ReflectDrive-2: Reinforcement-Learning-Aligned Self-Editing for Discrete Diffusion Driving

Huimin Wang[*] [†], **Yue Wang**[*], **Bihao Cui**[‡]

Pengxiang Li, Ben Lu, Mingqian Wang, Tong Wang, Chuan Tang, Teng Zhang, Kun Zhan

LiAuto

Abstract

We introduce ReflectDrive-2, a masked discrete diffusion planner with a separate action expert for autonomous driving that represents plans as discrete trajectory tokens and generates them through parallel masked decoding. This discrete token space enables in-place trajectory revision: AutoEdit rewrites selected tokens using the same model, without requiring an auxiliary refinement network. To train this capability, we use a two-stage procedure. First, we construct structure-aware perturbations of expert trajectories along longitudinal progress and lateral heading directions and supervise the model to recover the original expert trajectory. We then fine-tune the full decision–draft–reflect rollout with reinforcement learning (RL), assigning terminal driving reward to the final post-edit trajectory and propagating policy-gradient credit through full-rollout transitions. Full-rollout RL proves crucial for coupling drafting and editing: under supervised training alone, inference-time AutoEdit improves PDMS by at most 0.3, whereas RL increases its gain to 1.9. We also co-design an efficient reflective decoding stack for the decision–draft–reflect pipeline, combining shared-prefix KV reuse, Alternating Step Decode, and fused on-device unmasking. On NAVSIM, ReflectDrive-2 achieves 91.0 PDMS with camera-only input and 94.8 PDMS in a best-of-6 oracle setting, while running at 31.8 ms average latency on NVIDIA Thor.

1 Introduction

Planning errors in imitation-learned driving policies are typically structured rather than random, arising mainly as longitudinal speed errors and lateral heading drift. These are also major directions of covariate shift in imitation learning [Bansal et al., 2019, Codevilla et al., 2018], making in-place structured revision a natural planning primitive. Existing planners do not support this well: modular and end-to-end methods [Fan et al., 2018, Kato et al., 2018, Bojarski et al., 2016, Hu et al., 2023, Chitta et al., 2022, Hu et al., 2022, Jiang et al., 2023a] output a single trajectory, autoregressive VLA planners [Kim et al., 2024, Tian et al., 2024, Sima et al., 2024] must regenerate the full sequence to revise it, and continuous diffusion planners [Janner et al., 2022, Chi et al., 2023, Liao et al., 2025, Xing et al., 2025] denoise Gaussian noise rather than structured driving errors. In contrast, masked discrete diffusion [Austin et al., 2021, Nie et al., 2025, Song et al., 2025, Bie et al., 2026] naturally supports in-place revision by re-masking and rewriting selected trajectory tokens.

Naively appending a self-editing stage yields limited benefit because supervised training decouples drafting and editing: the drafter is optimized without regard to the post-edit outcome, and the

[*]Equal Contribution.

[†]Contact author: wanghuimin1@lixiang.com

[‡]Project Lead.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

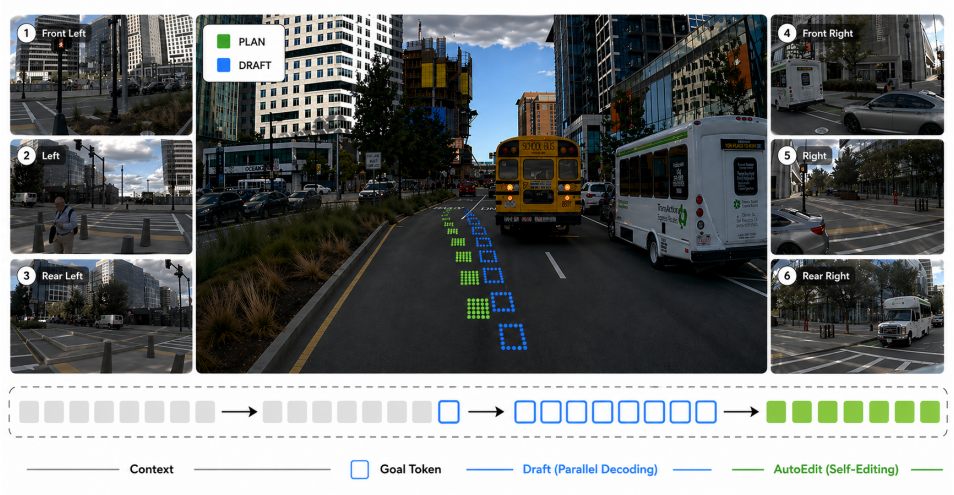

Figure 1: **ReflectDrive-2 plans through a decision–draft–reflect process in a shared discrete token space.** Conditioned on surround-view cameras, navigation, and ego-state, a goal-point posterior commits a single Goal Token, masked discrete diffusion parallel-decodes a trajectory Draft (blue), and AutoEdit rewrites tokens in place to yield the final Plan (green).

editor receives no signal about which rewrites improve closed-loop behavior. We address this with reinforcement learning over the full decision-draft-reflect rollout, assigning a shared terminal reward to both stages, which jointly improves draft generation and the AutoEdit refinement built upon it.

