of its latent $\mathbf{z}$ under the learned prior. These observations are supported qualitatively in Fig. 5.

5.2. Analyzing Generated Scenarios

Before improving the given planner, the analysis from Sec. 4.1 is used to identify useful scenarios by filtering out unsolvable scenarios and classifying collision types. For classification, collision features are clustered with $k = 10$ and clusters are visualized to manually assign the semantic labels shown in Fig. 6. The distribution of generated collision scenarios for both planners in Sec. 5.1 is shown in Fig. 6(a) (see supplement for visualized examples from clusters). STRIVE generates a diverse set of scenarios with solvable scenes found in all clusters. "Head On" is the most frequently generated scenario type, likely because *Replay* is non-reactive and *Rule-based* cannot change lanes. "Behind" exhibits the highest rate of unsolvable scenarios since being hit from behind is often the result of a negligent following vehicle, rather than undesirable planner behavior. *Replay* is much more susceptible to being cut off since it is open-loop, while the closed-loop *Rule-based* can successfully react to avoid such collisions.

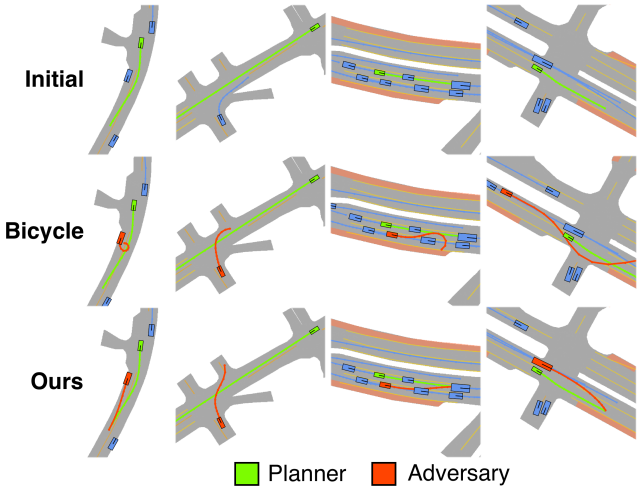

Figure 5. Qualitative comparison of generated scenarios for the *Replay* planner. *Bicycle* often produces semantically unrealistic trajectories (e.g. sharp turns and erratic swerving) as no learned traffic model is leveraged.

5.3. Improving Rule-Based Planner

Now that we have a large set of labeled collision scenarios, in addition to the original nuScenes data containing typical scenarios (with few collisions), we can *improve* the *Rule-based* planner to be better prepared for challenging situations. Besides uncovering fundamental flaws that lead us to add new functionality, improvement is based on hyperparameter tuning via a grid search over possible settings (see Sec. 4.2). For each set of hyperparameters, the planner is rolled out within all scenarios of a dataset, and the optimal tuning is chosen based on collision rate.

The planner is first tuned on regular scenarios before adversarial optimization is performed to create a set of challenging scenarios to guide further improvements. Performance of this initial regular-tuned planner on held out regular (*Reg*) and collision (*Coll*) scenarios is shown in the top row of Tab. 3. Before any improvements, the planner collides in 68.6% of challenging and 4.6% of regular scenarios.

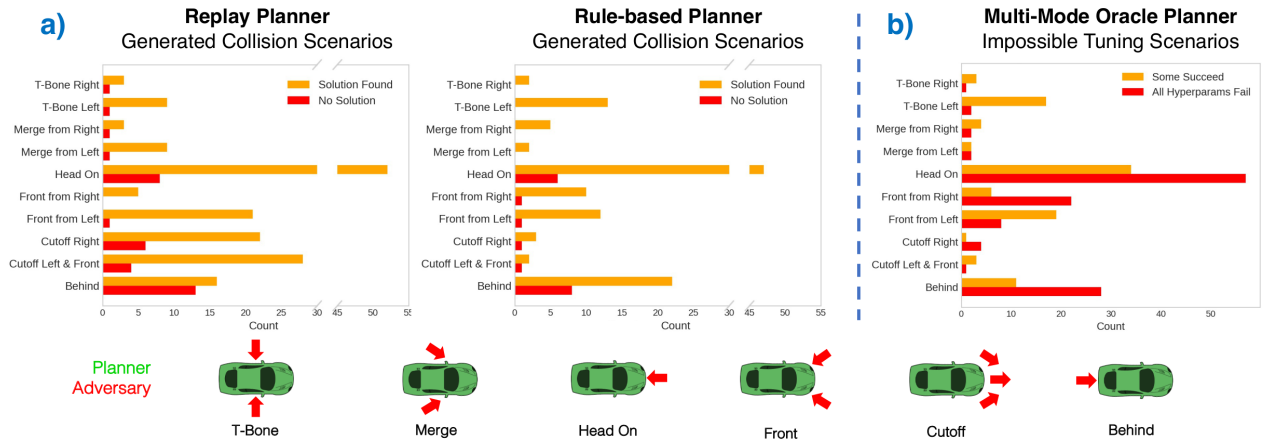

Figure 6. (Bottom) Collision types are depicted; the arrow indicates the position/direction of the adversary. **(a)** Distribution of generated scenarios for both planners. **(b)** Scenarios used to tune the multi-mode *oracle* planner. Scenarios where all parameter settings cause a collision are in red. A majority of *Head On*, *Front from Right*, and *Behind* scenarios always fail due to the inability to change lanes.

