

A Algorithm of Training and Sampling in Details

We mostly follow the training and sampling procedures from [6] and show the detailed algorithms for training and sampling in the following.

A.1 Training

Algorithm 1 Training

```

1: Require a real-world driving dataset  $D$ , conditional diffusion model to train  $\mu_\theta^0$ , transition function  $f$ , denoising steps  $K$ .
2: while not converge do
3:    $\mathbf{c}, \tau^0 \sim D$ 
4:    $k \sim \{1, \dots, K\}$ 
5:    $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
6:   Corrupt action trajectory  $\tau_a^k = \sqrt{\bar{\alpha}_k} \tau_a^0 + \sqrt{1 - \bar{\alpha}_k} \epsilon$  with  $\bar{\alpha}_k = \prod_{l=0}^k 1 - \beta_l$ 
7:   Get the corresponding state trajectory  $\tau_s^k = f(s_0, \tau_a^k)$ 
8:   Use model to predict the uncorrupted trajectory  $\hat{\tau}_a^0 = \mu_\theta^0(\tau^k, k, \mathbf{c})$ 
9:   Get the predicted state trajectory  $\hat{\tau}^0 = [\hat{\tau}_a^0; f(s_0, \hat{\tau}_a^0)]$ 
10:  Take gradient step on  $\nabla_\theta \|\tau^0 - \hat{\tau}^0\|^2$ 
11: end while

```

Contrary to [6], which samples trajectories at the agent level, we opt for scene-level trajectory sampling, allowing the model to make joint predictions on all scene agents. The process is detailed in Algorithm 1. During each training iteration, the context \mathbf{c} and the ground-truth trajectory τ^0 are sampled from a real-world driving dataset, and the denoising step k is uniformly selected from $\{1, \dots, K\}$. We derive the noisy input τ^k from τ^0 by initially corrupting the action trajectory via $\tau_a^k = \sqrt{\bar{\alpha}_k} \tau_a^0 + \sqrt{1 - \bar{\alpha}_k} \epsilon$, with $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\bar{\alpha}_k = \prod_{l=0}^k 1 - \beta_l$. Subsequently, the corresponding state is computed as $\tau_s^k = f(s_0, \tau_a^k)$. The diffusion model indirectly parameterizes μ_θ in eq. (2) by predicting the uncorrupted trajectory $\hat{\tau}^0 = [\hat{\tau}_a^0; f(s_0, \hat{\tau}_a^0)]$, where $\hat{\tau}_a^0 = \mu_\theta^0(\tau^k, k, \mathbf{c})$ is the network's direct output (see [12, 13, 40]). We then use a simplified loss function to train the model as follows:

$$L(\theta) = \mathbb{E}_{\epsilon, k, \tau^0, \mathbf{c}} [\|\tau^0 - \hat{\tau}^0\|^2]. \quad (5)$$

A cosine variance schedule [13, 40] is utilized in the diffusion process, employing $K = 100$ diffusion steps.

A.2 Sampling

We show the guided sampling algorithm in Algorithm 2 which is directly from [6] as the notations and procedure remain the same. The key difference is that our diffusion model formulation and backbone models are all at scene-level rather than agent-level as in [6]. The scene-level formulation helps to improve scene-level realism and decrease failure rates as the agents' interactions can be captured by the model inherently.

