SparseWorld: Enhancing End-to-End Autonomous Driving via World Models with Sparse Scene Representation

Supplementary Material

Anonymous Authors

I. Additional Method Details

A. Pseudocode of SparseWorld

Algorithm 1 presents the complete procedure of SparseWorld, which operates in three sequential stages as illustrated in Fig. 2: (i) Instance-Aware Driving Baseline, (ii) Future Forecasting with Sparse Dreamer, and (iii) Motion Planning Refinement. The algorithm takes historical sensor observations as input, and outputs refined ego and agent trajectories with enhanced safety and accuracy.

Algorithm 1: SparseWorld: End-to-End Autonomous Driving with Sparse World Model

Input: Historical observations $O_{-h:0}$
Output: Refined ego and agent trajectory T^*
Stage 1: Baseline Processing ;
 # Predict historical instances and base trajectory
$I_{-h:0} \leftarrow \text{InstancePerception}(O_{-h:0})$
$T_{\text{base}} \leftarrow \text{MotionPlanner}(I_{-h:0})$
 # Derive action conditions from base trajectory
$C_0 \leftarrow \text{ActionCondition}(T_{\text{base}})$
Stage 2: Future Forecasting ;
Initialize Instance Memory Queue with $I_{-h:0}$
 # Autoregressive future forecasting loop
for $t = 1$ **to** f **do**
  # Predict next frame instances
  $I_t \leftarrow \text{SparseDreamer}(I_{t-m-1:t-1}, C_{t-1})$
  # Update action conditions for next timestep
  $C_t \leftarrow \text{FutureMotionPlanner}(I_t)$
  # Update queue with newly predicted instances
  Update Instance Memory Queue with I_t
Stage 3: Motion Planning Refinement ;
 # Refine motion planning using future instances
$T_{\text{refined}} \leftarrow \text{MotionPlanningRefinement}(I_0, I_{1:f})$
 # Get trajectory from future motion planner
$T_{\text{future}} \leftarrow \text{FutureMotionPlanner}(I_1)$
 # Select safest trajectory from candidates
$T^* \leftarrow$
AdaptiveTrajectorySelection$(T_{\text{base}}, T_{\text{refined}}, T_{\text{future}})$
return T^*

B. Details for Planning Refinement

As mentioned in the main text, safety-critical loss $\mathcal{L}_{scl}$ is computed based on the safety-critical adjustment vector $v^{adj} \in \mathbb{R}^{T\times 2}$. And the calculation process of $v^{adj} \in \mathbb{R}^{T\times 2}$ is denoted as:

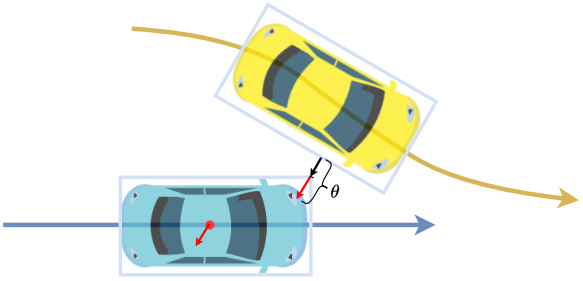

Fig. 1: **Illustration of Adjustment Vector Computation**, The blue car and the yellow car represent the ego vehicle b_i^{ego} and another vehicle $b_{i,j}^{agent}$ at the i-th future step, respectively. The black arrow denotes the minimum distance vector (MDV) between the bounding boxes, while the red arrow indicates the adjustment vector $v_{i,j}^{adj}$ required by the ego vehicle at the i-th future step contributed by $b_{i,j}^{agent}$.

$$v^{adj} = \text{SAV}(\tau, a^{\text{agent}}, \tau^{agent}, \theta) \tag{1}$$

where $\tau \in \mathbb{R}^{T\times 2}$ denotes the ego trajectory in the next T steps, $a^{agent} \in \mathbb{R}^{N_a\times 11}$ and $\tau^{agent} \in \mathbb{R}^{N_a\times T\times 2}$ represents the N_a surrounding agent anchors and their future trajectories, θ means the safety distance threshold. Firstly, we calculate the bounding boxes of the surrounding agents and the ego-vehicle based on their geometric attributes and future positions, denoted as $B^{agent} = \{b_{i,j}^{agent} \in \mathbb{R}^4 | i = 0, \ldots, T, j = 0, \ldots, N_a\}$ and $B^{ego} = \{b_i^{ego} \in \mathbb{R}^4 | i = 0, \ldots, T\}$, respectively. Then, for the j-th agent on the i-th future step $b_{i,j}^{agent}$, we calculate the minimum distance vector $v_{i,j} \in \mathbb{R}^2$ and the corresponding safety vector $s_{i,j} \in \mathbb{R}^2$ between $b_{i,j}^{agent}$ and b_i^{ego} as follows:

