Reconstruct, Perturb, Constrain, then Attack: A Generative Adversarial Attack Paradigm and Implementation for Trajectory Prediction

Supplementary Material

Implementation Details

Design of Reconstruct Module

Trajectory Embedding of Agent Motion Feature Net. For trajectory embedding, we follow the design of Forecast-MAE [1]. The trajectory data first passes through a series of Neighborhood Attention Blocks (NAT Blocks) to extract local motion features. NAT Blocks function similarly to traditional Attention Blocks, where outputs of the Feed-Forward Network (implemented using MLP) and Neighborhood Attention Layer are connected via residual connections, with Layer Normalization applied at each step. Subsequently, the output is fed into a Feature Pyramid Network (FPN) composed of 1D-convolution layers, designed to extract global motion features.

Lane Embedding of Map Lane Feature Net. For lane embedding, we adopt the design inspired by VectorNet [2], utilizing 1D max-pooling to extract lane features. The max-pooling mechanism is implemented through a series of 1D-convolution layers, where each network layer consists of two 1D-convolution layers with Batch Normalization and ReLU activation applied between them. During forward propagation, max-pooling is performed at each network layer to generate the final feature results.

Design of Perturb Module

Self-Feature Net. We employ a Differential Attention mechanism [3] to replace the traditional Attention mechanism. This approach focuses on timesteps that have a significant impact on the attack effectiveness while reducing the contributions of less influential timesteps. This ensures the attack can be achieved with minimal perturbation. After the trajectory is embedded via an MLP, it is passed through a Differential Attention Block to obtain its self-representative feature vector. The design of the Differential Attention Block resembles that of a traditional Attention Block, where the outputs of the Differential Attention and FFN are connected via residual links, with Layer Normalization applied within the block.

Imitation Net. Since the previous step already enhances the self-representation using the Differential Attention mechanism, we employ a standard Multi-Head Attention (MHA) [4] mechanism in this step to implement cross-attention. This allows the original trajectory to reference key timesteps from the reconstructed trajectory, improving the effectiveness of the attack. The feature representation of the original trajectory is used as the query, while the reconstructed trajectory's features are used as the key and value. These are passed into a Multi-Head Attention Block, producing a cross-referenced feature representation. The design of the Multi-Head Attention Block is consistent with the previous mentioned Attention Blocks.

Design of Constrain Module

Hard Condition. As described in Sec. III-D0c, we tensorize the design of the Distance Constraining mechanism and integrate it before the input layer of the neural network of Constrainer. This setup allows for efficient enforcement of physical constraints.

Dynamics Net. After passing through the Hard Condition module, the trajectory will be input into this module. It is first embedded into a latent vector using an MLP, and then passed into two multi-layer GRUs. These GRUs separately output the speed and heading angle of the constrained trajectory.

Differentiable Dynamics Model. We adopt the differentiable dynamics model from AdvDO [5] to compute a portion of the loss. This model allows us to derive the next timestep's motion state, such as speed, heading angle, and position, from the current state using dynamic parameters. We have formulated this as an inverse dynamics equation, which computes the dynamics state at each timestep by backwardly inferring from trajectory way-points:

$$v_t = \frac{\|p_{t+1} - p_t\|}{\Delta t}, \tag{A.1}$$

$$\theta_t = \arctan\left(\frac{p_t^x}{p_t^y}\right), \tag{A.2}$$

$$a_t = \frac{v_{t+1} - v_t}{\Delta t}, \tag{A.3}$$

$$\kappa_t = \frac{d\theta_t}{v_t}, \tag{A.4}$$

where:

- v_t represents the speed at timestep t, calculated as the Euclidean distance between consecutive way-points p_t and p_{t+1} divided by the timestep Δt;
- θ_t denotes the heading angle at timestep t, computed using the arc-tangent function based on the coordinates of the position at time t (i.e., p_t^x and p_t^y);
- a_t is the acceleration at time t, defined as the change in speed between consecutive timesteps divided by Δt;
- κ_t represents the curvature at time t, calculated as the rate of change of the heading angle with respect to the speed v_t.