Algorithm 2 Guided Sampling

```

1: Require conditional diffusion model  $\mu_\theta$ , transition function  $f$ , guide  $\mathcal{J}$ , scale  $\alpha$ , covariances  $\Sigma^k$ , diffusion steps  $K$ , inner gradient descent steps  $W$ , number of actions to take before re-planning  $l$ .
2: while not done do
3:   Observe state  $s_0$  and context  $\mathbf{c}$ 
4:   Initialize trajectory  $\tau_a^K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ;  $\tau_s^K = f(s_0, \tau_a^K)$ ;  $\tau^K = [\tau_a^K; \tau_s^K]$ 
5:   for  $k = K, K-1, \dots, 1$  do
6:      $\mu := \tau_a^{k-1} = \mu_\theta(\tau^k, k, \mathbf{c})$ 
7:      $\mu^{(0)} = \mu$ 
8:     for  $j = 1, \dots, W$  do
9:        $\mu^{(j)} = \mu^{(j-1)} + \alpha \nabla \mathcal{J}(\mu^{(j-1)})$ 
10:       $\Delta \mu = |\mu^{(j)} - \mu^{(0)}|$ 
11:       $\Delta \mu \leftarrow \text{clip}(\Delta \mu, -\beta_k, \beta_k)$ 
12:       $\mu^{(j)} \leftarrow \mu^{(0)} + \Delta \mu$ 
13:     end for
14:      $\tau_a^{k-1} \sim \mathcal{N}(\mu^{(M)}, \Sigma^k)$ ;  $\tau_s^{k-1} = f(s_0, \tau_a^{k-1})$ ;
        $\tau^{k-1} = [\tau_a^{k-1}; \tau_s^{k-1}]$ 
15:   end for
16:   Execute first  $l$  actions of trajectory  $\tau^0$ 
17: end while

```

Following [6, 12], the predicted mean is a weighted sum between the predicted clean action trajectory and the input action trajectory from last denoising step:

$$\hat{\tau}_a^{k-1} = \mu_\theta(\tau^k, k, \mathbf{c}) = \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1 - \bar{\alpha}_k} \hat{\tau}_a^0 + \frac{\sqrt{\alpha_k}(1 - \bar{\alpha}_{k-1})}{1 - \bar{\alpha}_k} \tau_a^k \quad (6)$$

The process of perturbing the predicted means from the diffusion model using gradients of a specified objective is summarized in algorithm 2. Following [6], we use an iterative projected gradient descent with the Adam optimizer and *filtration*, i.e., we guide several samples from the diffusion model and choose the one with the best rule satisfaction based on \mathcal{J} .

B More Details on Architecture

B.1 Detailed Architecture

We show the detailed data flow of our proposed architecture in Figure A1. Its main difference with the simplified architecture shown in Figure 2 is that we show position encoding, rFFN, and the details of the guidance module explicitly.

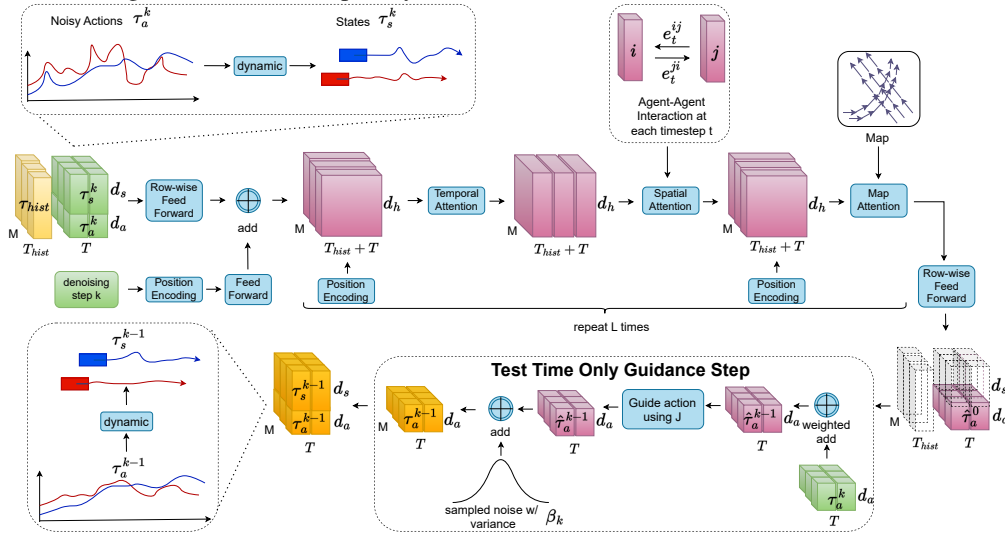


Figure A1: Test time denoising step using multi-agent spatial-temporal transformer. d_s , d_a , and d_h represent the dimensions of action, state, and latent for each vehicle per timestep.

