Learning Realistic Traffic Agents in Closed-loop Supplementary Materials

Anonymous Author(s)
Affiliation
Address
email

A Additional Results

Metrics: In order to measure the realism of our traffic models, we use a set of metrics which evaluate both the traffic models' ability to match human demonstration data in the nominal scenarios and avoid infractions in both nominal and simulated long-tail scenarios.

- *Reconstruction:* In nominal scenarios where expert demonstrations exist, we consider a set of metrics which evaluate how close a traffic model's simulation is to the real world conditioned on the same initial condition. We measure the final displacement error (**FDE**) [1], defined as the L2 distance between an agent's position in a simulated scenario vs the ground truth scenario after 5s. We also measure the along-track error (**ATE**) and cross-track error (**CTE**) of an agent's simulated position projected onto the ground truth trajectory. This decomposition disentangles speed variability and lateral deviations respectively.
- *Distributional*: While reconstruction metrics compare pairs of real and simulated logs, we can compute distributional similarity metrics as an additional method to gauge realism. We compute the Jensen-Shannon Divergence (**JSD**) [2] between histograms of scenario features to compute their distributional similarity. Features include agent kinematics like acceleration and speed, pairwise agent interactions like distance to lead vehicle, and map interactions like lateral deviation from lane centerline.
- *Infraction Rate*: Finally, we measure the rate of traffic infractions made by agents controlled by a traffic model. Similar to prior work [3], we measure percentage of agents that end up in **collision** or drive **off-road**. As this metric does not require ground truth scenarios for pairing or computing statistics, it can be used in simulated long-tail scenarios that do not have ground truth.

Comparison to state-of-the-art: In our main paper, we presented select results from our comparison to state-of-the-art traffic models on both nominal and long-tail scenarios. Here, we include additional tradeoff plots for all metrics in Figure 1. We also include a table of detailed metrics for all methods in Table 1. Building on our observations in the main paper, we see that RTR outperforms and expands the existing Pareto frontier on all metrics and scenario sets. IL methods achieve strong reconstruction/distributional realism metrics but suffer from high infraction rates, while RL methods attain the opposite. RTR achieves the best of both worlds—a testament to its ability to learn human-like driving while avoiding unrealistic traffic infractions.

Long-Tail Scenarios: In our main paper, we evaluated our approach of using procedurally generated long-tail scenarios against the alternative of mining hard scenarios from data. Here, we include additional tradeoff plots for all metrics in Figure 2, with the detailed metrics in Table 2. We see that training on both nominal and long-tail scenarios outperforms the alternatives in most cases.

Distributional Realism: In Figure 3, we include additional plots showing the histograms used to compute JSD distributional realism metrics on the nominal scenario set. We can see that RL

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

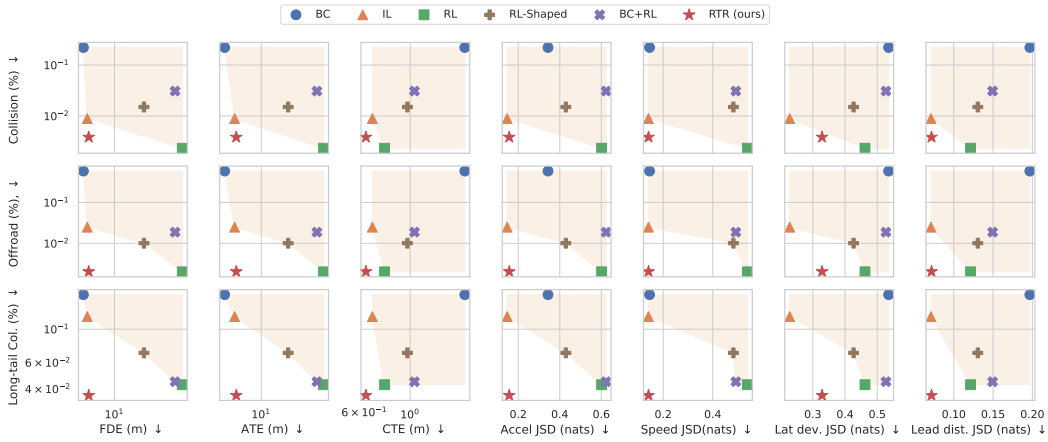

Figure 1: Additional plots comparing infraction / realism tradeoff of RTR compared to baseline models. We see that RTR outperforms and expands the existing Pareto frontier for all metrics.