MODEL PARAMETERS

Reconstruct Module

Training Configuration. The specific experimental setup is detailed in Tab. A.I. We use the commonly adopted AdamW optimizer for trajectory prediction tasks and employ a cosine annealing scheduler to decay the learning rate. After extensive experimentation, we found that the best overall performance is achieved when the training epochs are set to 60.

TABLE A.I
TRAIN SETTING FOR RECONSTRUCT MODULE.

CONFIG	VALUE
SEED	42
EPOCHS	60
WARM-UP EPOCHS	10
BATCH SIZE	32
LEARNING RATE (LR)	1E-3
WEIGHT DECAY	1E-4
LR SCHEDULER	COSINE
OPTIMIZER	ADAMW

Network Parameters. Details of the network parameters are provided in Tab. A.II. Here, hidden dimension refers to the dimensionality of the feature latent vectors, and encoder depth denotes the number of NAT Block layers. Steps indicate the lengths of the past (model output) and future (model input) trajectories. Other parameters are used to configure the Attention layers.

TABLE A.II
PARAMETER SETTING FOR RECONSTRUCT MODULE.

PARAMETER	VALUE
HIDDEN DIMENSION	128
ENCODER DEPTH	4
ATTENTION HEADS	8
MLP RATIO	4
QKV BIAS	FALSE
HISTORICAL STEPS	50
FUTURE STEPS	60
DROPOUT RATE	0.2

Perturb Module

Training Configuration and Network Parameters. The training parameters are the same as those used for the Reconstruct Module, as listed in Tab. A.I. The specific network parameters are detailed in Tab. A.III.

- Hidden dimension refers to the dimensionality of the feature latent vector.
- Attention heads indicate the number of heads in both the Differential Attention and Multi-Head Attention layers.
- Encoder depth specifies the number of layers in the Differential Attention mechanism.
- Loss scale corresponds to the parameter α in Eq. 8, which adjusts the weight of the perturbation loss.

Through experiments, we found that setting the loss scale α to 0.5 balances the perturbation loss $\mathcal{L}_{ptb}$ and the target loss $\mathcal{L}_{tgt}$ in Eq. 8 effectively, with both contributing approximately equally to the total loss during training.

TABLE A.III
PARAMETER SETTING FOR PERTURB MODULE.

PARAMETER	VALUE
HIDDEN DIMENSION	128
ENCODER DEPTH	1
ATTENTION HEADS	8
DROPOUT RATE	0.2
LOSS SCALE	0.5
TRAJECTORY STEPS	50

Constrain Module

Training Configuration and Network Parameters. During training, the batch size is set to 64, the number of epochs to 30, and the warm-up epochs to 5. Other parameters are identical to those used in the Reconstruct Module and are listed in Tab. A.I. The detailed network parameters are shown in Tab. A.IV. Among these, delta time refers to the time interval in the differentiable dynamics model, which matches the sampling interval of the Argoverse 2 dataset. Decoder layers specify the number of layers in the GRU decoder. Dynamic loss rate indicates the proportion of the loss $\mathcal{L}_{dyn}$ derived from the dynamics model, corresponding to μ in Eq. 10. Through experiments, we found that a dynamic loss rate of 0.3 yields the best results.

TABLE A.IV
PARAMETER SETTING FOR CONSTRAIN MODULE.

PARAMETER	VALUE
HIDDEN DIMENSION	64
DELTA TIME	0.1
DECODER LAYERS	2
DROPOUT RATE	0.2
DYNAMIC LOSS RATE	0.3
TRAJECTORY STEPS	50

EXPERIMENT DETAILS

Main Experiments Model Setup

Models to Attack. In the experiments presented in this paper, we selected three representative models from the Argoverse 2 dataset for trajectory prediction attacks: Forecast-MAE[1] [1], QCNet[2] [6], and DeMo[3] [7]. We began by downloading the pre-trained models from their respective open-source repositories and obtaining the original dataset. The models were then validated using the processing methods specific to each model on the original dataset, producing the unperturbed validation results.