B.2 Gated Attention

Following [35], we use a variant of the original scaled dot-product attention block. In particular, we use a gating function to fuse the environmental features m_i^t with the central agent's features h_i^t , enabling the block to have more control over the feature update. The resulting query, key, and value vectors of the social attention layer are taken as inputs to the block:

$$\begin{aligned} \alpha_t^i &= \text{softmax} \left(\frac{\mathbf{q}_t^{i^\top}}{\sqrt{d_k}} \cdot [\{\mathbf{k}_t^{ij}\}_{j \in \mathcal{N}_i}] \right), \\ \mathbf{m}_t^i &= \sum_{j \in \mathcal{N}_i} \alpha_t^{ij} \mathbf{v}_t^{ij}, \\ \mathbf{g}_t^i &= \text{sigmoid}(\mathbf{W}^{\text{gate}} [\mathbf{h}_t^i, \mathbf{m}_t^i]), \\ \hat{\mathbf{h}}_t^i &= \mathbf{g}_t^i \odot \mathbf{W}^{\text{self}} \mathbf{h}_t^i + (1 - \mathbf{g}_t^i) \odot \mathbf{m}_t^i, \end{aligned} \quad (7)$$

where \mathcal{N}_i is the set of agent i 's neighbors (all the agents except the agent itself within a certain social radius), \mathbf{W}^{gate} and \mathbf{W}^{self} are learnable matrices, and \odot denotes element-wise product.

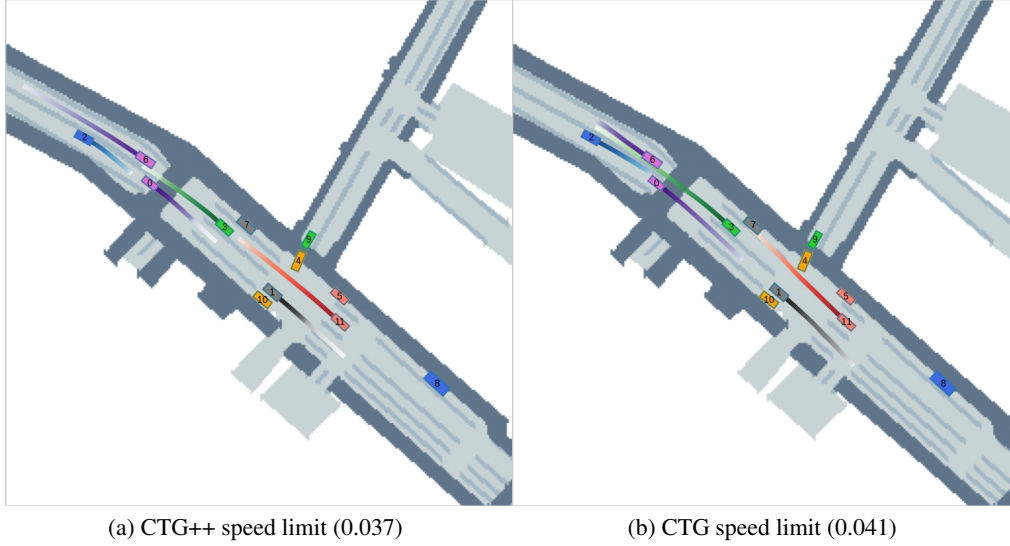


Figure A2: Qualitative comparison between CTG++ and CTG under speed limit STL rule (the numbers in parentheses represent rule violations). CTG++ achieves lower rule violation than CTG. Besides, CTG involves collision between the blue vehicle and the green vehicle.

