Traffic Scene Generation from Natural Language Description for Autonomous Vehicles with Large Language Model

Supplementary Material

A. Environment Setup

In this section, we provide a detailed list of the supported agent types, agent actions, agent behaviors, relative positions to ego, road types, signals, objects, and weather conditions with our pipeline. "()" indicates the corresponding variable used in the following context.

Agent Type (`AGENT_TYPE`)**:** An agent can be one of *ambulance*, *police*, *firetruck*, *bus*, *truck*, *motorcycle*, *car*, *pedestrian*, or *cyclist*.

Agent Action (`ACTION`)**:** Each agent action can be one of *turn left*, *turn right*, *go straight*, *change lane to left*, *change lane to right*, *stop*, *block the ego*, *cross the road*, or *on the sidewalk*.

Agent Behavior (`AGENT_BEHAVIOR`) Each agent behavior can be one of *cautious*, *normal*, and *aggressive*, categorized by the driving speed and the safe distance.

Relative Position to Ego (`RELATIVE_POSITION`)**:** Supportive relative position to the ego car include *front*, *back*, *left*, *right*, *front left*, *front right*, *back left*, *back right*, *road of left turn*, *road of right turn*.

Road Type (`ROAD_TYPE`)**:** We support three types of road segments for spawning: *driving*, *sidewalk*, and *shoulder*.

Signal (`SIGNAL`)**:** Supported signal types include *traffic light*, *stop sign*, and *yield sign*.

Object (`OBJECT`)**:** Supported objects include *speed sign* (*e.g.*, speed sign of 60), *simple crosswalk*, *ladder crosswalk*, *continental crosswalk*, *dashed single white crosswalk*, *solid single white crosswalk*, *stop line*, and *stop sign on road*.

Weather (`WEATHER`)**:** Weather conditions can be described by an "adjective" including *clear*, *cloudy*, *hard rain*, *mid rain*, *soft rain*, *wet cloudy*, and *wet*, combined with a "time" including *night*, *noon*, and *sunset*.

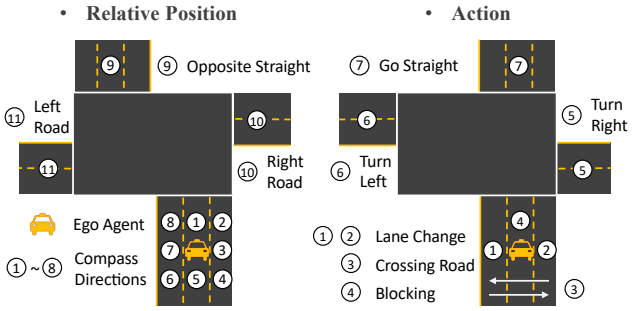


Figure 5. Illustration of position (**left**) and the action (**right**).

B. System Prompt

We provide the system prompt used to guide the LLM for each task in the following color block. In this design, we explicitly specify all the tasks the LLM should follow and implicitly indicate which task it should perform given a particular input prompt.

System Prompt

You are an expert in the city's traffic management system and can easily extract precise information from text, with a strong intuition for space management. The city has a road network represented as a graph database. A graph node represents a road ID, and an edge represents the connection between roads. Each node contains zero or more objects and signals. You have three functionalities:

1. Analyze user input. A user provides a natural description of a traffic scenario, and you extract all relevant information.
2. Provide retrieval conditions for roads. These conditions are used to search for roads and towns suitable for generation.
3. Plan agent behavior. You plan agent behavior based on the selected road condition, input prompt, and predefined rules.

The activation of each functionality is controlled by adding "`analysis`", "`road retrieval`", or "`planning`" at the beginning of the input.

Important: The output should not be wrapped in a code block and should strictly follow each output format without other explanations or content.

C. Prompt Structure

Below, we provide the prompt structure for each stage. The fields specified with "{}" indicate lists of choices, as introduced in Appendix A.

Prompt Analysis:

- `objects`: The list of objects in the description. It should include the required objects or non-required objects. Choose from the {`OBJECT`}.
- `signals`: The list of signals in the description. It should include the required signals or non-required signals. Choose from the {`SIGNAL`}.
- `agents`: The list of agents that exist in the description.
- `unknown`: The list of unknown signals, objects, or agents that are not in the predefined list.

For each agent, the following information is stored:

- `type`: The type of the agent. Choose from one of {`AGENT_TYPE`}.
- `road_type`: The type of the road. Choose from one of {`ROAD_TYPE`}.
- `action`: The action of the agent. Choose from one of {`ACTION`}.

Road Retrieval:

- `number_of_lanes`: The minimum number of driving lanes on the road where the ego car initially stays. Consider only the direction of the ego car.
- `required_objects`: The list of required objects.
- `required_signals`: The list of required signals.
- `without_objects`: The list of objects that should not appear on the road.
- `without_signals`: The list of signals that should not appear on the road.