Improvement	**Collision** (%) Reg / Coll	**Coll Vel** (m/s) Reg / Coll	**Accel** (m/s^2) Reg / Coll
None (regular-tuned)	4.6 / 68.6	4.59 / 10.48	1.96 / 2.26
+ Challenging data	6.0 / 51.4	5.48 / 13.88	2.29 / 2.50
+ Extra *learned* mode	4.6 / 54.3	4.60 / 10.86	2.02 / 2.55
+ Extra *oracle* mode	4.6 / 54.3	4.59 / 10.40	1.96 / 2.39

Table 3. Results for improving *Rule-Based* planner. By including challenging tuning data and adding an extra set (mode) of hyperparameters, performance on collision scenarios (*Coll*) is greatly improved while maintaining low collision rates in regular scenarios (*Reg*). Acceleration is measured in the forward direction.

Note that avoiding collisions altogether on regular scenarios is not possible – even if we choose optimal hyperparameters per scenario, the collison rate is still 3.2%.

Tuning on Challenging Scenes. The first improvement, shown in the second row of Tab. 3, is to naïvely combine regular and challenging scenarios for tuning. Combined tuning greatly reduces the collision rate on challenging scenarios, but negatively impacts performance on regular driving. This points to a first *fundamental issue*: the planner uses a single set of hyperparameters for all driving situations without considering the context.

Multi-Mode Operation. To address this, we add a second set of parameters such that the planner has one for regular and one for accident-prone situations. Using this second "accident mode" of operation requires a binary classification of the current scene during rollout. For this, we augment the planner with a learned component that decides which parameter set to use based on a moving window of the past $2s$ of traffic; it is trained on scenarios generated by STRIVE. The extra parameter set is tuned on collision scenarios only. As shown in the third row of Tab. 3, this learned extra mode reduces the collision rate on challenging scenarios by 14.3% compared to the vanilla planner without hindering performance on regular scenes. We compare it to an *oracle* version (bottom row) that automatically switches into accident mode $2s$ before a collision is supposed to happen on generated scenarios, showing the learned version is achieving near-optimal performance.

Lane Change Limitation. The inability of the planner to switch lanes is another fundamental issue exposed by collision scenarios. Fig. 6(b) shows the distribution of tuning scenarios for the *oracle* multi-mode version; red bars indicate "impossible" scenarios where all possible sets of hyperparameters collide. A majority of "Head On" and "Behind" scenarios are impossible, pointing out the lane change limitation. Adversarial optimization has indeed exploited the flaw and the proposed analysis made it visible.

6. Discussion

STRIVE enables automatic and scalable generation of plausible, accident-prone scenarios to improve a given planner. However, remaining limitations offer potential future directions. Our method assumes perfect perception and only attacks the planner, but using our traffic model scenario parameterization to additionally attack detection and tracking is of great interest. The current version of STRIVE generates scenarios from existing data and only considers collisions between vehicles, but other dangerous or uncomfortable scenarios, such as incidents involving pedestrians and cyclists, are also important. Furthermore, other kinds of adversaries like adding/removing assets and changing map topology will uncover additional AV weaknesses.

Our method is intended to make AVs safer by exposing them to challenging and rare scenarios similar to the real world. However, as we've shown, naïvely tuning a planner on adversarial scenarios can cause more dangerous behavior in "normal" settings. Care must be taken to integrate generated scenarios into AV testing and design in a safe way.

Acknowledgments. Author Guibas was supported by a Vannevar Bush faculty fellowship. The authors thank Amlan Kar for the fruitful discussion and feedback throughout the project.