Method	Infraction (%)		Reconstruction (m)			JSD (nats)				LT-Inf. (%)
	Col.	Off Rd.	FDE	ATE	CTE	Acc.	Speed	Lat. D.	Ld. D.	Col.
BC	22.13	58.68	**4.50**	**3.60**	1.84	0.34	0.54	**0.14**	0.20	17.00
IL	0.89	2.48	4.98	4.75	0.66	**0.15**	**0.23**	**0.14**	**0.07**	12.13
RL	**0.23**	**0.20**	56.92	56.91	0.75	0.60	0.46	0.54	0.12	4.26
RL-Shaped	1.50	1.01	21.29	21.17	0.97	0.43	0.43	0.48	0.13	6.95
BC+RL	3.08	1.88	47.30	47.26	1.05	0.62	0.53	0.49	0.15	4.46
RTR	0.38	**0.20**	5.16	4.97	**0.61**	0.16	0.33	**0.14**	**0.07**	**3.61**

Table 1: Detailed breakdown of metrics. Metrics on the left (resp. right) are computed on nominal scenarios (resp. long-tail scenarios). IL methods achieve strong reconstruction/distributional realism metrics but suffer from high infraction rates, while RL methods attain the opposite. RTR achieves the best of both worlds, with high reconstruction/distributional realism and low infraction rates.

methods (RL, RL-Shaped, and BC + RL) struggle to capture human-like driving, particularly in speed and acceleration JSD where the RL methods tend to brake more often than humans. BC exhibits slightly better results overall, but it has worse map interaction reasoning due to distribution shift from compounding errors. In contrast, RTR captures human-like driving significantly better, closely matching IL in distributional realism while also improving on its infraction rate as seen in other results.

Qualitative Results: We include qualitative results comparing RTR against the baselines Figures 4, 5, 6, and 7. Across fork, merge, and long-tail scenarios, we see that RTR exhibits the greatest realism of the competing methods.

B Learning

B.1 Loss Derivation

In this section, we will provide more details on the loss derivation using the Lagrangian. Recall that we begin with the following optimization problem

$$\begin{aligned} \arg\min_{\pi} \quad & D_{\text{KL}}\left(P^{\pi}(\tau) \parallel P^{E}(\tau)\right) \\ \text{s.t.} \quad & \mathbb{E}_{P^{\pi}}\left[R(\tau)\right] \geq 0 \end{aligned} \tag{1}$$

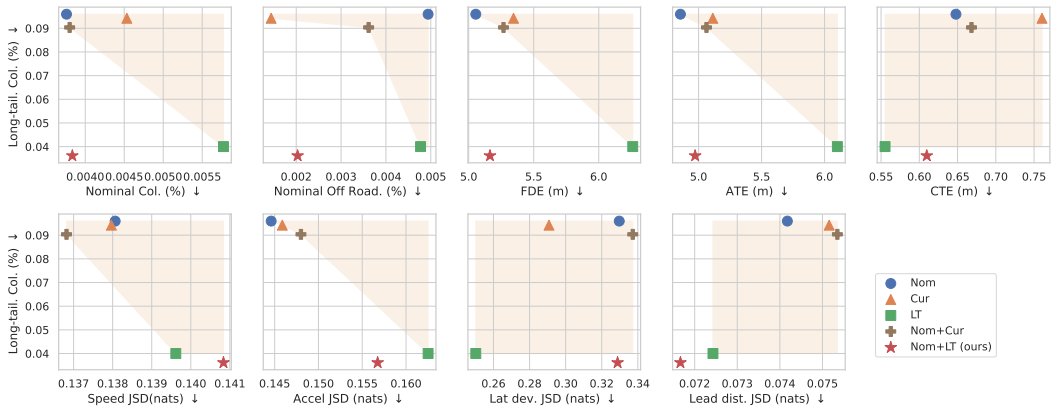

Figure 2: Additional plots showing the tradeoff between infraction rate on the long-tail set and other realism metrics on the nominal set, for models trained on different scenario sets. We see that for most metrics, training on both nominal and long-tail scenarios obtain the best tradeoff.