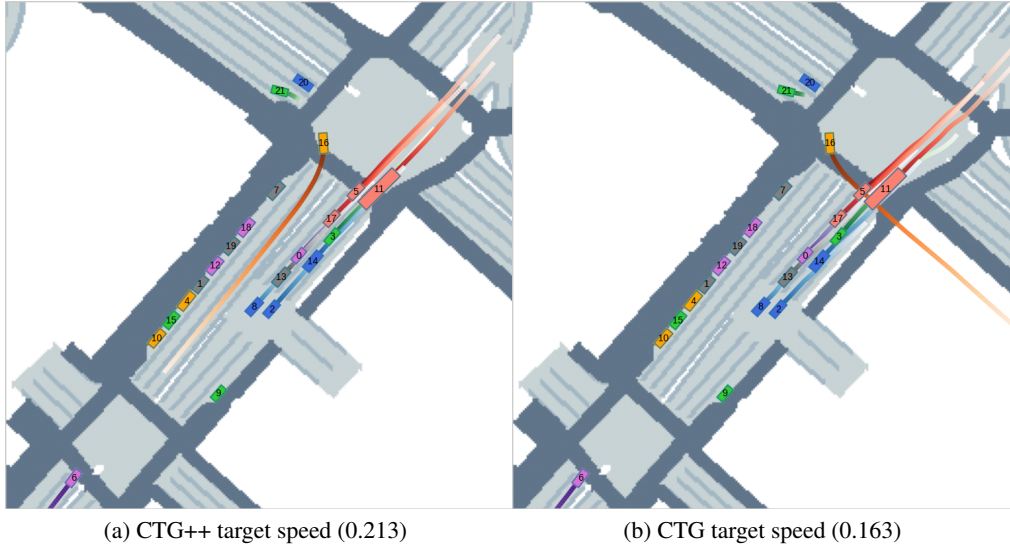


Figure A3: Qualitative comparison between CTG++ and CTG under target speed STL rule (the numbers in parentheses represent rule violations). Although CTG achieves a bit better target speed rule satisfaction, it involves a vehicle collides with crossing vehicles and then goes off-road.

493 In this section, we show a few qualitative examples (Figure A2 - Figure A7) comparing CTG++ and
 494 the strongest baseline (in terms of rule satisfaction) under the STL rules. Overall, CTG++ generates
 495 realistic, rule-satisfying trajectories. The baseline method can usually also satisfy the rule. However,
 496 their trajectories usually sacrifice one or more of the following aspects: (1) the trajectories are curvy,
 497 unrealistic, (2) the trajectories involve off-road accidents, and (3) the agent interaction is sub-optimal
 498 leading to collision(s).

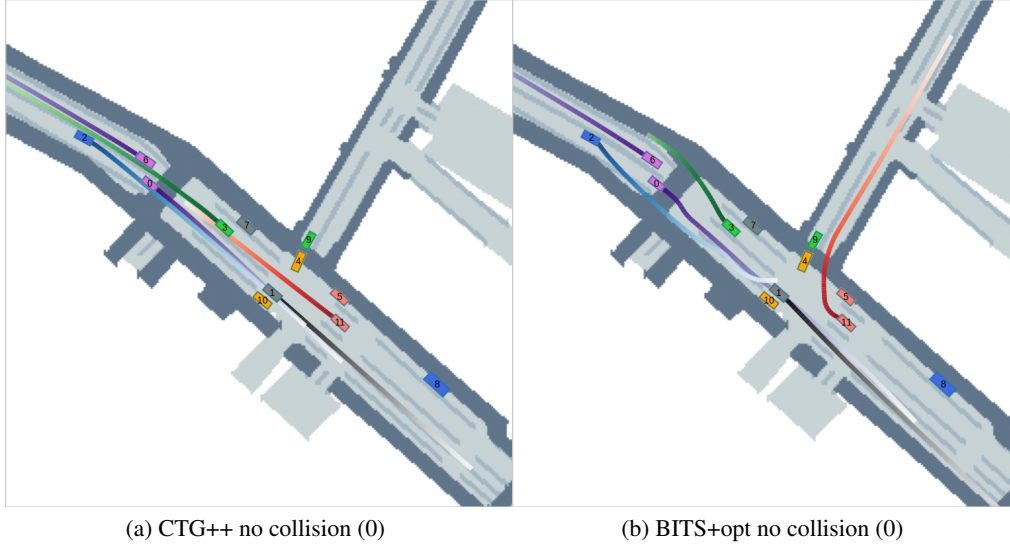


Figure A4: Qualitative comparison between CTG++ and BITS+opt under no collision STL rule (the numbers in parentheses represent rule violations). Both methods satisfies the rule perfectly as no collision happens. However, BITS+opt have highly curve, unrealistic trajectories as the cost of satisfying the rule.