References

[1] Yasasa Abeysirigoonawardena, Florian Shkurti, and Gregory Dudek. Generating adversarial driving scenarios in high-fidelity simulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8271–8277. IEEE, 2019. 1, 2, 6

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 12

[3] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst, 2018. 2, 3

[4] Luca Bergamini, Yawei Ye, Oliver Scheel, Long Chen, Chih Hu, Luca Del Pero, Błażej Osiński, Hugo Grimmet, and Peter Ondruska. Simnet: Learning reactive self-driving simulations from real-world observations. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages –. IEEE, 2021. 2

[5] 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 multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 2, 6, 12, 13, 16, 18

[6] 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. 2, 3, 12, 18

[7] Sergio Casas, Abbas Sadat, and Raquel Urtasun. Mp3: A unified model to map, perceive, predict and plan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14403–14412, 2021. 3

[8] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *arXiv preprint arXiv:1910.05449*, 2019. 2

[9] Baiming Chen, Xiang Chen, Qiong Wu, and Liang Li. Adversarial evaluation of autonomous vehicles in lane-change scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 2021. 1, 2, 6

[10] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2090–2096. IEEE, 2019. 2

[11] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. Meta-sim2: Unsupervised learning of scene structure for synthetic data generation, 2020. 2

[12] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. Meta-sim2: Unsupervised learning of scene structure for synthetic data generation. In *European Conference on Computer Vision*, pages 715–733. Springer, 2020. 2

[13] Wenhao Ding, Baiming Chen, Bo Li, Kim Ji Eun, and Ding Zhao. Multimodal safety-critical scenarios generation for decision-making algorithms evaluation. *IEEE Robotics and Automation Letters*, 6(2):1551–1558, 2021. 1, 2

[14] Wenhao Ding, Baiming Chen, Minjun Xu, and Ding Zhao. Learning to collide: An adaptive safety-critical scenarios generating method. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2243–2250. IEEE, 2020. 1, 2

[15] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, Nitin Singh, and Jeff Schneider. Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2095–2104, 2020. 2

[16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017. 2

[17] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles Qi, Yin Zhou, et al. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. *ICCV*, 2021. 2

[18] Liangji Fang, Qinhong Jiang, Jianping Shi, and Bolei Zhou. Tpnet: Trajectory proposal network for motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6797–6806, 2020. 2

[19] Zahra Ghodsi, Siva Kumar Sastry Hari, Iuri Frosio, Timothy Tsai, Alejandro Troccoli, Stephen W Keckler, Siddharth Garg, and Anima Anandkumar. Generating and characterizing scenarios for safety testing of autonomous vehicles. *arXiv preprint arXiv:2103.07403*, 2021. 2

[20] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset. https://level-5.global/level5/data/, 2020. 2

[21] Boris Ivanovic and Marco Pavone. The trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2375–2384, 2019. 2

[22] Daniel D. Johnson, Hugo Larochelle, and Daniel Tarlow. Learning graph structure with a finite-state automaton layer, 2020. 2

[23] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4551–4560, 2019. 2

[24] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Level 5 perception dataset 2020. https://level-5.global/level5/data/, 2019. 2

[25] Seung Wook Kim, , Jonah Philion, Antonio Torralba, and Sanja Fidler. DriveGAN: Towards a Controllable High-Quality Neural Simulation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021. 2

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 5, 13, 14, 15

[27] Moritz Klischat and Matthias Althoff. Generating critical test scenarios for automated vehicles with evolutionary algorithms. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2352–2358. IEEE, 2019. 1, 2

[28] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. pages 1094–1099, 06 2015. 2, 3, 13

[29] Vineet Kosaraju, Amir Sadeghian, Roberto Martín-Martín, Ian Reid, S Hamid Rezatofighi, and Silvio Savarese. Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 2

[30] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *ROBOMECH journal*, 1(1):1–14, 2014. 2

[31] Yiming Li, Congcong Wen, Felix Juefei-Xu, and Chen Feng. Fooling lidar perception via adversarial trajectory perturbation. *arXiv preprint arXiv:2103.15326*, 2021. 2

[32] Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. Learning lane graph representations for motion forecasting. In *European Conference on Computer Vision*, pages 541–556. Springer, 2020. 2

[33] Yicheng Liu, Jinghuai Zhang, Liangji Fang, Qinhong Jiang, and Bolei Zhou. Multimodal motion prediction with stacked transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7577–7586, 2021. 2

[34] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582. IEEE, 2018. 2

[35] 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. 18

[36] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Hähnel, Tim Hilden, Gabriel Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, and Sebastian Thrun. Junior: The stanford entry in the urban challenge. pages 91–123, 01 2009. 3, 5

[37] Wassim G Najm, John D Smith, and Mikio Yanagisawa. *Pre-Crash Scenario Typology for Crash Avoidance Research*. U.S. Department of Transportation, National Highway Traffic Safety Administration, 2007. 3

[38] Matthew O'Kelly, Aman Sinha, Hongseok Namkoong, Russ Tedrake, and John C Duchi. Scalable end-to-end autonomous vehicle testing via rare-event simulation. In *NeurIPS*, 2018. 1, 2, 6

[39] Avik Pal, Jonah Philion, Yuan-Hong Liao, and Sanja Fidler. Emergent road rules in multi-agent driving environments. In *International Conference on Learning Representations*, 2020. 2