	Infraction (%)		Reconstruction (m)			JSD (nats)				LT-Inf. (%)
Training Scenarios	Col.	Off Rd.	FDE	ATE	CTE	Acc.	Speed	Lat. D	Ld. D.	Col.
Nominal	0.38	0.49	5.05	4.86	0.65	0.14	0.14	0.33	0.07	9.60
Curated	0.45	0.14	5.34	5.11	0.76	0.15	0.14	0.29	0.08	9.42
Long-tail	0.58	0.48	6.26	6.10	0.56	0.16	0.14	0.25	0.07	4.00
Nom. + Cur	0.38	0.30	5.27	5.06	0.67	0.15	0.14	0.34	0.08	9.04
Nom. + LT (ours)	0.38	0.20	5.16	4.97	0.61	0.16	0.14	0.33	0.07	3.61

Table 2: Detailed breakdown of realism and infraction metrics for training on different scenario sets.

We form the Lagrangian of the optimization problem

$$\mathcal{L}(\pi, \lambda) = D_{\text{KL}}\left(P^{\pi}(\tau) \parallel P^{E}(\tau)\right) + \lambda \mathbb{E}_{P^{\pi}}\left[R(\tau)\right] \tag{2}$$

$$= \mathbb{E}_{P^{\pi}}\left[\log \frac{P^{\pi}(\tau)}{P^{E}(\tau)} - \lambda R(\tau)\right] \tag{3}$$

$$= \mathbb{E}_{P^{\pi}}\left[-\log P^{E}(\tau) - \lambda R(\tau)\right] - H(\pi) \tag{4}$$

where λ is a Lagrangian multiplier and

$$H(\pi) = -\mathbb{E}_{P^{\pi}}\left[\log P^{\pi}(\tau)\right] \tag{5}$$

$$= -\mathbb{E}_{P^{\pi}}\left[\rho_0(\boldsymbol{s}_0) \sum_{t=0}^{T-1} \log \pi(\boldsymbol{a}^t | \boldsymbol{s}^t)\right] \tag{6}$$

under deterministic dynamics is the causal entropy [4]. Using the Lagragian, the optimization problem is converted to an unconstrained problem

$$\pi^{\star} = \arg\min_{\pi} \max_{\lambda} \mathcal{L}(\pi, \lambda) \tag{7}$$

Equation 7 can be optimized in a number of ways, such as iteratively solving the inner maximization over λ and outer minimization over π. We take a simplified approximate approach where we simply set $\lambda_{\text{fixed}} \geq 0$ as a hyperparameter, leading to what is ultimately a relaxed constraint or penalty method.

$$\pi^{*} \approx \arg\min_{\pi} \mathbb{E}_{P^{\pi}}\left[-\log P^{E}(\tau) - \lambda_{\text{fixed}} R(\tau)\right] - H(\pi) \tag{8}$$

The causal entropy term is included as an entropy regularization term in some learning algorithms such as PPO [5]. In practice, we found that it was not necessary to include.

B.2 Imitation Learning Loss

Recall that the imitation learning component of the loss is given as

$$\mathcal{L}^{\text{IL}} = \mathbb{E}_{\tau^E \sim D}\left[\mathbb{E}_{\tau \sim P^{\pi}(\cdot|\boldsymbol{s}_0^E)}\left[D(\tau^E, \tau)\right]\right] \tag{9}$$

$$= E_{(\boldsymbol{s}_0^E, \ldots, \boldsymbol{s}_T^E) \sim D}\left[\sum_{t=1}^{T} d(\boldsymbol{s}_t^E, \tilde{\boldsymbol{s}}_t)\right] \tag{10}$$

where

$$\tilde{\boldsymbol{a}}_t \sim \pi(\boldsymbol{a}|\tilde{\boldsymbol{s}}_t) \tag{11}$$

$$\tilde{\boldsymbol{s}}_{t+1} = \tilde{\boldsymbol{s}}_t + f(\tilde{\boldsymbol{s}}_t, \tilde{\boldsymbol{a}}_t)\mathrm{d}t \tag{12}$$

Because the dynamics function f as described in Section B.7 is differentiable, Equation 10 completely differentiable using the reparameterization trick [6] when sampling from the policy. To compute the inner expectation in Equation 9, we simply sample a single rollout. In practice, we found that directly using the mean without sampling is also sufficient.