$$v_{i,j} = \text{MDV}(b_i^{ego}, b_{i,j}^{agent}) \tag{2}$$

$$s_{i,j} = \theta \times \hat{v}_{i,j} \tag{3}$$

Fig. 1 provides a visualized example of the computational process. The function MDV computes the minimum distance vector between two rectangles, while the safety vector $s_{i,j}$ is calculated as the product of a safety threshold and a unit direction vector of $v_{i,j}$. The adjustment vector contributed by $b_{i,j}^{agent}$ on the i-th future step is calculated as:

$$v_{i,j}^{adj} = \begin{cases} s_{i,j} - v_{i,j}, \text{if } ||v_{i,j}||_2 < \theta \\ (0,0), \qquad \text{if } ||v_{i,j}||_2 >= \theta \end{cases} \tag{4}$$

The adjustment vector at the i-th step $v_i^{adj} \in \mathbb{R}^2$ is computed as the average of contributions from all surrounding agents. By concatenating the adjustment vectors across all steps, the overall adjustment vector can be obtained.

$$v_i^{adj} = \frac{1}{N_a} \sum_{j=1}^{N_a} v_{i,j}^{adj} \tag{5}$$

$$v^{adj} = \text{CONCAT}(\{v_i^{adj} | i = \{1, \ldots, T\}\}) \tag{6}$$

In our work, θ is set to 0.5m during both training and inference.

C. Loss Function

In principle, we align the instance detection and motion planning loss formulation with those used in the baseline method. For clarity, we use SparseWorld-S as a representative example to elaborate on the details.

a) Instance Detection Loss.: The detection loss is formulated as a linear combination of the Focal Loss $\mathcal{L}_{d_{cls}}$ for classification and the L1 Loss $\mathcal{L}_{d_{reg}}$ for box regression:

$$\mathcal{L}_{det} = \lambda_{d_{cls}} \mathcal{L}_{d_{cls}} + \lambda_{d_{reg}} \mathcal{L}_{d_{reg}} \tag{7}$$

where $\lambda_{d_{cls}}$ and $\lambda_{d_{reg}}$ are set to 2 and 0.25, respectively. The mapping loss is similar to the detection loss, we define it as the following equation:

$$\mathcal{L}_{map} = \lambda_{m_{cls}} \mathcal{L}_{m_{cls}} + \lambda_{m_{reg}} \mathcal{L}_{m_{reg}} \tag{8}$$

where $\lambda_{m_{cls}}$ and $\lambda_{m_{reg}}$ are set to 1 and 10, respectively.

b) Motion and Planning Loss.: We compute the average displacement error (ADE) between the multi-model outputs and the ground truth trajectory. The trajectory yielding the minimum ADE is designated as the positive sample, while other trajectories are treated as negative samples. Furthermore, for the planning component, the ego status is also predicted. For the training process, Focal Loss is utilized for classification, and L1 Loss is applied for regression:

$$\begin{aligned} \mathcal{L}_{mp} = & \lambda_{motion_{cls}} \mathcal{L}_{motion_{cls}} + \lambda_{motion_{reg}} \mathcal{L}_{motion_{reg}} \\ & + \lambda_{plan_{cls}} \mathcal{L}_{plan_{cls}} + \lambda_{plan_{reg}} \mathcal{L}_{plan_{reg}} \\ & + \lambda_{plan_{status}} \mathcal{L}_{plan_{status}} \end{aligned} \tag{9}$$

where $\lambda_{motion_{cls}}$ and $\lambda_{motion_{reg}}$ are set to 0.2 and 0.2, $\lambda_{plan_{cls}}$, $\lambda_{plan_{reg}}$ and $\lambda_{plan_{status}}$ are set to 0.5, 1.0 and 1.0, respectively.