[40] 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. 5, 13

[41] Jonah Philion, Amlan Kar, and Sanja Fidler. Learning to evaluate perception models using planner-centric metrics. In *CVPR*, 2020. 2

[42] Philip Polack, Florent Altché, 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. 3, 13

[43] Cinjon Resnick, Or Litany, Amlan Kar, Karsten Kreis, James Lucas, Kyunghyun Cho, and Sanja Fidler. Causal bert: Improving object detection by searching for challenging groups. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 2972–2981, October 2021. 2

[44] Abbas Sadat, Sergio Casas, Mengye Ren, Xinyu Wu, Pranaab Dhawan, and Raquel Urtasun. Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. In *European Conference on Computer Vision*, pages 414–430. Springer, 2020. 3

[45] Abbas Sadat, Mengye Ren, Andrei Pokrovsky, Yen-Chen Lin, Ersin Yumer, and Raquel Urtasun. Jointly learnable behavior and trajectory planning for self-driving vehicles. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3949–3956. IEEE, 2019. 3

[46] 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. 2, 18

[47] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2446–2454, 2020. 2

[48] 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. 2, 3, 4, 12, 13, 18

[49] James Tu, Mengye Ren, Sivabalan Manivasagam, Ming Liang, Bin Yang, Richard Du, Frank Cheng, and Raquel

Urtasun. Physically realizable adversarial examples for lidar object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13716–13725, 2020. 2

[50] Christopher Urmson, Joshua Anhalt, J. Andrew (Drew) Bagnell, Christopher R. Baker, Robert E. Bittner, John M. Dolan, David Duggins, David Ferguson, Tugrul Galatali, Hartmut Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Alonzo Kelly, David Kohanbash, Maxim Likhachev, Nick Miller, Kevin Peterson, Raj Rajkumar, Paul Rybski, Bryan Salesky, Sebastian Scherer, Young-Woo Seo, Reid Simmons, Sanjiv Singh, Jarrod M. Snider, Anthony (Tony) Stentz, William (Red) L. Whittaker, and Jason Ziglar. Tartan racing: A multi-modal approach to the darpa urban challenge. Technical report, Carnegie Mellon University, Pittsburgh, PA, April 2007. 3, 5

[51] Sai Vemprala and Ashish Kapoor. Adversarial attacks on optimization based planners. *arXiv preprint arXiv:2011.00095*, 2020. 1, 2

[52] Matt Vitelli, Yan Chang, Yawei Ye, Maciej Wołczyk, Błażej Osiński, Moritz Niendorf, Hugo Grimmett, Qiangui Huang, Ashesh Jain, and Peter Ondruska. Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies. *arXiv preprint arXiv:2109.13602*, 2021. 3

[53] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000. 2

[54] Jingkang 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. 1, 2, 3, 6, 15

[55] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 12

[56] 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. 18

[57] 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. 18

[58] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8660–8669, 2019. 3

Appendices

Here we include additional technical details and results omitted from the main paper for brevity. Appendix A gives details on the main technical methods, Appendix B gives details for experiments, and Appendix C provides additional results to supplement those in the main paper.

Video Results. We encourage viewing the video results on the project page to get a better sense of the kinds of scenarios produced by STRIVE.

A. 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.

A.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 [2] unless noted otherwise.

State Representation. In practice, trajectories used as input and supervision for the traffic model are represented as $\mathbf{Y}^i = [\mathbf{y}^i_1, \mathbf{y}^i_2, \dots, \mathbf{y}^i_T]$ for agent i with T timesteps where the state at time t is $\mathbf{y}^i_t = [x_t, y_t, \theta^x_t, \theta^y_t, v_t, \dot{\theta}_t] \in \mathbb{R}^6$ containing the 2D position (x_t, y_t), heading unit vector (θ^x_t, θ^y_t), 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 hidden size 128. To handle missing frames where an agent has not been annotated in nuScenes [5], 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 $\mathbf{m}^i \in 64$ is extracted from a local map crop of size 256×256 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 [55].

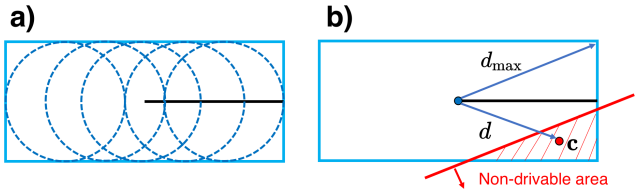

Figure 7. Collision penalties. (a) Computing $\mathcal{L}_{\text{agent}}$ estimates each vehicle by a set of 5 discs. (b) $\mathcal{L}_{\text{env}}$ is computed based on the distance between a collision point $\mathbf{c}$ and the vehicle position.