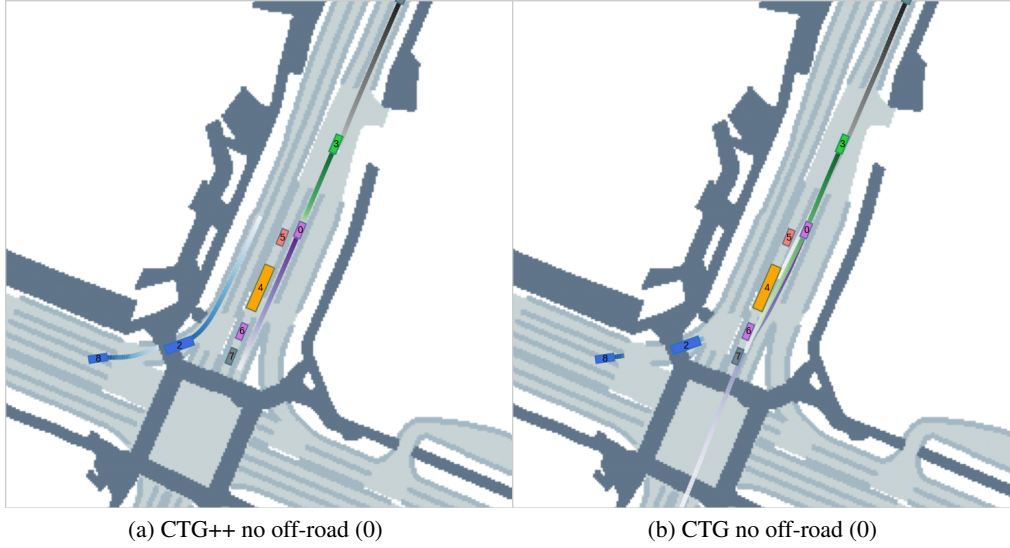


Figure A5: Qualitative comparison between CTG++ and CTG under no off-road STL rule (the numbers in parentheses represent rule violations). Both methods satisfies the rule perfectly as no off-road happens. However, CTG lead to multiple collisions among the pink vehicle and vehicles that are stationary.

D Hyperparameters

D.1 Training Hyperparameters

CTG++ is trained on a machine with Intel i9 12900 and NVIDIA GeForce RTX 3090. It takes approximately 10 hours to train CTG++ for 50K iterations. We use Adam optimizer with a learning rate of $1e-4$.

D.2 Pair Selection Criteria for GPT query based rules

We choose two vehicles A and B in each scene such that they satisfy the following criteria:

- Both have current speed larger than 2m/s.