Next, we utilized the trained RPC-driven attack model to process and generate perturbed historical trajectories (from timestep 0 to 50) using the original dataset. These perturbed trajectories were written back into a copy of the original dataset, simulating a real-world attack scenario. Finally, we

[1]https://github.com/jchengai/forecast-mae
[2]https://github.com/ZikangZhou/QCNet
[3]https://github.com/fudan-zvg/DeMo

ran the attacked dataset through the target models (Forecast-MAE, QCNet, and DeMo) and validated the models again to obtain the attacked validation results.

Models to Compare. We selected the only two open-source trajectory prediction attack models for comparison: AdvTP[4] [8] and SA-Attack[5] [9]. Both AdvTP (PGD) and SA-Attack are based on the Projected Gradient Descent (PGD) method. For this comparison, we used the default settings for both models: a learning rate of 0.01, a physical constraint value of 1, a random seed count of 10, and 100 iterations per seed. For consistency with RPC's attack mode, both models were set to Multi-frame mode, with the attack length covering the entire trajectory. Additionally, we modified all the internal loops in these models to use PyTorch vectorized operations, enhancing their performance.

For AdvTP (PSO), which is based on a Particle Swarm Optimization (PSO) method, we adhered to its original settings, using 10 particles, 100 iterations, and the parameters $c_1 = 0.5$, $c_2 = 0.3$, and $w = 1.0$. The physical constraint was set to 1, and the attack length was set to cover the entire trajectory, matching the setup of RPC.

Extra Experiments Model Setup

Transferability Experiments. To evaluate the transferability of our proposed RPC-based adversarial attack, we selected representative trajectory prediction models from two popular datasets: nuScenes [10] and Argoverse 1 [11]. Specifically, we included LAformer[6] [12], Q-EANet[7] [13], and PGP[8] [14] for nuScenes, and HPNet[9] [15], DGFNet[10] [16], and HiVT[11] [17] for Argoverse 1. The evaluation protocol is similar to that in the Main Experiments section. For each model, we obtained the publicly available implementation and pre-trained weights from the corresponding official repositories. Then, we evaluated their performance on adversarially perturbed data generated by our attack pipeline.

A practical challenge in this setting is the inconsistency in trajectory and map formats across datasets. For instance, nuScenes and Argoverse 2 differ in trajectory lengths (e.g., number of past and future time steps), and all three datasets (nuScenes, Argoverse 1, and Argoverse 2) adopt different formats for encoding map context, such as lane graphs and semantic raster maps. To enable fair and consistent evaluation, we took the input/output format of each original model as the reference and transformed our adversarial inputs into the Argoverse 2 format. When necessary, we performed trajectory truncation or padding to match the expected length, and we aligned map features accordingly. All evaluations were then conducted on the converted inputs under each model's original inference pipeline.

[4]https://github.com/zqzqz/AdvTrajectoryPrediction
[5]https://github.com/eclipse-bot/SA-Attack
[6]https://github.com/mengmengliu1998/LAformer
[7]https://github.com/Jctrp/socialea
[8]https://github.com/nachiket92/PGP
[9]https://github.com/XiaolongTang23/HPNet
[10]https://github.com/XinGP/DGFNet
[11]https://github.com/ZikangZhou/HiVT

Robustness Experiments. We evaluate the robustness of our adversarial samples against various defense strategies. All evaluated defense baselines are derived from SSAT [18], which itself integrates multiple robustness techniques, including those proposed in [8] and [19]. We categorize these methods into three types: smoothing-based, augmentation, and adversarial training.