B.3 Reward Function

Sparse reward: Recall that we use the following reward function

$$R^{(i)}(\boldsymbol{s}, a^{(i)}) = \begin{cases} -1 & \text{if an infraction occurs} \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

In our experiments, we consider collisions events and driving off-road as infractions. Collisions are computed by checking for overlap between the bounding boxes of agents. Off-road is computed by checking if an agent's bounding box still intersects with the road polygon. Note that when optimizing the reward, we apply early termination of the scenario in the event of an infraction.

Shaped reward: For the RL-Shaped baseline, use the same reward in Equation 13 with an additional term which encourages driving at the speed limit.

$$R^{(i)}_{shaped}(\boldsymbol{s}, a^{(i)}) = R^{(i)}(\boldsymbol{s}, a^{(i)}) + 0.5(C - \delta)/C \tag{14}$$

where $\delta = \text{abs}(\text{velocity} - \text{speed limit})$ and $C = 30$. We terminate the episode of $\delta \geq C$.

B.4 Reinforcement Learning Loss

We describe our factorized approach to multiagent PPO [5] in more detail. Starting off we compute a per-agent probability ratio.

$$r^{(i)} = \frac{\pi(a^{(i)}|\boldsymbol{s})}{\pi_{\text{old}}(a^{(i)}|\boldsymbol{s})}. \tag{15}$$

Our centralized value-function uses the same architecture as our policy, and computes per-agent value estimates $\hat{V}^{(i)}(\boldsymbol{s})$. Details of the architecture are found in Section B.6. The value model is trained using per-agent value targets, which are computed with per-agent rewards $R_t^{(i)} = R^{(i)}(\boldsymbol{s}_t, a_t^{(i)})$

$$\mathcal{L}^{\text{value}} = \sum_{i}^{N} (\hat{V}^{(i)} - V^{(i)})^2 \tag{16}$$

$$V^{(i)} = \sum_{t=0}^{T} \gamma^t R_t^{(i)} \tag{17}$$

We can obtain a per-agent GAE using the value model as well,

$$A^{(i)} = \text{GAE}(R_0^{(i)}, \ldots, R_{T-1}^{(i)}, \hat{V}^{(i)}(\boldsymbol{s}_T)) \tag{18}$$

The PPO policy loss is simply the sum of per-agent PPO loss,

$$\mathcal{L}^{\text{policy}} = \sum_{i=1}^{N} \min(r^{(i)}A^{(i)}, \text{clip}(r^{(i)}, 1-\epsilon, 1+\epsilon)A^{(i)}) \tag{19}$$

Finally, the overall loss is the sum of the policy and value learning loss.

$$\mathcal{L}^{\text{RL}} = \mathcal{L}^{\text{policy}} + \mathcal{L}^{\text{value}} \tag{20}$$

B.5 Input Parameterization

Agent history: Following [7], we adopt an viewpoint invariant representation of an agent's past trajectory. We encode the past trajectory as a sequence of pair-wise relative positional encodings between the past waypoints and the current pose. Each relative positional encoding consists of the sine and cosine of distance and heading difference of a pair of poses. See [7] for details.

Lane graph: To construct our lane graph representation $G = (V, E)$, We first obtain the lane graph nodes by discretizing centerlines in the high-definition (HD) map into lane segments every 10m. We use length, width, curvature, speed limit, and lane boundary type (e.g., solid, dashed) as node features. Following [8], we then connect nodes with 4 different relationships: successors, predecessors, left and right neighbors.

B.6 Model Architecture

Briefly, the RTR model architecture is composed of three main building blocks: (1) context encoders for embedding lane graph and agent history inputs; (2) interaction module for capturing scene-level interaction; and (3a) action decoder for parameterizing the per-agent policy and (3b) value decoder for the value model. Note that the policy model and the value model use the same architecture, but are trained completely separately and do not share any parameters.

History encoder: The *history encoder* consists of a 1D residual neural network (ResNet) followed by a gated recurrent unit (GRU) that extracts agent features $h_a^{(i)} = f(\boldsymbol{s}^{(i)})$ from a sliding window of past agent states $\boldsymbol{s}$. Intuitively, the 1D CNN captures local temporal patterns, and the GRU aggregates them into a global feature.