c) Future Forecasting Loss Function.: We optimize SparseWorld with future instances forecasting, motion prediction and trajectory planning in an end-to-end manner by leveraging the following loss functions:

$$\mathcal{L}_{ins} = \frac{1}{f} \sum_{t=1}^{f} (\mathcal{L}_{det} + \mathcal{L}_{map} + \mathcal{L}_{mp}) \tag{10}$$

where f denotes the number of future frames, $\mathcal{L}_{det}$ represents the detection loss, $\mathcal{L}_{map}$ refers the mapping loss and $\mathcal{L}_{mp}$ indicates the motion and planning loss.

d) Motion Planning Refinement Loss Function.: The loss function used for motion planning refinement is essentially the same as the motion and planning loss, with the key difference being the inclusion of the safety-critical loss for trajectory planning refinement:

$$\mathcal{L}_{refine} = \mathcal{L}_{mp} + \lambda_{scl} \mathcal{L}_{scl} \tag{11}$$

where λ_{scl} is set to 0.1.

II. Experimental Settings

A. Dataset

We initially conduct our experiments for future agent and map layout forecasting, as well as for motion planning refinement, on the nuScenes dataset. Furthermore, to validate the effectiveness of our planning refinement, we perform closed-loop experiments on Bench2Drive. Sparseworld takes the corresponding images from two historical frames and the current frame as input to predict agent states, map layouts, motion predictions, and the planning trajectory for four future timestamps. The results from these future timestamps are then used to refine the motion planning output of the current frame. To ensure the availability of corresponding past frames for input and future frames for evaluation during training, we trimmed the nuScenes dataset and the Bench2Drive open-loop training set. For the purpose of fair comparison, during the nuScenes planning evaluation, any test sequences absent from our trimmed set but present in the official complete sequences are populated with the test results from the baseline method. In contrast, no modifications to the evaluation process are necessary for the closed-loop planning test, as the evaluation is always conducted on continuous temporal sequences.

B. Implementation Details

We primarily report the implementation details of SparseWorld-S, as the experimental setup of SparseWorld-V follows an almost identical configuration. The training process for SparseWorld-S, building upon a model pre-trained with SparseDrive, is divided into two stages. In Stage 1, the modules corresponding to the future forecasting process are trained. In Stage 2, the future forecasting modules from Stage 1 are first loaded and their weights are frozen. The training then focuses on the motion planning refinement modules. During Stage 1, we predict and supervise four future frames of instances. In contrast, during stage2, we roll out only two future frames of instances. We use ResNet50 as the backbone network. The input image size is 256×704 for the nuScenes dataset and 384×704 for the Bench2Drive dataset. For future agents, the perception range is a circle with a 55m radius. For the future map layout, the perception range covers 60m longitudinally and 30m laterally. For both motion and planning, the number of modes is set to 6, with prediction horizons of 12 and 6 timesteps, respectively. We utilize the AdamW optimizer and a Cosine Annealing scheduler for model training. On the nuScenes dataset, we train Stage 1 for 100 epochs with a batch size of 4 and a

TABLE I: **Future instance forecasting results on the nuScenes validation dataset.** Avg. denotes the average performance of that in 0.5s, 1.0s, 1.5s, and 2.0s. We use bold numbers to denote the future forecasting results.

	Online Mapping/mAP ↑						3D Object Detection/NDS ↑					
Method	0s	0.5s	1.0s	1.5s	2.0s	Avg.	0s	0.5s	1.0s	1.5s	2.0s	Avg.
SparseWorld-S	56.66	**56.48**	**54.20**	**51.83**	**48.96**	**52.87**	0.523	**0.501**	**0.477**	**0.454**	**0.433**	**0.467**
SparseWorld-V	39.83	**37.51**	**35.68**	**33.51**	**31.33**	**34.51**	0.389	**0.378**	**0.370**	**0.356**	**0.342**	**0.362**

TABLE II: **Future instance forecasting results for online mapping on the nuScenes validation set under different noise levels.** σ denotes the sparsity coefficient used to scale the noise variance.