For the smoothing-based methods, we follow the setup described in [8], applying temporal smoothing to either the training or testing trajectories before feeding them into the target model. This aims to suppress high-frequency perturbations introduced by adversarial attacks. For data augmentation-based defenses, we adopt the procedure described in [18], where Gaussian noise is added to the trajectories as a form of robustness enhancement. Since the original paper does not specify exact noise levels, we perform a grid search over noise ratios ranging from 10% to 30%, and empirically find that adding noise at a 15% level yields the best trade-off. This setting is therefore adopted for our evaluations. Regarding adversarial training, we do not implement the strategy proposed in [18], due to its high implementation complexity. Instead, we adopt a naive adversarial training baseline following [20], where the model is trained jointly on clean and adversarially perturbed samples. We conduct experiments using different types of adversarial samples to investigate their influence on model robustness.

Real-word Experiments. Due to the challenges in accessing actual autonomous driving environments and related legal and ethical concerns, it is difficult to deploy our attack directly on real-world autonomous driving models. Therefore, we evaluate our approach using a mature simulation environment combined with an operational autonomous driving platform. To further ensure realism, we randomly modify a portion of the sensor data (represented here by trajectories and map information) to simulate sensor distortions. Existing studies [21]–[23] indicate that, in most cases, around 90% of the data captured by modern sensors is reliable, so we modify 10% of the map and trajectory information either by occluding parts of the data or adding random noise, thereby increasing authenticity. Our experimental scenarios benefit from nuPlan [24], and we construct 300 realistic traffic scenarios based on it. The experimental platform is built on Apollo 7.0 [25] in conjunction with LGSVL 2021.3 [26], which is analogous to the combination described in [8], albeit with updated versions.

Metrics

Trajectory Prediction Performance. In trajectory prediction, particularly for common multimodal prediction tasks in autonomous driving, three core metrics are widely used: minADE$_k$, minFDE$_k$, and MR$_k$. These metrics evaluate the quality and effectiveness of predicted trajectories and are adopted in major benchmark datasets [10], [11], [27] and their leaderboards. Given a model that generates K candidate trajectories, the evaluation is based on the best trajectory among them—-i.e., the one closest to the ground truth—-following the widely-used winner-takes-all strategy in trajectory forecasting.

The minADE$_k$ (minimum Average Displacement Error) measures the average point-wise Euclidean distance between the best predicted trajectory and the ground truth trajectory. It reflects the precision of the closest predicted trajectory across the entire time horizon. Suppose the model generates K trajectories $\hat{Y}^1, \hat{Y}^2, ..., \hat{Y}^K$ for a given agent, and the ground truth trajectory is Y. Each trajectory consists of T time steps with 2D coordinates (x_t, y_t). For the k-th predicted trajectory $\hat{Y}^k$, its ADE is defined as:

$$\text{ADE}(\hat{Y}^k, Y) = \frac{1}{T}\sum_{t=1}^{T}\sqrt{(\hat{x}_{t,k} - x_t)^2 + (\hat{y}_{t,k} - y_t)^2} \quad \text{(A.5)}$$

The minADE$_k$ is then computed as:

$$\text{minADE}_k = \min_{k\in\{1,\ldots,K\}} \text{ADE}(\hat{Y}^k, Y) \quad \text{(A.6)}$$

The minFDE$_k$ (minimum Final Displacement Error) is similar to minADE$_k$, but it focuses solely on the final position of the predicted trajectory. This metric evaluates the model's best ability to predict the correct endpoint. For the k-th predicted trajectory $\hat{Y}^k$, its FDE is computed as the Euclidean distance between the predicted and true final positions at time step T:

$$\text{FDE}(\hat{Y}^k, Y) = \sqrt{(\hat{x}_{T,k} - x_T)^2 + (\hat{y}_{T,k} - y_T)^2} \quad \text{(A.7)}$$

The minFDE$_k$ is then calculated as the minimum FDE across the K predicted trajectories:

$$\text{minFDE}_k = \min_{k\in\{1,\ldots,K\}} \text{FDE}(\hat{Y}^k, Y) \quad \text{(A.8)}$$