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) like the scene interaction module [6, 48]. 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 $j \to i$, 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 $\sigma^i \in \mathbb{R}^{32}$ parameterizes the diagonal covariance matrix so that $p_\theta(\mathbf{z}^i | X, \mathcal{M}) = \mathcal{N}(\mu^i_\theta(X, \mathcal{M}), \sigma^i_\theta(X, \mathcal{M}))$ where θ 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 $\mathbf{Y}^i_{\text{gt}}$, $\mathbf{s}^i$, 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 $q_\phi(\mathbf{z}^i | Y_{\text{gt}}, X, \mathcal{M}) = \mathcal{N}(\mu^i_\phi(Y_{\text{gt}}, X, \mathcal{M}), \sigma^i_\phi(Y_{\text{gt}}, X, \mathcal{M}))$ with ϕ the learned pa-

rameters 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 $[\mathbf{z}^i, \mathbf{p}^i_{t-1}, \mathbf{m}^i_{t-1}, \mathbf{s}^i, \mathbf{b}^i]$ 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.* $\mathbf{p}^i_0 = \mathbf{p}^i$ and $\mathbf{m}^i_0 = \mathbf{m}^i$.

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 $\mathbf{o}^i_t = [\dot{v}^i_t, \ddot{\theta}^i_t] \in \mathbb{R}^2$ at each node, which contains current linear and angular acceleration. The kinematic bicycle model [28, 42] is then used to get the actual agent state $\mathbf{y}^i_t = \mathcal{K}(\mathbf{y}^i_{t-1}, \mathbf{o}^i_t, \mathbf{b}^i)$. 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 $\mathbf{p}^i_t = \text{GRU}(\mathbf{p}^i_{t-1}, \mathbf{y}^i_t)$ with 3 layers (the GRU uses a hidden state of size 64 that is omitted here for brevity). The new map feature $\mathbf{m}^i_t$ is extracted using the same CNN as before, but with an updated map crop in the local frame of $\mathbf{y}^i_t$. The feature at each node can then be updated to $[\mathbf{z}^i, \mathbf{p}^i_t, \mathbf{m}^i_t, \mathbf{s}^i, \mathbf{b}^i]$ 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:

$$\mathcal{L}_{\text{cvae}} = \mathcal{L}_{\text{recon}} + w_{\text{KL}}\mathcal{L}_{\text{KL}} + w_{\text{coll}}\mathcal{L}_{\text{coll}} \tag{13}$$

$$\mathcal{L}_{\text{recon}} = \sum_{i=1}^{N} ||\mathbf{Y}^i_{\text{post}} - \mathbf{Y}^i_{\text{gt}}||^2 \tag{14}$$

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}(q_\phi(Z|Y_{\text{gt}}, X, \mathcal{M})||p_\theta(Z|X, \mathcal{M})) \tag{15}$$

$$\mathcal{L}_{\text{coll}} = \mathcal{L}_{\text{agent}} + \mathcal{L}_{\text{env}}. \tag{16}$$

In practice, $\mathcal{L}_{\text{recon}}$ only supervises the position and heading, *i.e.* $\mathbf{Y}^i_{\text{post}}$ and $\mathbf{Y}^i_{\text{gt}}$ in Eq. (14) contain only $[x_t, y_t, \theta^x_t, \theta^y_t]$. 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 [48]. 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. 7(a), and compute the loss by summing over all pairs of agents (i, j) over time as:

$$\mathcal{L}_{\text{agent}} = \frac{1}{N^2} \sum_{(i,j), i \neq j} \sum_{t=1}^{T} \mathcal{L}_{\text{pair}}(\mathbf{y}^i_t, \mathbf{y}^j_t) \tag{17}$$

$$\mathcal{L}_{\text{pair}}(\mathbf{y}^i_t, \mathbf{y}^j_t) = \begin{cases} 1 - \dfrac{d}{r_i + r_j}, & d \leq r_i + r_j \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

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 [5] 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:

$$\mathcal{L}_{\text{env}} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}_{\text{drivable}}(\mathbf{y}_t, \mathcal{M}) \tag{19}$$

$$\mathcal{L}_{\text{drivable}}(\mathbf{y}_t, \mathcal{M}) = \begin{cases} 1 - \dfrac{d}{d_{\text{max}}}, & \text{if partial collision} \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

where d is the distance between the vehicle position (center of bounding box) and collision point $\mathbf{c}$, and d_{max} is half the ego bounding box diagonal as shown in Fig. 7(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 [40] and trained using the ADAM optimizer [26] with learning rate $1e^{-5}$ for 110 epochs. Losses are weighted with $w_{\text{KL}} = 4e^{-3}$ and w_{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_{KL} is linearly annealed over time, starting from 0.0 at the start of training up to the full value at epoch 20.