		Online Mapping/mAP ↑					
Method	σ	0s	0.5s	1.0s	1.5s	2.0s	Avg.
SparseWorld-S	0.0	**56.66 (100%)**	56.48	54.20	51.83	48.96	**52.87 (100%)**
SparseWorld-S	0.3	**56.18 (99.2%)**	56.08	53.74	51.15	48.35	**52.33 (99.0%)**
SparseWorld-S	0.9	**47.51 (83.8%)**	47.27	44.98	42.78	40.15	**43.80 (82.8%)**

TABLE III: **Future instance forecasting results for 3D object detection on the nuScenes validation set under different noise levels.** σ denotes the sparsity coefficient used to scale the noise variance.

		3D Object Detection/NDS ↑					
Method	σ	0s	0.5s	1.0s	1.5s	2.0s	Avg.
SparseWorld-S	0.0	**0.523 (100%)**	0.501	0.477	0.454	0.433	**0.467 (100%)**
SparseWorld-S	0.1	**0.517 (98.9%)**	0.495	0.471	0.449	0.428	**0.461 (98.7%)**
SparseWorld-S	0.3	**0.479 (91.6%)**	0.459	0.439	0.420	0.402	**0.430 (93.4%)**

learning rate of 2×10^{-4}. Stage 2 is then trained for 20 epochs, with all other hyperparameters remaining the same as in Stage 1. On the Bench2Drive dataset, Stage 1 is trained for 10 epochs with a batch size of 4 and a learning rate of 2×10^{-4}, while Stage 2 is trained for 4 epochs using the same hyperparameters.All training experiments were conducted on 8 NVIDIA RTX 4090D 24GB GPUs, while the closed-loop evaluations were run on 8 NVIDIA RTX 3090 24GB GPUs.

III. Additional Experiments

A. Agents and Map Layout Forecasting Results.

In the main paper, our analysis primarily centers on verifying whether the world model effectively captures the dynamic evolution of the scene. In this section, we extend the investigation by examining the performance of SparseDreamer across different baselines and under varying noise perturbations.

a) Results on Different Baselines.: Table I illustrates the online mapping and 3D detection results in the future. The metrics at 0s correspond to the perception performance of the corresponding baselines, while results from 0.5s to 2.0s reflect the forecasting capability of SparseWorld. Notably, SparseWorld-V retains **87.9%** of the baseline performance in 3D detection and **78.7%** in online mapping at 2.0s, whereas SparseWorld-S achieves **82.8%** in 3D detection and **86.4%** in online mapping. These results demonstrate that both variants effectively perform future prediction. SparseWorld-S exhibits superior fidelity in online mapping, while SparseWorld-V shows stronger performance in 3D object detection. This indicates that the SparseDreamer module is versatile and compatible with both BEV-based and BEV-free perception frameworks, delivering robust performance across different baseline architectures.

b) Noise Robustness Analysis.: We further investigate the sensitivity of the SparseDreamer module to noisy inputs. Since the module takes the current-frame instance features together with anchors as input, and the anchors encode critical geometric information, we inject Gaussian noise into all anchor attributes to examine its robustness. For 3D object detection, noise is added to the position, size, and orientation of each anchor; for online mapping, noise is applied to the 3D locations of map points. The noise follows a zero-mean normal distribution whose variance is scaled by a sparsity coefficient σ. Table II and Table III summarize the online mapping and 3D object detection future forecasting results under increasing noise levels. As the noise magnitude grows, the perception quality of the baseline system degrades, and the future instance forecasting accuracy of SparseDreamer also exhibits a corresponding decline over the 2-second horizon. Importantly, the average degradation ratio closely aligns with that of the baseline, demonstrating that SparseDreamer maintains strong robustness to anchor noise.

B. Open-loop Planning Results.

The primary advantage of incorporating a world model into an autonomous driving system lies in its ability to enhance trajectory safety. By providing explicit foresight into future scene evolution, the world model allows the planner to anticipate and proactively avoid potential hazards. In the main paper, we have shown that integrating SparseWorld yields a substantial reduction in collision rate. Here, we further investigate how SparseWorld affects additional planning metrics—specifically Lane-Keeping and Comfortness—both before and after integration. For each trajectory sample, the lane-keeping score is assigned a value of 1 if the model's lane-change decisions match those of the expert trajectory; otherwise, it is set to 0. Similarly, the comfortness score

Fig. 2: **Qualitative results of motion prediction refinement on the nuScenes validation set.** We visualize the most confident trajectory among the six predicted ones. The baseline result from SparseDrive is marked in orange, the refined result from SparseWorld-S is shown in purple, and the ground truth is indicated in black.

is set to 1 if the trajectory satisfies the comfort rules—i.e., it contains no abrupt acceleration, deceleration, or steering maneuvers—and 0 otherwise. We report the average scores computed over all trajectories in the validation set. As shown in Table IV, SparseWorld introduces consistent improvements in lane-keeping and comfortness. These gains indicate that the enhanced safety does not arise from overly cautious behaviors such as unnecessary lane changes or abrupt braking. Rather, the model leverages predictive future context to produce safer, smoother, and more competent trajectories, demonstrating an improved driving style instead of a shift toward conservative or risk-averse planning.