Lane graph encoder: The *lane graph encoder* is a graph convolutional network (GCN) [8] that extracts map features $h_m = g(\boldsymbol{m})$ from a given lane-graph G of map $\boldsymbol{m}$. We use hidden channel dimensions of [128, 128, 128, 128], layer normalization (LN), and max pooling aggregation.

Interaction module: To model scene-level interaction (i.e., agent-to-agent, agent-to-map, and map-to-map), we build a heterogeneous spatial graph G' by adding agent nodes to the original lane graph G. Besides the original lane graph edges, we connect agent nodes to their closest lane graph nodes. All agent nodes are also fully connected to each other. We use a *scene encoder* parameterized by a heterogeneous graph neural network (HeteroGNN) [7] to process map features and agent features into fused features,

$$\{h^{(1)}, \dots h^{(N)}\} = \text{HeteroGNN}(\{h_a^{(1)}, \dots, h_a^{(N)}\}, h_m). \tag{21}$$

These fused features are then provided as input to the decoder.

Action decoder: Finally, we pass the fused features into a 4-layer MLP with hidden dimensions [128, 128, 128] to predict agent's acceleration and steering angle distributions (parameterized as Normals).

$$(\mu^{(i)}, \sigma^{(i)}) = \text{MLP}(h^{(i)}) \tag{22}$$

$$\pi(a^{(i)}|\mathbf{s}) = \mathcal{N}(\mu^{(i)}, \sigma^{(i)}) \tag{23}$$

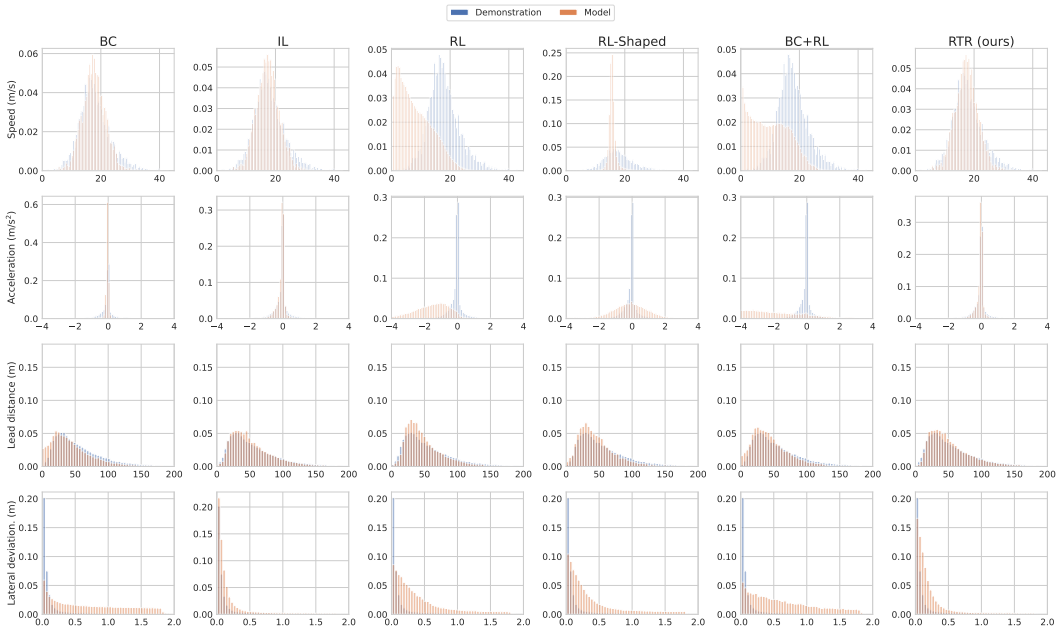

Figure 3: Histograms of scenario features for all methods used to compute JSD distributional realism metrics. We see that BC and RL methods often struggle with capturing the data distribution compared to IL and RTR. Notably, RTR closely matches IL performance in distributional realism, while greatly improving infraction rate as seen in other results.