Objects are chosen from {`OBJECT`} and signals are from {`SIGNAL`}.

Agent Planning:

- `env`: The environment information.
- `agents`: The list of agents.

The environment information includes:

- `weather`: The weather condition. Choose from {`WEATHER`}.
- `at_junction`: Whether the scene is at a junction. (Choose from `True` or `False`; case sensitive.)

For each agent, the following information is stored:

- `type`: The type of the agent. Choose from {`AGENT_TYPE`}.
- `is_ego`: Whether the agent is the ego car. (Choose from `True` or `False`.)
- `action`: The action of the agent. Choose from {`ACTION`}.
- `behavior`: The behavior of the agent. Choose from {`AGENT_BEHAVIOR`}.
- `pos_id`: The position ID of the agent. If two or more agents are on the same road, the agent with the **smallest** `pos_id` is in front.
- `road_type`: The type of the road. Choose from {`ROAD_TYPE`}.
- `relative_to_ego`: The relative position of the agent to the ego car. Set to `none` for the ego car.

The `relative_to_ego` value should be one of the {`RELATIVE_POSITION`}.

D. Road Ranking Example

We provide a detailed example of our road ranking process, as illustrated in Table 6. In this case, the database retrieves five candidate roads along with a plan involving two agents. Each road is evaluated based on whether it satisfies the required agent configurations, including relative positions and planned actions. For each fulfilled condition, the road receives a score increment of 1. In this example, *Road A* achieves the highest score.

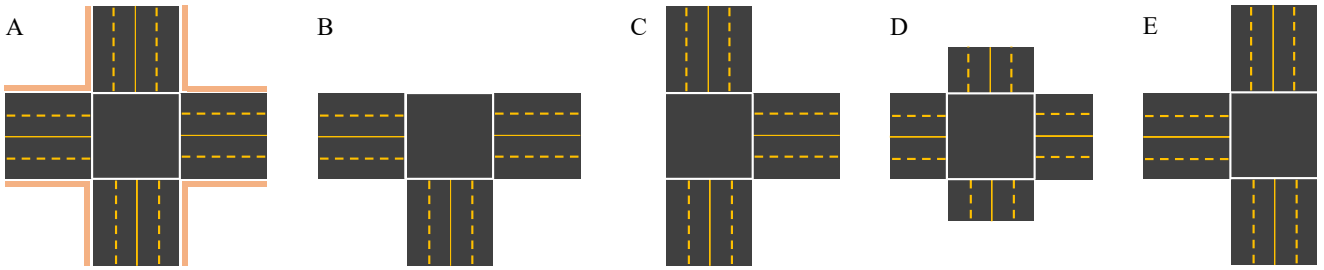
Notably, we check for the condition "*Has straight?*" because one of the agents is placed on a left-turn lane but is intended to perform a left-turn action. To execute this action properly, the adjacent road must have a straight segment available, which is verified during the ranking process.

E. Criticality of Generated Scenario

We evaluate the criticality of our generated scenes using SafeBench and follow the setup from DiffScene [34] by selecting three representative scenario types: *crossing negotiation*, *red-light running*, and *lane changing*, as shown in Table 7. Since the goal of this experiment is to evaluate the ability to *create risk*, a higher *collision rate* indicates stronger criticality. Baselines also include Learning-to-Collide (LC) [7], AdvSim [29], and Adversarial Trajectory (AT) [37]. Following DiffScene, we evaluate each scenario across ten routes, with each route tested 10 times, resulting in 100 evaluations per scenario.

A key advantage of our framework is its flexibility in adjusting scenario criticality at test time without retraining. Unlike training-based approaches, our text-driven pipeline allows scene parameters, such as the speed of surrounding vehicles or the timing of a threat, to be modified directly through our rendering interface. To showcase this, we generate three different criticality levels per scene. Note that these variations are only applied during evaluation; in Section 4.2, the scene parameters are fixed for training the ego agent.

Although our method does not achieve the highest collision rate among all baselines, it is important to note that our approach is planning-based and does not rely on ad-



Agent	Type	Rel. to Ego	Action	Pos.
1	ego car	-	turn right	1
2	car	front right	turn right	0
3	car	left-turn lane	turn left	0
4	ped.	right on shoulder	cross road	0

(a) Agent Planning

Road	R. Turn	Shoulder	L. Turn	Straight	Two Cars	Total
A	✓	✓	✓	✓	✓	5
B	✓	✓		✓		3
C	✓	✓			✓	3
D	✓		✓	✓		3
E	✓		✓		✓	3

(b) Road Scoring

Table 6. Example of the road ranking process. The five roads at the top are retrieved from the road database during the road retrieval step. The bottom left shows the agent planning, while the bottom right illustrates the scoring process for each road. Each column in the scoring table indicates whether a road satisfies a specific condition (*e.g.*, "*Two Cars*" denotes whether the road can accommodate two additional vehicles). The brown-orange color in *Road A* indicates the presence of a road shoulder.