TABLE IV: **Additional Open-loop Results of E2E Methods on nuScenes validation dataset.** LK indicates Lane-Keeping, Comf represents Comfortness.

Method	Avg.L2↓	Col.(%)↑	LK(%)↑	Comf(%)↑
VAD-Tiny	0.78	0.38	95.39	98.37
SparseWorld-V	0.59	0.24	96.21	98.88
SparseDrive-S	0.61	0.08	96.40	98.31
SparseWorld-S	0.65	0.05	96.73	98.54

IV. Additional Visualizations

A. *Future forecasting*

In Fig. 3, we present qualitative results of our future predictions under three distinct weather environments. For each environment, the top row displays the input, consisting of historical agent and map layouts derived from perception, while the bottom row shows the output from SparseWorld, which includes the predicted agent and map layouts for the subsequent two seconds. The first case, under sunny conditions, illustrates the ego-vehicle entering an intersection. This demonstrates that given the perception results from the past three frames, SparseWorld can accurately predict the vehicle entering the intersection by interpreting the evolving map and the vehicle's motion, achieving the prediction with high fidelity. The second case, during rainy weather, shows the ego-vehicle making a left turn at an intersection. Visibility is limited under the condition. Consequently, historical perception may fail to correctly identify all distant vehicles in the oncoming lane, which in turn affects the prediction for future

frames. However, the identification of nearby vehicles and the map remains accurate. SparseWorld is still able to output a future scene without significant distortion, thereby not adversely affecting the ego-vehicle's final planning outcome. The third example presents a nighttime scene where the ego-vehicle is driving in a lane with heavy traffic. As shown, the motion predictions of Sparseworld for multiple agents in future frames remain largely accurate, and the predicted scene exhibits no noticeable distortion. As a result, the ego-vehicle's final plan is not significantly impacted. These results indicate that SparseWorld can accurately model the future states of both dynamic and static objects, thereby understanding how the driving scene evolves over time.

B. Motion Planning Refinement

a) Motion Prediction Refinement: Fig. 2 illustrates the effectiveness of our refinement on the motion predictor. Both our method and the baseline predict six trajectories, with the one of highest confidence visualized. In the illustrated cases, the baseline method makes incorrect estimations of the vehicle's displacement or rotation, failing to reflect the actual future motion. In comparison, when such errors occur in the baseline method, the motion predictor refined by future instance prediction continues to perform reliably, yielding more accurate results and validating the effectiveness of our approach.

b) Adaptive Trajectory Selection: Fig. 4 illustrates the decision-making process in the adaptive trajectory selection module. The left column of the figure presents ground-truth examples where, despite successful detection of surrounding agents by the perception module, the baseline planning trajectories still fail to avoid potential collisions. As shown in the center column, beyond the base trajectory, SparseWorld retains two additional candidate trajectories: the refined trajectory with safety-critical loss applied and the future trajectory generated from future forecasting. The safety-critical loss is computed for each trajectory candidate to evaluate its associated risk, and the trajectory with the lowest risk is selected for execution. As demonstrated in the right column, although the base trajectory remains hazardous in these cases, the future trajectory is significantly safer, helping the system successfully avoids potential collisions. This safety-critical adaptive trajectory selection module greatly enhances the decision-making ability, leading to improved overall system robustness and reliability in complex driving scenarios.