A.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_{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 roll-

out within the initial scenario) that are contained in Y_{init}:

$$\min_{Z_{\text{init}}} w_{\text{match}}||Y - Y_{\text{init}}||^2 - w_{\text{prior}} \log p_\theta(Z_{\text{init}}|X, \mathcal{M}) \quad (21)$$

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.

A.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 δ 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 $180°$ 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 γ^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

$$\mathcal{L}_{\text{prior}} = -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(\mathbf{z}^i|X, \mathcal{M}) \cdot w_{\text{prior}}(\gamma^i) \quad (22)$$

$$w_{\text{prior}}(\gamma^i) = \gamma^i w_{\text{prior}}^{\text{max}} + (1 - \gamma^i) w_{\text{prior}}^{\text{min}} \quad (23)$$

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 actually uses γ^i to interpolate between max and min hyperparameters and is written:

$$\mathcal{L}_{\text{init}} = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{z}^i - \mathbf{z}_{\text{init}}^i||^2 \cdot w_{\text{init}}(\gamma^i) \quad (24)$$

$$w_{\text{init}}(\gamma^i) = \gamma^i w_{\text{init}}^{\text{max}} + (1 - \gamma^i) w_{\text{init}}^{\text{min}} \quad (25)$$

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. (17) 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. (19) 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 γ^i (*i.e.* unlikely adversaries). This term only affects agents that are close to the planner but have large γ^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 are for a machine with an NVIDIA Titan RTX GPU and 12x Intel i7-7800X@3.50GHz CPUs. We optimize for 200 iterations using the ADAM [26] optimizer (we found L-BFGS is overly prone to local minima for this problem) with learning rate 0.05.

A.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_{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:

$$\min_{Z} ||Y_{\text{adv}} - Y||^2 - \alpha \log p_\theta(Z|X, \mathcal{M}) \quad (26)$$

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:

$$\min_{\mathbf{z}^{\text{plan}}} \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{coll}}. \quad (27)$$

The prior loss is

$$\mathcal{L}_{\text{prior}} = -w_{\text{prior}} \log p_\theta(\mathbf{z}^{\text{plan}}|X, \mathcal{M}) \quad (28)$$

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. (17) but only discourages collisions between the planner and all non-ego agents. $\mathcal{L}_{\text{env}}$ is the same as defined in Eq. (19) 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 [26] optimizer for 200 iterations with a learning rate of 0.05.

A.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 Appendix A.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.

A.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 [54]). For this, we use the candidate adversary determined by the feasibility check in Appendix A.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

$$\min_{\mathbf{A}} ||\mathbf{Y} - \mathbf{Y}_{\text{init}}||^2. \tag{29}$$

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

$$\min_{\mathbf{A}} w_{\text{adv}}\mathcal{L}_{\text{adv}} + w_{\text{accel}}\mathcal{L}_{\text{accel}} + \mathcal{L}_{\text{coll}}. \tag{30}$$

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

$$\mathcal{L}_{\text{adv}} = \sum_{t=1}^{T} ||\mathbf{y}_t - \mathbf{y}_t^{\text{plan}}|| \cdot \delta_t \tag{31}$$

$$\delta_t = \frac{\exp(-||\mathbf{y}_t - \mathbf{y}_t^{\text{plan}}||)}{\sum_t \exp(-||\mathbf{y}_t - \mathbf{y}_t^{\text{plan}}||)}. \tag{32}$$

The *softmin* in equation Eq. (32) 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 [54] – 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:

$$\mathcal{L}_{\text{accel}} = \frac{1}{T}\sum_{t=1}^{T} ||\mathbf{a}_t||^2. \tag{33}$$

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. (17) but only discourages collisions between the adversary and other (fixed) non-ego agents. $\mathcal{L}_{\text{env}}$ is the same as defined in Eq. (19) 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 Appendix A.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 Appendix A.4.

A.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 τ.
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 Δt 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_{max}, and the maximum speed and forward acceleration.

B. Experimental Details

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

B.1. Data and Metrics

Dataset. The nuScenes [5] 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 Appendix A.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 compute 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 Appendix A.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.

B.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).

B.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 Appendix A.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 Appendix A.6 for details on how solution rate and NLL are measured for *Bicycle*.

B.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 in the **supplementary HTML page**.

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 *Rule-based* planner performance while being easily interpretable.

B.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 Appendix B.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 Appendix A.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^2, 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.

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 Appendix A.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.

Model	ADE (m) ↓	FDE (m) ↓
LDS-AF [35]	1.66	3.58
DLow-AF [56]	1.78	3.77
Trajectron++ [46]	1.51	-
AgentFormer [57]	**1.45**	**2.86**
Ours, Full	1.75	3.57
Ours, No Bicycle	1.60	3.17

Table 4. 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	**1.72**	**3.45**	**7.2**	**3.7**
No $\mathcal{L}_{\text{env}}$	1.91	3.92	13.2	5.0
No Autoregress	3.68	8.00	16.2	5.4

Table 5. 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.