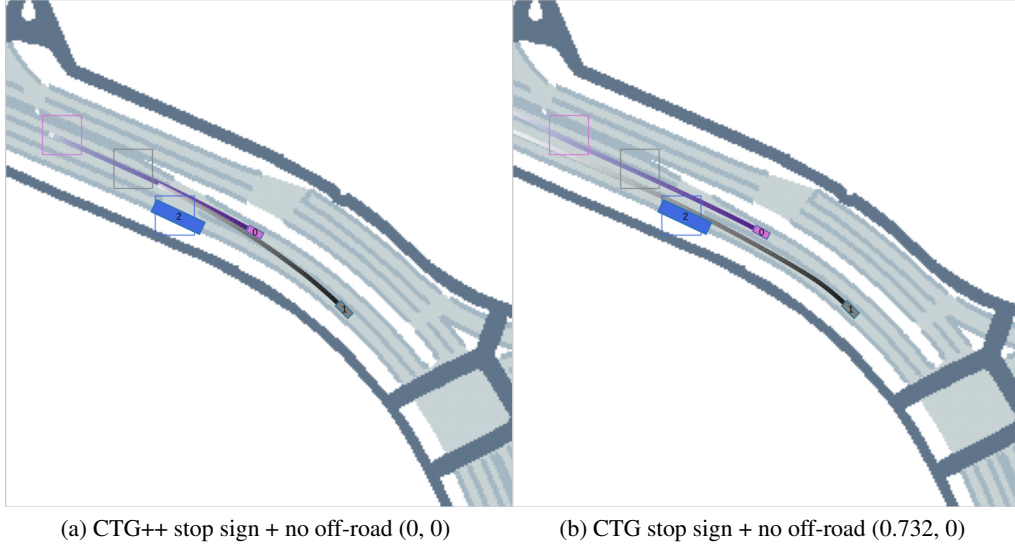


Figure A6: Qualitative comparison between CTG++ and CTG under stop sign and no off-road STL rule (the numbers in parentheses represent rules violations). Vehicles are supposed to stop within the marked bounding boxes without going off-road. CTG++ satisfies both rules while CTG only satisfies the no off-road rule. Besides, CTG involves a collision between the grey vehicle and the blue vehicle.

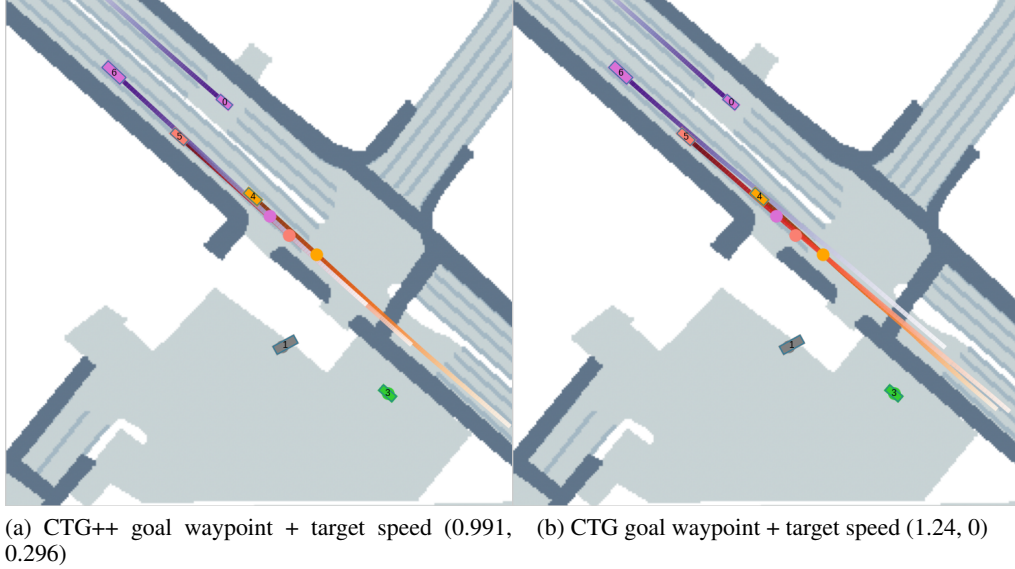


Figure A7: Qualitative comparison between CTG++ and CTG under goal waypoint + target speed STL rule (the numbers in parentheses represent rules violations). Vehicles are supposed to reach the marked waypoints with target speed (same speed as in the dataset). CTG++ satisfies both rules better than CTG. Besides, CTG involves a collision between the two orange vehicles in the end.

- 507 • At 0s and 2s, the distance between A and B is within the range 10m to 30m.
- 508 • At 0s and 2s, the orientation difference between a and b is smaller than 108 degrees (for
- 509 GPT collision) and 36 degrees (for GPT keep distance).

510 The criteria is a coarse-grained filtration for those pairs that are more likely to have keep distance /
511 collision interactions in the original training dataset. If more than one pair in the scene satisfy the
512 following criteria, we select the pair with smallest distance. If none of pairs in a scene satisfy the
513 following criteria, we skip the scene. After the filtrations, out of the 100 validation scenes, we have
514 50 scenes remained for **GPT collision** and 40 scenes for **GPT keep distance**.