We call the resulting system *ReflectDrive-2*, a reflective masked-diffusion VLA planner, and its self-editing mechanism **AutoEdit**. The model takes panoramic images, navigation tokens, and ego state as input, and predicts discrete trajectory tokens for a 4-second plan, with the final waypoint tokens defining a behavior hypothesis. Goal points are sampled from the predicted goal posterior using top-K sampling with non-maximum suppression. AutoEdit is pretrained with structure-aware longitudinal and lateral perturbations, then jointly optimized with the drafter under RL using a shared vision-language backbone.

To summarize, our main contributions are:

- **ReflectDrive-2.** We propose *ReflectDrive-2*, a driving VLA that plans via *decision–draft–reflect*: a goal posterior produces behavior hypotheses, masked discrete diffusion drafts trajectories, and AutoEdit revises them in the same token space. It achieves 91.0 camera-only PDMS and 94.8 PDMS with best-of-6 oracle selection on NAVSIM.
- **Reward-coupled AutoEdit.** We introduce **AutoEdit**, a self-correction mechanism pre-trained with structure-aware perturbations and jointly optimized with the drafter by RL over the full rollout, substantially improving editing effectiveness.
- **Efficient reflective decoding.** We develop a reflection-aware runtime stack with shared-prefix KV cache reuse, ASD as temporal AutoEdit, and fused CUDA unmasking, reaching 31.8 ms average latency on NVIDIA Thor.

2 Related Work

2.1 End-to-End and VLA Planning

End-to-end planners map sensor inputs directly to trajectories without explicit modular decomposition [Chitta et al., 2022, Hu et al., 2023, Jiang et al., 2023a, Hu et al., 2022], while SMART [Feng et al., 2024] autoregressively predicts tokenized multi-agent trajectories. VLA planners [Li et al., 2025a, Zhou et al., 2025, Li et al., 2025b, Kim et al., 2024, Tian et al., 2024, Sima et al., 2024] leverage language priors but remain sequential, so latency scales with trajectory length and correction requires full re-decoding. Continuous diffusion planners [Janner et al., 2022, Chi et al., 2023, Liao et al., 2025, Xing et al., 2025, Zheng et al., 2026] support parallel generation but typically require

many denoising steps, while guided variants [Zhong et al., 2023, Jiang et al., 2023b] further increase cost through per-step gradient guidance. ReflectDrive-2 instead uses masked discrete diffusion, which enables parallel generation in a few rounds and native token-level editing within the same rollout. Unlike prior planners, it couples in-place editing with a shared policy objective and reward signal.

2.2 Discrete Diffusion and Token-Space Editing

Discrete diffusion extends diffusion modeling to categorical spaces [Austin et al., 2021], and masked prediction enables parallel generation through iterative unmasking [Chang et al., 2022]. This paradigm has recently scaled to language modeling, including LLaDA [Nie et al., 2025], Seed Diffusion [Song et al., 2025], MDLM [Lou et al., 2024a], SEDD [Lou et al., 2024b], Block Diffusion [Arriola et al., 2025], Fast-dLLM [Wu et al., 2025], and LLaDA 2.0/2.1 [Bie et al., 2025, 2026], the latter introducing Token-to-Token editing via low-confidence token regeneration. Such re-masking makes discrete diffusion well suited to editable planning, but most existing editing mechanisms rely on decoding heuristics or separately trained refiners. For example, LLaDA 2.1 T2T [Bie et al., 2026] edits by confidence rather than the structured failures encountered in control. By contrast, AutoEdit is trained on structure-aware trajectory perturbations aligned with common driving errors, including longitudinal progress mistakes and lateral heading drift.

Recent refinement-based multimodal diffusion models are also related. DriveFine [Dang et al., 2026] is the closest prior driving work, but trains its refiner separately from the drafter. ReflectDrive-2 instead treats drafting and editing as a single rollout, assigning terminal driving reward to the post-edit trajectory and propagating policy-gradient credit through both stages. "From denoising to refining" [Ji et al., 2025] studies corrective refinement in vision–language diffusion models, but not closed-loop driving or reward-coupled editing. LLaDA-VLA [Wen et al., 2025] applies discrete diffusion to robot control, whereas ReflectDrive-2 focuses on autonomous driving and token-space self-editing under a shared rollout reward.