Value decoder: For the value model, a 4-layer MLP instead regresses a single scalar value representing the value

$$\hat{V}^{(i)} = \text{MLP}_{\text{value}}\left(h^{(i)}_{\text{value}}\right) \tag{24}$$

B.7 Kinematic Bicycle Model

We use a kinematic bicycle model [9] for our environment dynamics. The bicycle model state is given as

$$s = (x, y, \theta, v) \tag{25}$$

where x, y is the position of the center of the rear axel, θ is the yaw, and v is the velocity. The bicycle model actions are

$$a = (u, \phi) \tag{26}$$

where u is the acceleration, and ϕ is the steering angle. The dynamics function $\dot{s} = f(s, a)$ is then defined as

$$\dot{x} = v\cos(\theta) \tag{27}$$

$$\dot{y} = v\sin(\theta) \tag{28}$$

$$\dot{\theta} = \frac{v}{L}\tan(\phi) \tag{29}$$

$$\dot{v} = u \tag{30}$$

where L is wheelbase length, i.e. the distance between the rear and front axel. We can use a simple finite difference approach to computing the next state

$$s_{t+1} = s_t + f(s_t, a_t)\mathrm{d}t \tag{31}$$

where $\mathrm{d}t$ is chosen to be 0.5 seconds in practice. We can apply the bicycle model to each agent individually to obtain the joint state dynamics function.

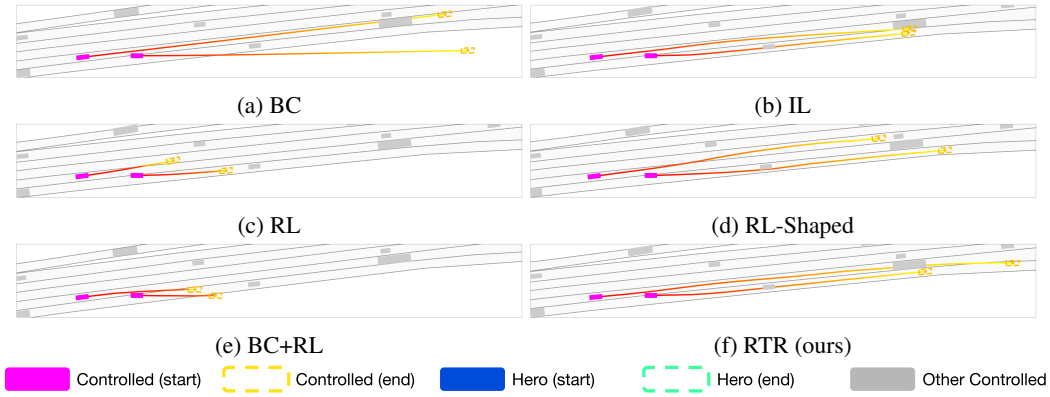

Figure 4: Qualitative results on a fork scenario. BC drives off the road, IL results in a collision while RL and BC+RL slow down. RL-Shaped drives straight and loses the interesting lane change behavior.

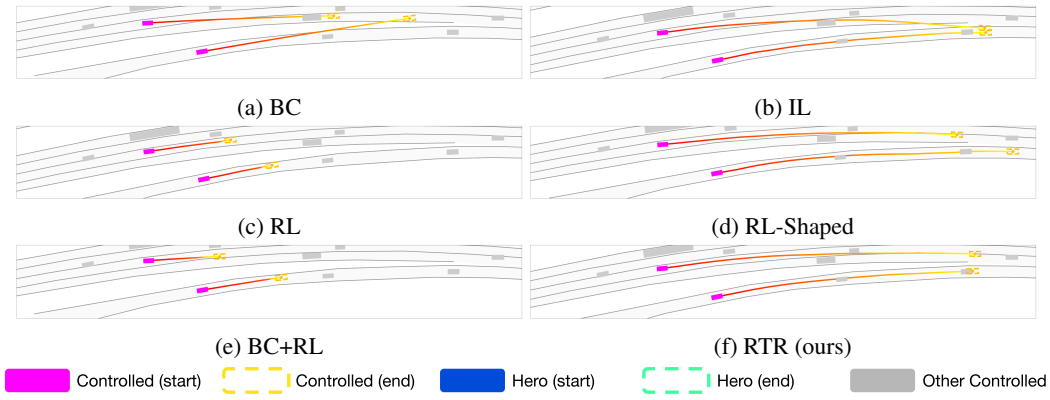

Figure 5: Qualitative results on a merge scenario. We see that RL methods slow down unrealistically. IL results in a collision while RTR maintains realism.