E Experiment Details

E.1 Metrics of Rule Violation

We provide the details for the metrics we use for measuring rule violation in this section. For all the metrics of rule violation, we average the metrics over all validation scenes. Besides, they are designed such that the smaller the better (i.e., rules are better satisfied).

GPT Keep Distance. the following vehicle’s (in the chosen pair) average l2 distance deviation from the specified range.

GPT Collision. if a collision happens between the two vehicles in the chosen pair.

No Collision. collision rate of all vehicles in a scene.

Speed Limit. average deviation from the speed limit of all vehicles in a scene.

Target Speed. average deviation from the target speed of all vehicles in a scene.

No Offroad. off-road rate of all vehicles in a scene. We consider a vehicle going off-road if its center goes off-road.

Goal Waypoint. average vehicle’s smallest l2 distance deviation from the specified corresponding goal waypoints of all vehicles in a scene.

Stop Sign. average smallest speed within the stop sign region of all vehicles in a scene.

F Details of Language Interface

In this section, we provide more details and limitation of our proposed language interface for traffic simulation.

F.1 Details of Vehicle Indexing

In practice, instead of using color for vehicles which is used in Figure 1 for better illustration purpose, we use indices according to the context from the driving dataset. The user can tell GPT4 the vehicles to control via their indices, e.g., ”vehicle 1 should collide with vehicle 2”.

F.2 Details of Prompting

In Figure 4, we provide an example of a pre-defined API function and a query-loss function pair. In our experiments, we additionally provide the following API functions:

transform_coord_world_to_agent_i. this function transform the predicted position and yaw from world coordinate to the agent i coordinate.

select_agent_ind. this function returns the slice of x with index i.

get_current_lane_projection. this function returns the projection of each vehicle predicted trajectory on its current lane in agent-centric coordinate.

get_left_lane_projection. this function is similar to get_current_lane except it returns the left lane waypoints. If there is no left lane, the original trajectory will be returned.

get_right_lane_projection. this function is similar to get_current_lane except it returns the right lane waypoints. If there is no right lane, the original trajectory will be returned.

In addition to the acceleration loss paired example shown in Figure 4, we provide another query-loss function pair example. ”Generate a loss class such that, vehicle 1 should always stay on the left side of vehicle 2.” The corresponding function penalize the cases when vehicle 1 not on the left side of vehicle 2. This function provides GPT4 a sense of the relationship between direction and the trajectories.

555 We additionally specify the dimension of the input trajectory and the input and output of the expected
556 loss function wrapped in a loss class such that GPT4 know which dimension of the trajectory to
557 operate on when needed: "The generated loss class should have a function: forward(x, data_batch,
558 agt_mask). x is a tensor representing the current trajectory with shape (B, N, T, 6) where B is the
559 number of vehicles (consisting of vehicle with index 0 to B-1), N is the number of samples for each
560 vehicle, T is the number of timesteps (each represents 0.1s), and 6 represents the (x, y, vel, yaw,
561 acc, yawvel) in corresponding agent coordinate of each vehicle. data_batch is a dictionary that can
562 be used as parameter for relevant APIs. The function should return a loss for every sample of every
563 vehicle with the shape (B, N) or return a loss for every sample with the shape (N)."

564 **F.3 Failure Cases**

565 The main limitation of the current language interface is on the complex interactions between the
566 vehicles and the map. As we don't explicitly pass the map information into the language interface, it
567 cannot handle commands involving heavy interaction with the map. For example, "vehicle A and B
568 move to the rightmost lane one by one and then both turn right at the next intersection". However, we
569 believe if one provides more helper functions (especially those interacting with the map) and more
570 relevant examples to the language interface, LLM can handle such more complicated commands.
571 However, as the current work aims to provide preliminary results on the feasibility of text-to-traffic,
572 we leave that for future work.