C. Supplemental Experiments

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

C.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 [5] 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

$$\text{ADE} = \min_k \frac{1}{T} \sum_{t=1}^{T} ||\hat{\mathbf{y}}_t^{(k)} - \mathbf{y}_t||_2 \tag{34}$$

$$\text{FDE} = \min_k ||\hat{\mathbf{y}}_T^{(k)} - \mathbf{y}_T||_2 \tag{35}$$

where $\hat{\mathbf{y}}_t^{(k)}$ is the predicted *position* of the agent in the kth 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 [48], which counts the number of agents in collision (*i.e*. have a bounding box overlap more than IoU 0.02 with another agent).

C.1.1 Baseline Comparison

Prediction performance is compared to reported results for recent state-of-the art models AgentFormer [57], Trajectron++ [46], DLow-AF [56], and LDS-AF [35]. Results are shown in Tab. 4. 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. 4. 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.

C.1.2 Ablation Study

To evaluate key design differences from TrafficSim [48] and ILVM [6], which our model is based on, we ablate various components of our traffic model design. Results are shown in Tab. 5, 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. 4, *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 [48]. 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.

C.2. Adversarial Optimization Ablation Study

Next, we study how various components of the adversarial optimization objective function (introduced in Sec 3.2

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

Table 6. 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.

of the main paper and detailed in Appendix A.3) affect the generated scenarios. We consider the following variations:

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

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

Results are shown in Tab. 6. 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. 6 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. 6 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 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 allows for some collisions as seen in Appendix C.1.2. *No* $\mathcal{L}_{init}$ 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* $\mathcal{L}_{init}, \gamma$ 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 γ, 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. 8. *No* $\mathcal{L}_{init}, \gamma, \delta$ enables many adversaries to attack simultaneously producing many unsolvable scenarios with unrealistic trajectories.

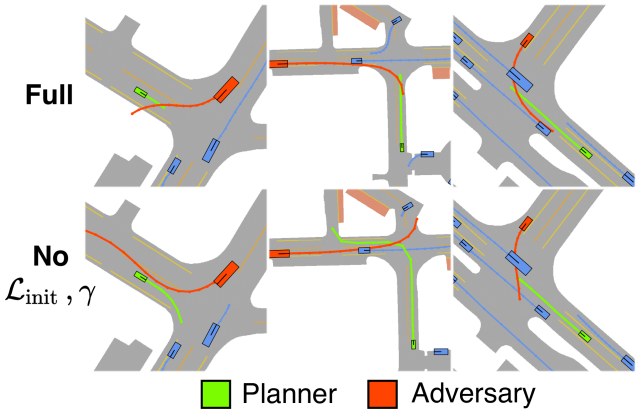

Figure 8. Comparison of generated adversarial scenarios for the *Rule-based* planner using the full objective function and a modified objective with no initialization loss or per-agent weighting in $\mathcal{L}_{\text{prior}}$. Removing per-agent weighting requires adversaries to be more likely under 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.

C.3. Solution Optimization Evaluation

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

C.4. Additional Qualitative Results

Additional qualitative results of STRIVE on the *Rule-based* planner are shown in Fig. 10. Video examples are also included in the supplementary HTML webpage.

C.5. 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 Appendix A.5. Results are shown in Fig. 9 and on the HTML webpage.

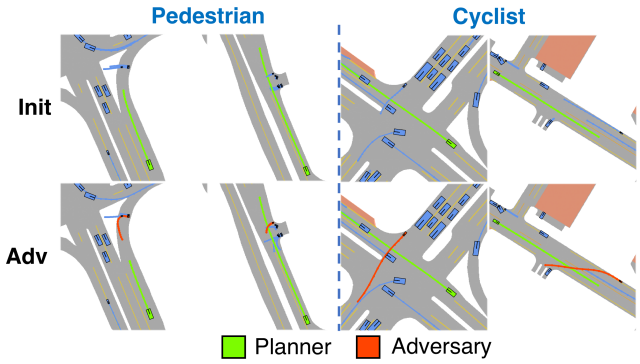

Figure 9. Generated scenarios for the *Replay* planner using a traffic model trained on *all* categories. Top row shows the initial scene and bottom is the output of adversarial optimization. When choosing the adversary ahead of time, STRIVE can cause collisions with both pedestrians (left) and cyclists (right).

C.6. Failure Cases and Limitations

In addition to the limitations discussed in Sec 6 of the main paper, Fig. 11 shows examples of other STRIVE limitations. First, our proposed solution optimization is iterative 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. 11(a) where the optimized solution simply does not not pull into the roundabout where the collision occurs. Fig. 11(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. 11(c), since these motions are rare in the traffic model training data.