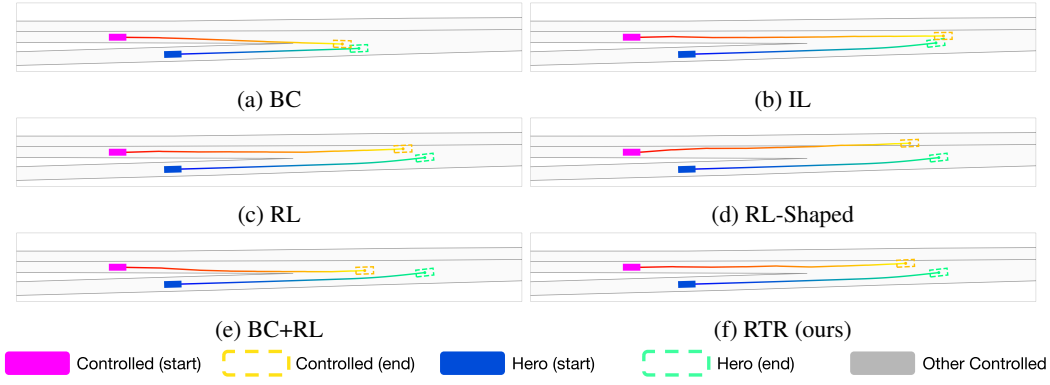

Figure 6: Qualitative results on procedurally generated merge scenario. IL and BC result in a collision. RTR maintains realism.

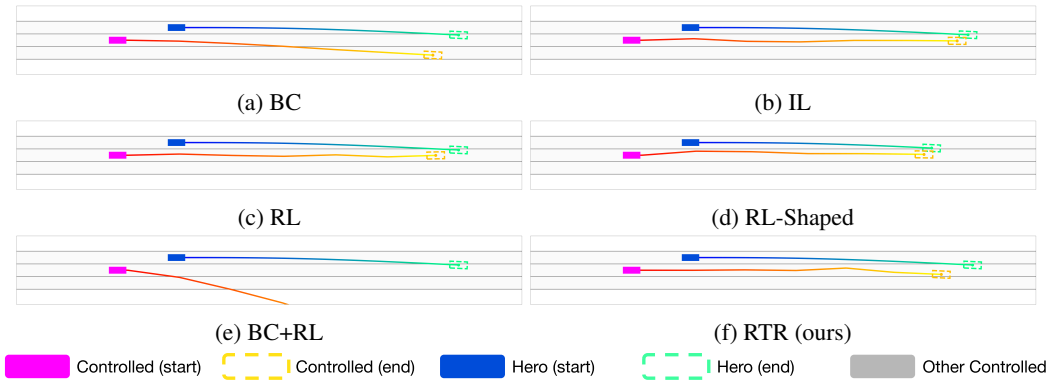

Figure 7: Qualitative results on a procedurally generated cut-in scenario. BC+RL drives off the road, while IL and RL-shaped result in a collision. RTR maintains realism.

References

[1] S. Casas, C. Gulino, S. Suo, K. Luo, R. Liao, and R. 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] M. Igl, D. Kim, A. Kuefler, P. Mougin, P. Shah, K. Shiarlis, D. Anguelov, M. Palatucci, B. White, and S. Whiteson. Symphony: Learning realistic and diverse agents for autonomous driving simulation, 2022. URL https://arxiv.org/abs/2205.03195.

[3] S. Suo, S. Regalado, S. Casas, and R. Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10400–10409, June 2021.

[4] J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[6] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[7] A. Cui, S. Casas, K. Wong, S. Suo, and R. Urtasun. Gorela: Go relative for viewpoint-invariant motion forecasting. *arXiv preprint arXiv:2211.02545*, 2022.

[8] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun. Learning lane graph representations for motion forecasting. In A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II*, volume 12347 of *Lecture Notes in Computer Science*, pages 541–556. Springer, 2020. doi:10.1007/978-3-030-58536-5_32. URL https://doi.org/10.1007/978-3-030-58536-5_32.

[9] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.