2.3 Reinforcement Learning for Diffusion Policies

DDPO [Black et al., 2024] and DPPO [Ren et al., 2025] apply policy gradients to continuous diffusion by viewing denoising as a multi-step MDP. For discrete diffusion, d1 [Zhao et al., 2025] uses GRPO-style RL without modeling the full multi-step structure; d2 [Wang et al., 2025b] restores it with step-aware gradients and group-relative advantage; and SPG [Wang et al., 2025a] derives tighter ELBO/EUBO bounds. In driving, HDP [Zheng et al., 2026] and DriveFine [Dang et al., 2026] apply RL to diffusion planners, but only for single-pass generation or refinement. ReflectDrive-2 instead optimizes a composed rollout, draft $\rightarrow$ AutoEdit, so the terminal reward jointly credits both drafting and editing. Increasing diffusion steps alone does not create a distinct edit phase for reward assignment; our formulation does. Section 4.5 formalizes this distinction, and Table 1 shows the resulting gain in editing effectiveness.

3 Preliminaries

3.1 Problem Setting

At time step t, the ego vehicle receives observation $\boldsymbol{o}_t = (\boldsymbol{v}_t, \boldsymbol{\ell}_t, \boldsymbol{s}_t)$, including panoramic visual tokens from three cameras over two frames, navigational instruction tokens encoding route and maneuver cues, and ego-state kinematic tokens. The instruction channel constitutes the "L" in our VLA, conditioning planning on intent in addition to scene context. The objective is to generate a future trajectory $\boldsymbol{\tau} = \{(x_k, y_k)\}_{k=1}^{K}$ that is safe, comfortable, rule-compliant, and consistent with $\boldsymbol{\ell}_t$; heading is derived from consecutive waypoints when required.

3.2 Masked Discrete Diffusion

Forward and reverse process. We encode the future trajectory as BEV coordinate tokens $\mathbf{x}_0$. In masked discrete diffusion [Austin et al., 2021, Nie et al., 2025], each token is independently replaced by `[MASK]` with probability $t \in [0, 1]$ to form $\mathbf{x}_t$, and a bidirectional Transformer p_θ reconstructs $\mathbf{x}_0$ conditioned on context $\mathbf{c} = (\boldsymbol{v}_t, \boldsymbol{\ell}_t, \boldsymbol{s}_t)$. Unlike prior work that supervises only masked positions [Nie

et al., 2025], we train on all positions:

$$\mathcal{L}_{\text{DLM}}(\theta) = -\mathbb{E}_{\mathbf{x}_0,\, t}\left[\frac{1}{L}\sum_{i=1}^{L}\log p_\theta(x_0^i \mid \mathbf{x}_t, \mathbf{c})\right]. \tag{1}$$

This improves stability and draft quality, requiring only a few parallel denoising rounds at inference.

Selective re-generation. Masked diffusion also enables in-place editing: for edit mask $\boldsymbol{e} \in \{0,1\}^L$, we mask the selected tokens and denoise from $t^* \approx |\boldsymbol{e}|/L$. LLaDA 2.1 [Bie et al., 2026] uses this for Token-to-Token editing based on low-confidence tokens. AutoEdit retains the same mechanism but learns the edit operator and couples it with drafting through shared RL reward (Sections 4.3 and 4.5).

3.3 KV Caching for Efficient Inference

Vanilla KV caching is ineffective in masked diffusion because bidirectional attention requires recomputing KV at each denoising step as masked tokens change [Nie et al., 2025]. Prior work mitigates this with blockwise generation and attention [Arriola et al., 2025, Bie et al., 2025, 2026]. We use causal attention for the scene-context prompt and block-wise attention for trajectory tokens, allowing prompt KV reuse while retaining bidirectional diffusion within the trajectory block (Section 5).

3.4 Reinforcement Learning Fine-Tuning

Because supervised learning does not directly optimize driving objectives, we fine-tune trajectory generation as a Markov decision process (MDP) with closed-loop reward. Following Wang et al. [2025b], we optimize $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|o)}[R(\tau)]$ using group-relative advantage over G sampled trajectories and a discrete-diffusion policy gradient:

$$\begin{aligned}\mathcal{L}(\theta) = -\mathbb{E}_{\tau_{1:G}\sim\pi_{\theta_{\text{old}}}(\cdot|o)}\Bigg[\frac{1}{G}\sum_{g=1}^{G}\frac{1}{L}\sum_{s=1}^{S}\sum_{p=1}^{L}\mathbf{1}_{\{x_{g,p}^{s+1}\neq x_{g,p}^{s}\}}\cdot \\ \min\left(r_{g,p}^{s}A_g, \text{clip}(r_{g,p}^{s}, 1-\epsilon, 1+\epsilon)A_g\right)\Bigg] + \lambda_{\text{KL}}D_{\text{KL}}\left(\pi_\theta \,\|\, \pi_{\text{ref}}\right),\end{aligned} \tag{2}$$

where $A_g = R(\tau_g) - \frac{1}{G}\sum_j R(\tau_j)$ and $r_{g,p}^s$ is the token-level importance ratio. The indicator selects updated tokens only. With $S = S_{\text{draft}} + S_{\text{edit}}$, the same reward jointly credits drafting and AutoEdit.