The MR$_k$ (Miss Rate) measures how frequently a model fails to make an acceptable prediction. If none of the K predicted trajectories meet a predefined "hit" criterion, the prediction is considered a *miss*. Specifically, in a prediction scenario, if all K trajectories have final displacement errors (FDE) greater than a distance threshold τ (commonly $\tau = 2.0$ meters, as used in [10], [11], [27]), the result is considered a miss:

$$\text{is\_miss} = \begin{cases} 1 & \text{if } \min_{k\in\{1,\ldots,K\}} \text{FDE}(\hat{Y}_k, Y) > \tau \\ 0 & \text{otherwise} \end{cases} \quad \text{(A.9)}$$

The MR$_k$ is then computed as the ratio of misses over the total number of test samples:

$$\text{MR}_k = \frac{\text{Number of Misses}}{\text{Total Number of Test Samples}} \quad \text{(A.10)}$$

Trajectory Prediction Stealthiness. We introduce four trajectory-level metrics to quantify stealthiness from a human perception and control consistency perspective: ADE, MDE, trajectory smoothness (Smo$_{\text{dist}}$) and curvature variation (Smo$_{\text{curv}}$). Both are computed based on differential properties of trajectories.

Similar to previously mentioned in Eq. A.5, ADE measures the average Euclidean distance between the adversarial trajectory and the ground truth trajectory across all historical time steps. The MDE (Maximum Displacement Error) quantifies the largest Euclidean distance between the adversarial trajectory and the ground truth at any given historical time step. It is defined as:

$$\text{MDE} = \max_{t\in\{1,\ldots,T\}} \sqrt{(\hat{x}_t - x_t)^2 + (\hat{y}_t - y_t)^2} \quad \text{(A.11)}$$

Moreover, we define Smo$_{\text{dist}}$ as the L2 norm of the root-mean-square (RMS) acceleration in both x and y directions. Given a trajectory $\mathbf{T} = (x_i, y_i)_{i=1}^{L}$ with L time steps:

$$\mathbf{v}_i = \left(\frac{x_{i+1} - x_i}{\Delta t}, \frac{y_{i+1} - y_i}{\Delta t}\right), \ i = 1, \ldots, L-1 \quad \text{(A.12)}$$

$$\mathbf{a}_i = \left(\frac{v_{i+1,x} - v_{i,x}}{\Delta t}, \frac{v_{i+1,y} - v_{i,y}}{\Delta t}\right), \ i = 1, \ldots, L-2 \quad \text{(A.13)}$$

$$\text{Smo}_{\text{dist}} = \sqrt{\frac{1}{L-2}\sum_{i=1}^{L-2} a_{i,x}^2 + \frac{1}{L-2}\sum_{i=1}^{L-2} a_{i,y}^2} \quad \text{(A.14)}$$

Smo$_{\text{curv}}$ measures the variation of trajectory curvature, which reflects abrupt changes in directional steering. High curvature fluctuations are generally unnatural and may be detected as anomalies. Given a trajectory $\mathbf{T} = (x_i, y_i)_{i=1}^{L}$ with L time steps:

$$\kappa_i = \frac{|\dot{x}_i \ddot{y}_i - \dot{y}_i \ddot{x}_i|}{(\dot{x}_i^2 + \dot{y}_i^2)^{3/2}}, \ i = 1, \ldots, L-2 \quad \text{(A.15)}$$

$$\text{Smo}_{\text{curv}} = \frac{1}{L-3}\sum_{i=1}^{L-3} |\kappa_{i+1} - \kappa_i| \quad \text{(A.16)}$$

Experiment Environment

Our experiments were conducted using Python 3.10.14, CUDA 12.1, and PyTorch 2.2.2. The hardware setup consisted of two RTX 3080 Ti GPUs and one i9-10900x CPU, with 64GB RAM. Following our parameter settings, the total GPU memory usage throughout the process ranged between 6 to 8 GB, and the entire training session lasts approximately 12 hours.

Additional Qualitative Results.

Beyond the results presented in the main text, we provide additional qualitative examples in Fig. A.1. The notations and visual elements are consistent with those in Fig. 3.