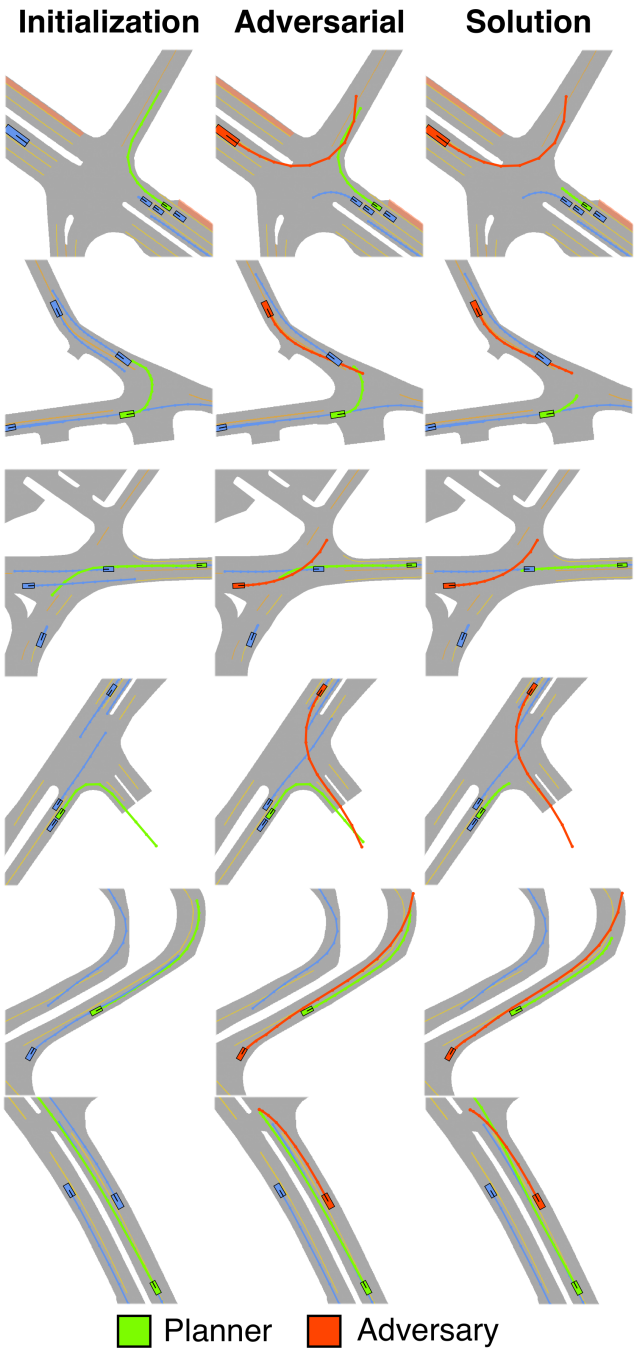

Figure 10. Additional qualitative results of STRIVE for the closed-loop *Rule-based* planner. Adversarial optimization makes large changes to the initial scenario from nuScenes, *e.g.* changing the intent of the adversary or moving stationary vehicles, to cause useful collision scenarios.

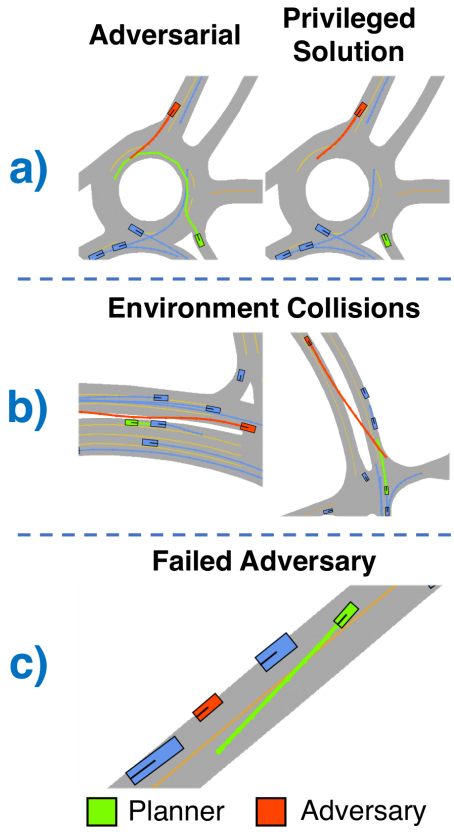

Figure 11. Example failure cases of STRIVE. (a) The solution optimization has access to privileged information, sometimes resulting in unrealistic "solutions". (b) Adversaries sometimes drive on non-drivable area to cause a collisions. (c) Attacks that require unlikely motion under the learned prior (*e.g.* a parked car pulling out) can be difficult to produce.