4 Method

4.1 ReflectDrive-2 Overview

ReflectDrive-2 formulates driving as a unified *decision–draft–reflect* process in a shared discrete token space. Given multimodal context $\mathbf{c} = (\boldsymbol{v}_t, \boldsymbol{\ell}_t, \boldsymbol{s}_t)$, it predicts goal-point hypotheses, generates full draft trajectories using a low-confidence remasking policy guided by decision keypoints, and refines them by AutoEdit, which re-masks and rewrites selected trajectory tokens using the same model. Global token-level revisions are only applied in the final reflection step to yield the final trajectory.

Supervised training combines trajectory drafting, perturbation-based correction for longitudinal and lateral planning errors, and a drivable-area-aware field loss. RL then optimizes the full decision-draft-reflect rollout: reward is assigned to the post-edit trajectory, and policy-gradient credit is shared by full rollout, making AutoEdit part of the policy rather than post-processing. The inference pipeline is

$$\mathbf{c} \to \{g_m\}_{m=1}^{N_g} \to \{\boldsymbol{x}_m^{(0)}\}_{m=1}^{N_g} \to \{\boldsymbol{x}_m^{(K)}\}_{m=1}^{N_g}, \tag{3}$$

where g_m is a sampled goal point, $\boldsymbol{x}_m^{(0)}$ the draft, and $\boldsymbol{x}_m^{(K)}$ the edited trajectory. All tokens are processed by a shared backbone, with diffusion applied to the action-token block.

4.2 Goal-Conditioned Masked Trajectory Diffusion

Multimodal context encoding. Two adjacent panoramic frames from three cameras are encoded by a ViT and concatenated with route-instruction and ego-state tokens. A shared backbone processes all tokens, with an action-specific FFN and head for trajectory prediction.

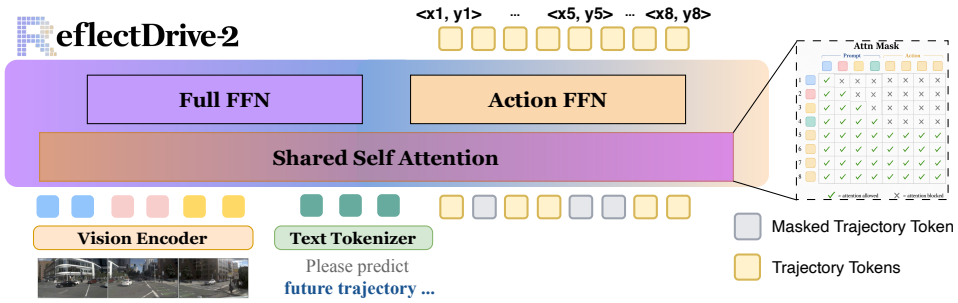

Figure 2: **ReflectDrive-2 architecture.** Vision, language, ego-state, and trajectory tokens share a single self-attention backbone, with a Full FFN over prompt tokens and a compact Action FFN over the action-token block. The attention pattern (right) is causal over the prompt, enabling shared-prefix KV reuse and block-wise bidirectional over action tokens.

Goal-point prediction. Rather than predicting a single endpoint, ReflectDrive-2 models a posterior over discrete BEV goal points. Each sampled goal serves as a behavior hypothesis and conditions a separate drafting branch. Training uses the expert endpoint; inference uses top-k sampling with BEV-space non-maximum suppression (NMS) to preserve distinct maneuver options.

Masked trajectory drafting. We discretize an 8-waypoint future trajectory as

$$\boldsymbol{x}_0 = [x_1, y_1, \ldots, x_8, y_8], \tag{4}$$

where (x_8, y_8) is the selected goal point. Training masks random tokens and applies Eq. (1); inference initializes non-goal tokens as `[MASK]` and fills them in a few parallel denoising rounds via low-confidence remask. This makes computation depend on denoising rounds rather than token length and directly supports later editing.

4.3 AutoEdit Trajectory Correction

AutoEdit is a token-to-token trajectory editor in the same discrete action space as the drafter. Given a concrete trajectory-token sequence, it predicts replacement tokens and commits selected updates directly, without re-masking or an auxiliary refinement network.