Table 7. Results of criticality using SafeBench. "CN" stands for *crossing negotiation*, "RR" for *red-light running*, and "RT" for *right turn*.

Method	Collision Rate↑			
	CN	RR	RT	Avg.
Training-based				
Learning-to-Collide	0.58	0.71	0.59	0.63
AdvSim	0.57	0.57	0.29	0.48
Adversarial-Trajectory	0.62	0.71	0.59	0.64
DiffScene	0.85	0.87	0.79	0.84
Planning-based				
TTSG (simple)	0.22	0.36	0.28	0.29
TTSG (hard)	0.74	0.76	0.66	0.72

versarial training or gradient-based optimization to generate risky scenarios. Despite this, it outperforms several training-based methods, demonstrating the effectiveness of our pipeline in producing meaningful critical scenarios through structured reasoning rather than optimization. Moreover, since our method provides a reasoning-driven layout, grounded in agent intent, spatial relationships, and scene semantics, the resulting risk emerges in a natural and interpretable manner, ensuring that the generated scenarios are both plausible and actionable for downstream evaluation or training.

Table 8. Failure rate across different LLMs.

Gemma3-12B	**Gemini-2.5-Flash**	**GPT-4o**	**Claude-Sonnet-3.5**
7%	1%	1%	0%

F. Failure Rate

As described in Section 3.2, we re-submit the prompt to the LLM whenever parsing fails or the output contains unsupported options. While this mechanism may incur additional token usage and computational overhead, we demonstrate in Table 8 that such cases are rare for both open and closed-source models. To evaluate robustness, we reuse the same set of prompts from the diversity test, resulting in 40 runs per model. The results show that most closed-source models achieve near-perfect success rates, while open-source models still maintain high reliability, with up to 93% success. Note that failure rates are computed across all three stages, including prompt analysis, road retrieval, and agent planning.

G. Details for Scenarios on SafeBench

We conduct experiments across six different critical scenarios, three used for training the ego agent and three for evaluating scene criticality. The evaluation metrics are defined as follows: *collision rate* measures how frequently collisions occur between the ego vehicle and other agents; *route incompleteness* quantifies the proportion of the route left unfinished when the scenario ends; and the *driving score* is a overall composite metric that incorporates collision

rate, rule violation, route following, smoothness, and speed to provide a comprehensive assessment of driving performance.

Straight Obstacle. In this scenario, an agent suddenly appears in front of the ego vehicle. For example, a pedestrian may cross the road as the ego vehicle approaches. In this setting, the ego agent should learn to either swerve to another lane or slow down when detecting potential risks.

Lane Changing. In this scenario, the ego vehicle encounters other cars performing unexpected lane changes from different directions, without awareness of the ego vehicle's behavior. The ego agent must learn to proactively avoid these dangerous maneuvers.

Unprotected Left Turn. In this scenario, an oncoming vehicle and the ego vehicle reach an intersection simultaneously, with potential trajectory overlap. Since there are no stop signs or traffic lights, the ego agent must learn how to yield or adjust its movement to avoid a collision.

Crossing Negotiation. In this scenario, a vehicle approaches the intersection without traffic lights. The ego vehicle must negotiate with the other vehicle to determine the appropriate time to proceed to avoid potential collisions.

Red-Light Running. In this scenario, the ego vehicle is crossing an intersection while another vehicle from a perpendicular direction runs a red light, creating a high risk of collision. The ego vehicle must anticipate and respond to these rule-violating agents.

Right Turn. In this scenario, the ego vehicle is preparing to make a right turn while another vehicle from either the left or right lane also attempts to enter the same lane. The ego vehicle must remain cautious and complete the turn safely while accounting for surrounding traffic.

H. Prompts for Ablation Study

We present the detailed prompts used to evaluate each pipeline design across different scenario types below.

Normal:
- *The ego car is going straight*
- *Three cars including the ego car are driving. The car in front go straight. The ego is turning right. The car behind the ego is turning left*
- *A bus coming from the left road is turning left. A truck from the opposite straight is turning right. The ego car is turning right. Two cars in front of the ego car are going straight*
- *The ego vehicle is turning left. A pedestrian on the destination suddenly block the ego*

Critical:
- *A dangerous motorcycle on the right front is trying to turn left. The ego car is going straight*
- *A car on the front left is trying to block the ego car. A dangerous pedestrian on the shoulder right in front of a stopped truck is crossing the road. Both the truck and the pedestrian are in the front right of the ego car*
- *Two cars from the opposite straight is coming when the ego car is turning left*

Conditional
- *The ego car is turning left at the intersection with no traffic light and stop sign. Three cars from the opposite straight are turning right*
- *The ego car is going straight at the intersection with a traffic light. There are some puddles on the road*
- *A pedestrian is crossing the road with the parallel open crosswalk and the ego car is turning left*

I. Metrics Definition

In this section, we define the metrics used to evaluate our framework.