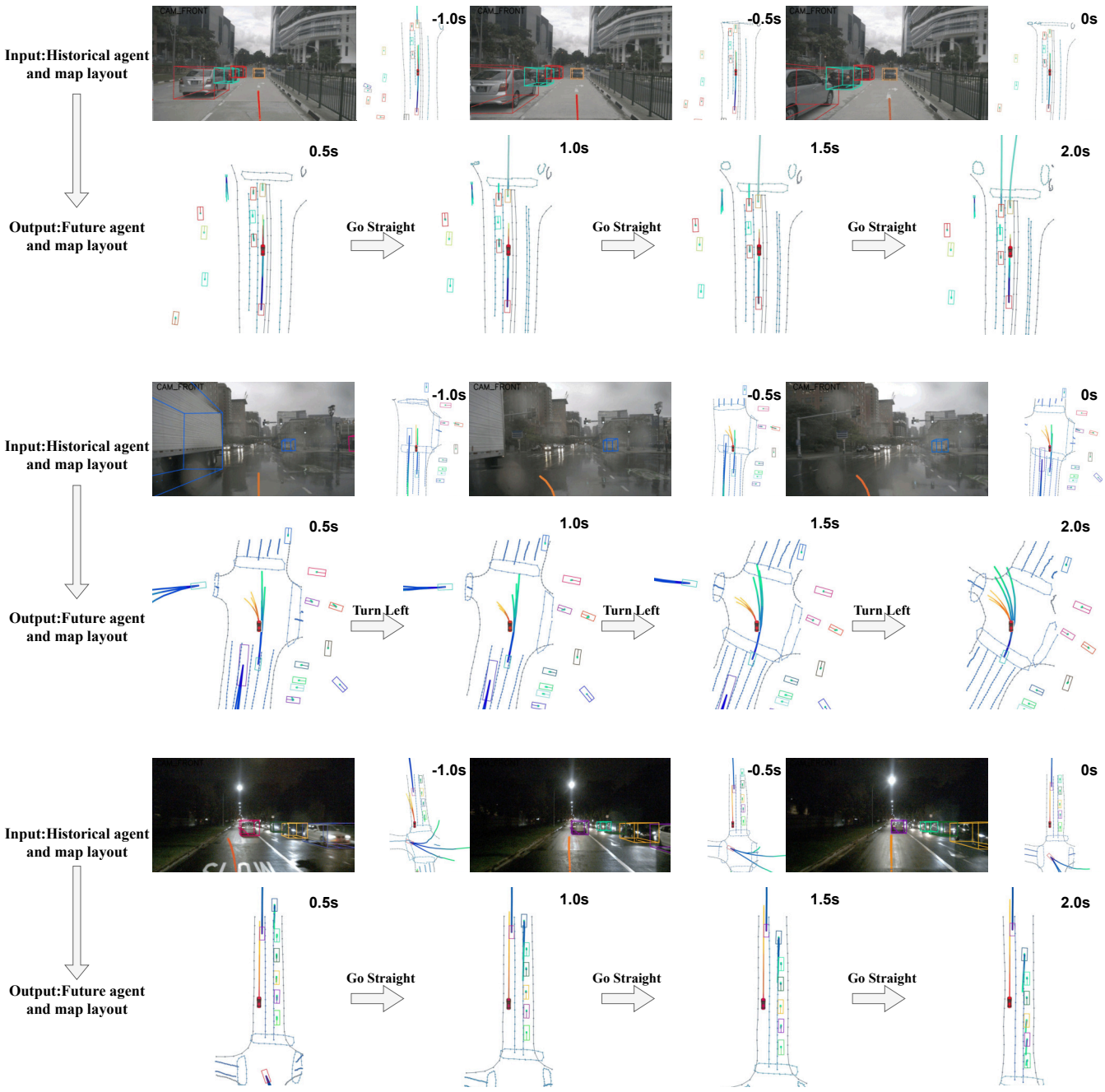

Fig. 3: **Qualitative results on the nuScenes validation set are presented under various weather and command conditions.** The input consists of agent and map layouts derived from the perception results of two historical frames and the current frame. These perception results are based on an input of six surround-view images, though only the front view is visualized here. The output includes predictions for agent and map layouts over the next two seconds. The conditions displayed cover a range of weather, including sunny, rainy, and nighttime scenarios, as well as driving actions such as proceeding straight, turning left, and the situations of entering and exiting intersections.