Structure-aware perturbations. Given a clean waypoint sequence $\boldsymbol{z}_0$, we synthesize perturbed trajectories $\tilde{\boldsymbol{z}}_0 = \mathcal{T}(\boldsymbol{z}_0)$ before tokenization using two error families: longitudinal progress errors and lateral heading deviations. Longitudinal perturbation rescales trajectory progress,

$$\tilde{\boldsymbol{z}}_i = \mathrm{Interp}(\boldsymbol{z}_0, \beta d_i), \qquad \beta \sim \mathcal{U}(\beta_{\min}, \beta_{\max}), \tag{5}$$

while lateral perturbation rotates the trajectory in the ego frame,

$$\tilde{\boldsymbol{z}}_i = \mathbf{R}(\alpha)\boldsymbol{z}_i, \qquad \mathbf{R}(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}, \qquad \alpha \sim \mathcal{U}(-\alpha_{\max}, \alpha_{\max}). \tag{6}$$

After tokenization to $\tilde{\boldsymbol{x}}_0$, AutoEdit is trained to recover the clean sequence:

$$q_\theta(\cdot \mid \tilde{\boldsymbol{x}}_0, \mathbf{c}) = \mathrm{softmax}\left(h_\theta(\tilde{\boldsymbol{x}}_0, \mathbf{c})\right), \tag{7}$$

$$\mathcal{L}_{\mathrm{SAP}}(\theta) = -\mathbb{E}_{\boldsymbol{x}_0, \mathcal{T}}\left[\frac{1}{L}\sum_{i=1}^{L} \log q_\theta(x_0^i \mid \tilde{\boldsymbol{x}}_0, \mathbf{c})\right]. \tag{8}$$

Inference-time AutoEdit. From draft $\boldsymbol{x}^{(0)}$, AutoEdit performs K rounds of direct token rewriting:

$$\boldsymbol{x}^{(k+1)} = \mathrm{T2TEdit}_\theta\left(\boldsymbol{x}^{(k)}, \mathbf{c}\right). \tag{9}$$

All tokens are replaced with the AutoEdit step model outcomes.

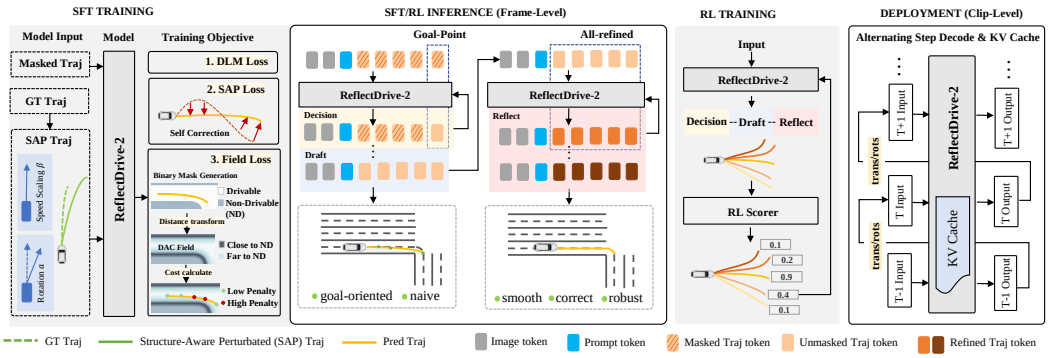

Figure 3: **Training, inference, and deployment of ReflectDrive-2. (a)** SFT with three objectives: masked-token loss $\mathcal{L}_{\text{DLM}}$, structure-aware perturbation loss $\mathcal{L}_{\text{SAP}}$ over longitudinal scaling β and lateral rotation α, and drivable-area field loss $\mathcal{L}_{\text{field}}$. **(b)** Frame-level *decision–draft–reflect* inference: commit a goal token, parallel-decode the draft, then rewrite tokens in place via AutoEdit. **(c)** RL fine-tuning scores full decision–draft–reflect rollouts with a closed-loop reward, propagating group-relative credit through both phases. **(d)** Clip-level deployment alternates full-step and lite-step frames with shared-prefix KV reuse; rigid-body transforms carry the previous plan into the current ego frame.

4.4 Constraint-Aware Supervised Objectives

Drivable-area field loss. Because $\mathcal{L}_{\text{DLM}}$ and $\mathcal{L}_{\text{SAP}}$ do not explicitly enforce drivable-area geometry, we add a field loss over the waypoint distribution induced by coordinate-token logits. At each waypoint, the model defines a spatial probability distribution over the BEV grid, and we penalize probability mass assigned to high-cost cells in a drivable-area compliance field. This regularizer discourages confident predictions in non-drivable regions while still allowing soft uncertainty near boundaries. The detailed formulation of the waypoint distribution, the field loss, and the drivable-area compliance field is provided in Section A.