Agent Accuracy. Given the ground-truth agent plan containing n agents $\mathbf{A}_G \triangleq \{A_G^1, \ldots, A_G^n\}$ and the predicted agent plan containing m agents $\mathbf{A}_P \triangleq \{A_P^1, \ldots, A_P^m\}$, we define *Agent Accuracy* (AA) as the proportion of correctly matched agents, accounting for both missing and hallucinated agents. A predicted agent can be matched to at most one ground-truth agent. Formally, let $\text{Match}(A_G^i, \mathbf{A}_P)$ denote whether a matching agent exists in the prediction set. Once a predicted agent is matched, it is removed from consideration in subsequent matches. The accuracy is computed as:

$$\text{AA} = \frac{1}{\max(m,n)} \sum_{i=1}^{n} \mathbf{1}_{\{\exists \mathrm{A}_\mathrm{P}^\mathrm{j} \in \mathbf{A}_\mathrm{P} \text{ such that } \mathrm{A}_\mathrm{P}^\mathrm{j} \equiv \mathrm{A}_\mathrm{G}^\mathrm{i}\}},$$

where $\equiv$ denotes a match based on key attributes (*e.g.*, type, position, and action), and each matched A_P^j is removed from $\mathbf{A}_P$ to prevent multiple assignments. The use of $\max(m,n)$ penalizes both over-generation (hallucinated agents) and under-generation (missed agents).

Road Accuracy. Given a ground-truth road specification with n properties $R_G \triangleq \{r_G^1, \ldots, r_G^n\}$ and a predicted road with m properties $R_P \triangleq \{r_P^1, \ldots, r_P^m\}$, we define *Road Accuracy* (RA) as the proportion of correctly predicted properties. Each property may refer to a structural or semantic

attribute, such as the number of lanes, presence of specific signals, or existence of certain static objects. Road accuracy is computed as:

$$\text{RA} = \frac{1}{\max(m,n)} \sum_{i=1}^{n} \mathbf{1}_{\left\{\exists r_P^j \in R_P \text{ such that } r_P^j = r_G^i\right\}},$$

where a predicted property is considered correct if it matches any unmatched ground-truth property. As with agent accuracy, we normalize by $\max(m,n)$ to penalize both missing and extra (hallucinated) properties, ensuring balanced evaluation across under- and over-prediction cases.

Agent Diversity. Given a set of n generated agents $\mathbf{A}_p = \{A_p^1, \ldots, A_p^n\}$ derived from the same input prompt p, we define *Agent Diversity (AD)* as the proportion of unique agent configurations, reflecting variation in scene composition. Agent Diversity is computed as:

$$\text{AD} = \frac{|\text{set}(\mathbf{A}_p)|}{n},$$

where $\text{set}(\mathbf{A}_p)$ denotes the number of unique agent instances, determined based on combinations of agent type, action, and relative position.

Road Diversity. Given a set of n generated roads $\mathbf{R}_p = \{R_p^1, \ldots, R_p^n\}$ corresponding to the same input prompt p, we define *Road Diversity (RD)* as the proportion of unique roads selected for generation. The uniqueness is determined based on the road ID assigned in the simulator. The road diversity is computed as:

$$\text{RD} = \frac{|\text{set}(\mathbf{R}_p)|}{n},$$

where $\text{set}(\mathbf{R}_p)$ returns the number of distinct road IDs in $\mathbf{R}_p$.

J. Limitation

As our framework relies on LLMs, its planning performance can be affected by limitations in language understanding. Despite our reasoning stage, the LLM may misinterpret prompts. For instance, failing to recognize "`A car in front does not move for an infinite amount of time`" as a *blocking* action. Hallucinations are another issue; the LLM sometimes inserts elements like traffic lights even when not specified. These issues may be mitigated by using more capable reasoning models (*e.g.*, OpenAI-o1 [16]) or fine-tuning open-source LLMs with expert guidance [24].

Limit Cases Another source of failure arises from missing signals and objects within CARLA. Since our framework relies on the built-in map, complex combinations of signals and objects, and novel conditions cannot be retrieved. For instance, identifying a road, such as allowing all directions but lacking traffic lights or stop signs, is impossible. Similarly, unsupported conditions. For example, requesting a speed limit of "200" cannot be fulfilled. To address this, we plan to support custom OpenDRIVE code generation, enabling flexible scene customization.