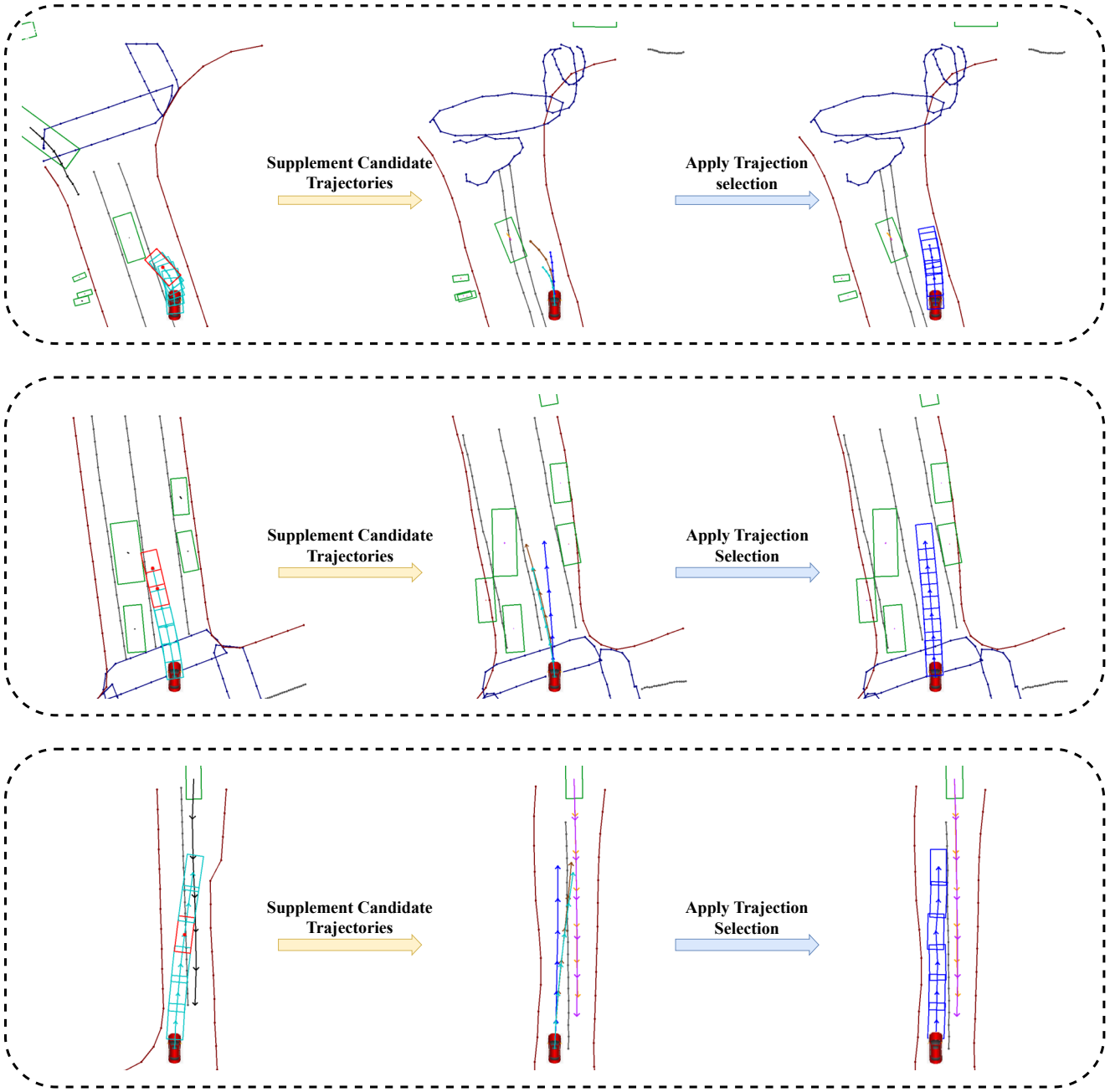

Fig. 4: **Qualitative results of trajectory planning refinement on the nuScenes validation set.** The cyan trajectory denotes the the base trajectory from the baseline, while the brown and blue trajectories depict the two candidates—one optimized via the safety-critical loss and the other generated through future forecasting—with the blue trajectory being regarded as the safer of the two. Left: Cases that base trajectories result in collisions. Center: Visualizations for all candidate trajectories. Right: The final selected trajectory. The agent and map layouts in the left column are based on ground truth, while those in the center and right columns are derived from model predictions. When a potential collision is detected, SparseWorld selects the safest trajectory among all candidates for execution.