Total supervised objective. The full supervised objective combines masked trajectory generation, structure-aware correction, and drivable-area regularization:

$$\mathcal{L}_{\text{sup}} = \mathcal{L}_{\text{DLM}} + \lambda_{\text{SAP}}\mathcal{L}_{\text{SAP}} + \lambda_{\text{field}}\mathcal{L}_{\text{field}}. \tag{10}$$

4.5 Reinforcement Learning over decision–draft–reflect Rollouts

We fine-tune ReflectDrive-2 with RL over a composed *decision–draft–reflect* rollout, rather than a single-pass diffusion process. For each scene, we sample multiple goal-conditioned candidates, execute drafting followed by AutoEdit, and assign terminal reward to the final post-edit trajectory. Policy-gradient credit is then applied to token transitions from both phases under one rollout objective. This design jointly improves draft generation and the AutoEdit refinement built upon the draft under the same closed-loop objective. The full rollout notation, terminal reward definition, group-relative advantage, and transition indicator are described in Section A.

5 Efficient Inference for Reflective Masked Planning

We optimize deployment as a reflection-aware serving stack rather than a collection of isolated tricks. The final system runs the full decision–draft–reflect pipeline on full-step frames and a lightweight temporal AutoEdit pass on lite-step frames, achieving an average latency of 31.8 ms per frame on NVIDIA Thor. Detailed optimization breakdowns and comprehensive deployment quality-gate validation are both provided in Section B.

Shared-prefix KV reuse. Goal proposal, drafting, and AutoEdit share the same visual, route-instruction, and ego-state prefix. We therefore cache this prefix once and reuse it across phases, reducing redundant attention computation.

Mutable action-cache rewinding. Because drafting and AutoEdit both modify the action-token block, its KV states become stale after each update. We rewind the cache to the shared-prefix boundary and recompute only the mutable action block; at block boundaries, cache rewrite is merged with the next token-update step to reduce overhead.

Action-expert FFN. Trajectory decoding uses a small action vocabulary and fixed-length token block, so we replace the full action-branch FFN with a compact expert FFN. This substantially reduces latency while preserving planning quality; detailed trajectory-level quality gates are reported in Section B.

Fused on-device token update. Drafting and AutoEdit both require token scoring, selection, and update. We fuse these operations into an on-device CUDA kernel, removing CPU-device synchronization and reducing per-step update cost.

Alternating Step Decode. Adjacent driving frames share substantial context and plan structure, so we alternate full-step and lite-step inference. Full-step frames run the complete decision–draft–reflect pipeline, while lite-step frames transform the previous plan into the current ego frame and refine it with a short AutoEdit pass:

$$\tilde{\tau}^{(t)} = \mathcal{T}_{t-1\to t}\Big(\mathrm{Shift}\left(\hat{\tau}^{(t-1)}\right)\Big), \qquad \boldsymbol{x}^{(t,0)} = \mathrm{Tok}\Big(\tilde{\tau}^{(t)}\Big), \qquad \hat{\boldsymbol{x}}^{(t)} = \mathrm{T2TEdit}_\theta^{S'}\Big(\boldsymbol{x}^{(t,0)}, \mathbf{c}^{(t)}\Big). \tag{11}$$

This reuses AutoEdit as an inter-frame temporal refiner with minimal quality loss; detailed deployment-benchmark comparisons against running the full pipeline on every frame are given in Section B. The resulting planner runs at 31.8 ms per frame on NVIDIA Thor, with 45.0 ms on full-step frames and 18.6 ms on lite-step frames.

6 Experiments

6.1 Experimental Setup

Dataset and metrics. We evaluate on NAVSIM [Dauner et al., 2024], a nuPlan-based [Caesar et al., 2021] closed-loop planning benchmark. The task is to predict a 4-second ego trajectory at 2 Hz. We train on navtrain ($1{,}192$ scenes) and evaluate on navtest (136 scenes). Performance is measured by PDMS, which aggregates no at-fault collision (NC), drivable-area compliance (DAC), time to collision (TTC), comfort, and ego progress (EP); its exact definition is given in Section E.

Implementation. ReflectDrive-2 uses a 0.7B masked-diffusion language backbone and a 0.1B ViT visual encoder, both initialized from proprietary pretrained weights and fully fine-tuned on NAVSIM. Inputs include two temporal frames from the left-front/front/right-front cameras, navigation instruction, and ego-state tokens; outputs are 8 waypoints represented by 16 discrete coordinate tokens. We first perform supervised fine-tuning, then reinforcement fine-tuning with PDMS reward.