Open-sourced Code

The implementation code of RPC paradigm application on trajectory prediction can be found in https://github.com/CircuitMurderer/RPC-TP.

References

[1] J. Cheng, X. Mei, and M. Liu, "Forecast-mae: Self-supervised pre-training for motion forecasting with masked autoencoders," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 8679–8689.

[2] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 525–11 533.

[3] T. Ye, L. Dong, Y. Xia, Y. Sun, Y. Zhu, G. Huang, and F. Wei, "Differential transformer," *arXiv preprint arXiv:2410.05258*, 2024.

[4] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[5] Y. Cao, C. Xiao, A. Anandkumar, D. Xu, and M. Pavone, "Advdo: Realistic adversarial attacks for trajectory prediction," in *European Conference on Computer Vision*. Springer, 2022, pp. 36–52.

[6] Z. Zhou, J. Wang, Y.-H. Li, and Y.-K. Huang, "Query-centric trajectory prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 863–17 873.

[7] B. Zhang, N. Song, and L. Zhang, "Demo: Decoupling motion forecasting into directional intentions and dynamic states," in *Proceedings of the Neural Information Processing Systems*, 2024.

[8] Q. Zhang, S. Hu, J. Sun, Q. A. Chen, and Z. M. Mao, "On adversarial robustness of trajectory prediction for autonomous vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 15 159–15 168.

[9] H. Yin, J. Li, P. Zhen, and J. Yan, "Sa-attack: Speed-adaptive stealthy adversarial attack on trajectory prediction," in *2024 IEEE Intelligent Vehicles Symposium*, 2024, pp. 1772–1778.

[10] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[11] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan *et al.*, "Argoverse: 3d tracking and forecasting with rich maps," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8748–8757.

[12] M. Liu, H. Cheng, L. Chen, H. Broszio, J. Li, R. Zhao, M. Sester, and M. Y. Yang, "Laformer: Trajectory prediction for autonomous driving with lane-aware scene constraints," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 2039–2049.

[13] J. Chen, Z. Wang, J. Wang, and B. Cai, "Q-eanet: Implicit social modeling for trajectory prediction via experience-anchored queries," *IET Intelligent Transport Systems*, vol. 18, no. 6, pp. 1004–1015, 2024.

[14] N. Deo, E. Wolff, and O. Beijbom, "Multimodal trajectory prediction conditioned on lane-graph traversals," in *Conference on Robot Learning*. PMLR, 2022, pp. 203–212.

[15] X. Tang, M. Kan, S. Shan, Z. Ji, J. Bai, and X. Chen, "Hpnet: Dynamic trajectory forecasting with historical prediction attention," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 15 261–15 270.

[16] G. Xin, D. Chu, L. Lu, Z. Deng, Y. Lu, and X. Wu, "Multi-agent trajectory prediction with difficulty-guided feature enhancement network," *IEEE Robotics and Automation Letters*, 2025.

[17] Z. Zhou, L. Ye, J. Wang, K. Wu, and K. Lu, "Hivt: Hierarchical vector transformer for multi-agent motion prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8823–8833.

[18] R. Jiao, X. Liu, T. Sato, Q. A. Chen, and Q. Zhu, "Semi-supervised semantics-guided adversarial training for robust trajectory prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 8207–8217.

[19] S.-A. Rebuffi, S. Gowal, D. A. Calian, F. Stimberg, O. Wiles, and T. Mann, "Fixing data augmentation to improve adversarial robustness," *arXiv preprint arXiv:2103.01946*, 2021.

[20] A. Madry, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[21] M. Yang, K. Jiang, J. Wen, L. Peng, Y. Yang, H. Wang, M. Yang, X. Jiao, and D. Yang, "Real-time evaluation of perception uncertainty and validity verification of autonomous driving," *Sensors*, vol. 23, no. 5, p. 2867, 2023.