Evaluation protocol. For standard evaluation, ReflectDrive-2 outputs a single plan by selecting the most confident goal after NMS. For best-of-N, multiple goals are sampled to yield the oracle-optimal closed-loop trajectory.

6.2 Effect of RL on Inference-Time AutoEdit

Table 1 isolates the key interaction behind ReflectDrive-2. Before RL, inference-time AutoEdit yields only marginal gains (at most $+0.3$ PDMS), even when trained with structure-aware perturbation. After RL over the full draft-and-edit rollout, the same inference-time AutoEdit improves PDMS by $+1.9$. As discussed in Section 4.5, the shared terminal reward couples the two stages: full-rollout RL jointly improves draft generation and the AutoEdit refinement built upon it. This effect depends on composed draft-and-edit rollouts and does not arise in single-pass RL formulations (Section 2.3). This interaction raises ReflectDrive-2's own PDMS from 89.1 without inference-time AutoEdit to 91.0 with AutoEdit. The final model outperforms ReCogDrive (90.8), the strongest camera-only VLA baseline in Table 2, by $+0.2$ PDMS.

Table 1: **Effect of inference-time AutoEdit across training regimes.** After supervised training, inference-time AutoEdit contributes at most $+0.3$ PDMS. After RL over the full draft-and-edit rollout, the same inference-time AutoEdit contributes $+1.9$ PDMS.

Training setting	w/o AutoEdit	w/ AutoEdit	Δ PDMS
DLM + DACF + AutoEdit training	87.7	88.0	+0.3
DLM + DACF + AutoEdit training + RL	89.1	**91.0**	**+1.9**

Table 2: **Closed-loop results on NAVSIM.** All methods are evaluated under standard evaluation unless marked otherwise. "C & L" denotes camera and LiDAR. † indicates best-of-6 evaluation.

Method	Input	NC↑	DAC↑	TTC↑	Comf.↑	EP↑	PDMS↑
End-to-End Methods							
UniAD [Hu et al., 2023]	Cam	97.8	91.9	92.9	100.0	78.8	83.4
TransFuser [Chitta et al., 2022]	C & L	97.7	92.8	92.8	100.0	79.2	84.0
Hydra-MDP [Li et al., 2024]	C & L	98.3	96.0	94.6	100.0	78.7	86.5
DiffusionDrive [Liao et al., 2025]	C & L	98.2	96.2	94.7	100.0	82.2	88.1
GoalFlow [Xing et al., 2025]	C & L	98.4	98.3	94.6	100.0	85.0	90.3
Camera-Only VLA Planners							
AutoVLA [Zhou et al., 2025]	Cam	98.4	95.6	98.0	99.9	81.9	89.1
DriveVLA-W0 [Li et al., 2025a]	Cam	98.7	99.1	95.3	99.3	83.3	90.2
ReCogDrive [Li et al., 2025b]	Cam	97.9	97.3	94.9	100.0	87.3	90.8
ReflectDrive-2 (Ours)	Cam	97.3	98.1	92.5	100.0	**89.4**	**91.0**
ReflectDrive-2†	Cam	98.5	99.2	95.5	99.8	**93.8**	**94.8**
Human (NAVSIM reference)	–	100.0	100.0	100.0	99.9	87.5	**94.8**

6.3 Closed-Loop Driving Performance

Table 2 reports NAVSIM results. Under the standard single-trajectory setting, ReflectDrive-2 achieves 91.0 PDMS with camera-only input, outperforming ReCogDrive (90.8), the strongest camera-only VLA baseline, and GoalFlow (90.3), which uses both camera and LiDAR. The gain is driven mainly by ego progress: ReflectDrive-2 attains the best EP (89.4) among all listed methods while maintaining strong DAC (98.1) and saturated comfort (100.0). Although NC and TTC are not the highest, the overall result indicates a favorable progress–constraint trade-off. The most relevant comparison is against camera-only VLA baselines, over which ReflectDrive-2 improves PDMS by $+0.2$ to $+1.9$, consistent with the rollout-level RL interaction isolated in Table 1 (more results in Section F).

The same table also reports a best-of-6 oracle analysis (†) to probe the learned goal-point posterior. Under oracle selection, ReflectDrive-2 reaches 94.8 PDMS, matching the NAVSIM human reference. The 3.8 PDMS gap over single-trajectory inference indicates substantial recoverable headroom from behavior-hypothesis selection, suggesting that the goal head captures meaningful scene-dependent multimodality rather than near-duplicate endpoint samples.