[22] Y. Zhang, A. Carballo, H. Yang, and K. Takeda, "Perception and sensing for autonomous vehicles under adverse weather conditions: A survey," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 196, pp. 146–177, 2023.

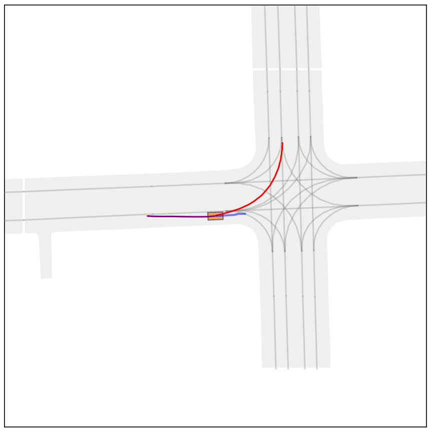
[23] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, p. 2140, 2021.

[24] H. Caesar, J. Kabzan, K. S. Tan, W. K. Fong, E. Wolff, A. Lang, L. Fletcher, O. Beijbom, and S. Omari, "nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles," *arXiv preprint arXiv:2106.11810*, 2021.

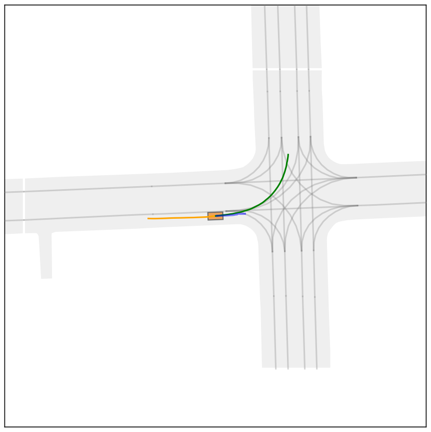
[25] Baidu Apollo Team, "Apollo auto: An open autonomous driving platform," https://apollo.auto/, 2017.

[26] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," in *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*. IEEE, 2020, pp. 1–6.

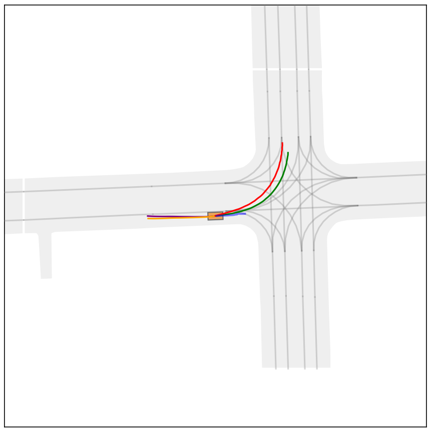
[27] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, and J. Hays, "Argoverse 2: Next generation datasets for self-driving perception and forecasting," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.



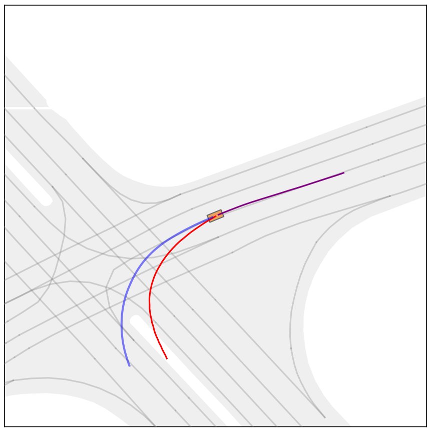
(a) Y_{true} (blue), X_{org} (purple), Y_{des} (red) in additional scenario 1.



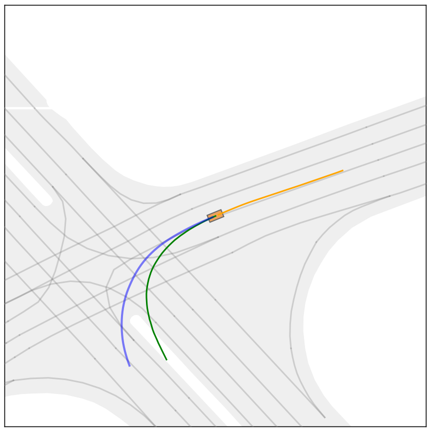
(b) Y_{true} (blue), Y_{pred} (green), X_{adv} (yellow) in additional scenario 1.



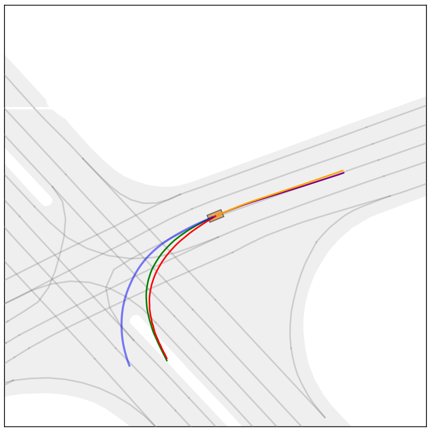
(c) All trajectories (original and attacked) in additional scenario 1.



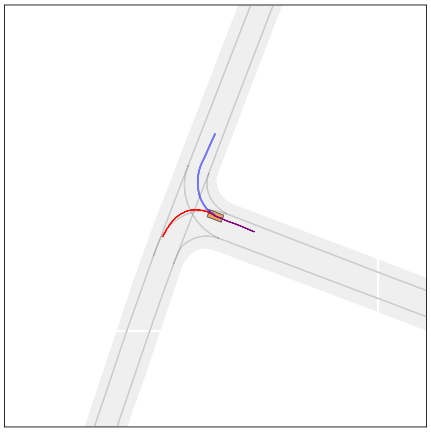
(d) Y_{true} (blue), X_{org} (purple), Y_{des} (red) in in additional scenario 2.



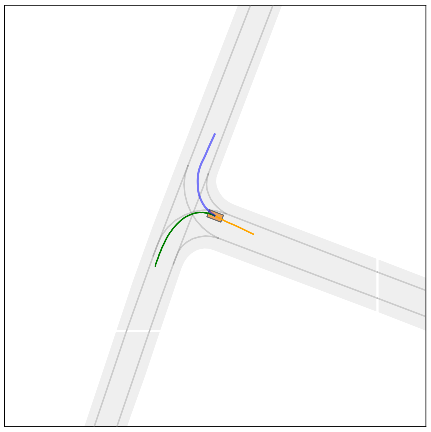
(e) Y_{true} (blue), Y_{pred} (green), X_{adv} (yellow) in additional scenario 2.



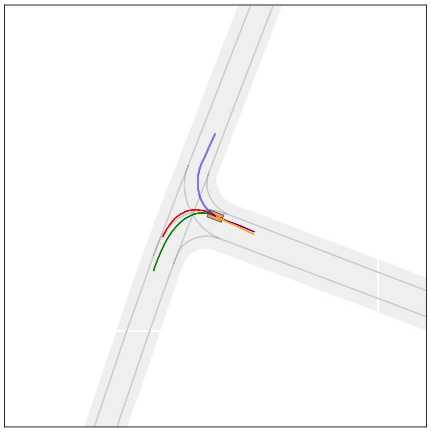
(f) All trajectories (original and attacked) in additional scenario 2.



(g) Y_{true} (blue), X_{org} (purple), Y_{des} (red) in in additional scenario 3.



(h) Y_{true} (blue), Y_{pred} (green), X_{adv} (yellow) in additional scenario 3.



(i) All trajectories (original and attacked) in additional scenario 3.

Fig. A.1. Additional prediction scenarios under our method's attack. Subfigures (a), (b), and (c) correspond to additional scenario 1; (d), (e), and (f) represent additional scenario 2; (g), (h), and (i) represent additional scenario 3. Y_{true} denotes the true future trajectory, Y_{pred} is the actual predicted trajectory by the model, Y_{des} is the attacker's desired future trajectory, X_{org} represents the original historical trajectory, and X_{adv} is the perturbed attack trajectory generated by our method.