And Table 3 isolates the effect of each training component without inference-time AutoEdit. Field loss (DACF) contributes $+2.4$ PDMS ($84.8 \rightarrow 87.2$) mainly through DAC. AutoEdit supervised training adds $+0.5$ more mainly from longitudinal perturbations. RL over the full rollout brings EP from 82.2 to 89.3 and PDMS to 89.1; combined with inference-time AutoEdit, the final score is 91.0 (Table 1). Other inference hyperparameters analysis is described in Section D.

6.4 Decision Diversity and Reflection

Goal points. Figure 4 illustrates multi-goal inference. In turning scenes (the two left subfigures), diverse goals yield distinct curved paths, with certain candidates better adhering to drivable boundaries. In interaction scenarios (the two right subfigures), the model exhibits longitudinal and lateral adaptive behaviors amid surrounding agents, including lane keeping, lane changing and speed adjustment. These goal points are not sampling artifacts, but explicit behavioral hypotheses decoded for downstream planning.

Table 3: **Effect of training components.** All rows are evaluated without inference-time AutoEdit to isolate the training signal.

Training objective	NC↑	DAC↑	TTC↑	Comf.↑	EP↑	PDMS↑
DLM	97.5	93.9	92.8	99.5	79.5	84.8
DLM + DACF	97.4	97.0	93.1	99.9	81.4	87.2
DLM + DACF + AutoEdit training	**97.8**	96.7	**93.6**	99.9	82.2	87.7
DLM + DACF + AutoEdit training + RL	96.3	**97.9**	88.9	99.7	**89.3**	**89.1**

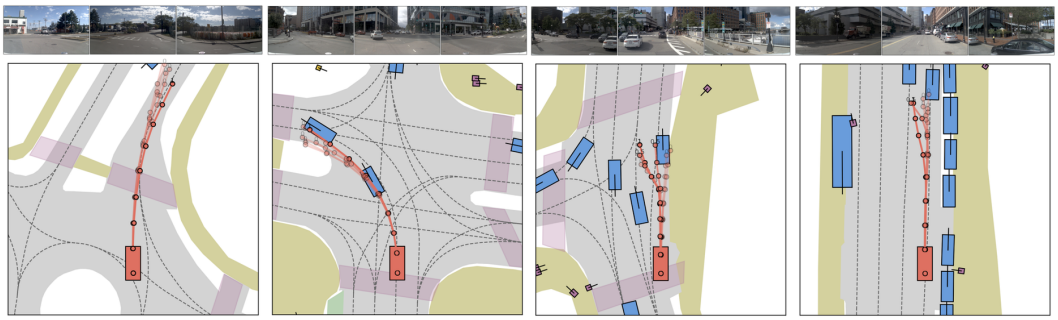

Figure 4: **Decision diversity from goal points.** Each goal anchors a distinct behavior hypothesis. Trajectory opacity encodes PDMS.

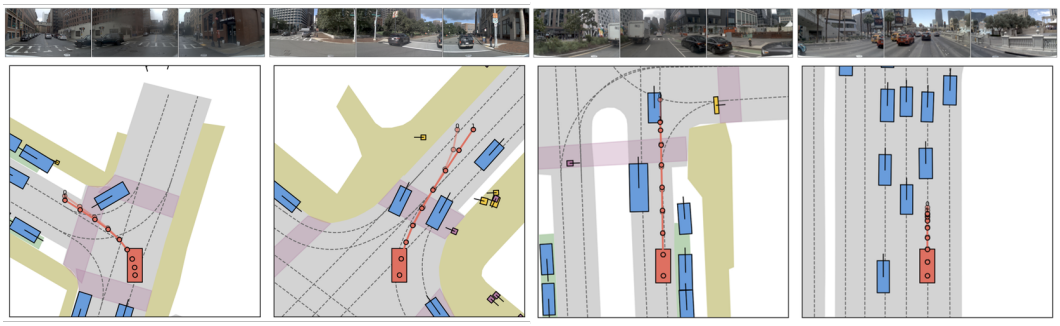

Figure 5: **Inference-time reflection with AutoEdit.** Semi-transparent: initial drafts. Solid: final outputs after AutoEdit. PDMS annotated at trajectory endpoints.

Reflection with AutoEdit. Figure 5 shows AutoEdit at inference. Semi-transparent curves are initial drafts; solid curves are post-AutoEdit. In the top row, AutoEdit pulls trajectories back into the drivable area; in the bottom row, it adjusts the plan around nearby agents. The revisions are structured rewrites in the same token space used for generation, not cosmetic smoothing.

7 Limitations & Future Work

We presented *ReflectDrive-2*, a reflective VLA planner that reframes autonomous driving as a joint process of decision–draft–reflect, all within a shared discrete token space. The central finding of this work is that self-correction in driving planning requires more than a trained editor. Applying reinforcement learning over the complete rollout changes this. This interaction raises the AutoEdit gain from $+0.3$ to $+1.9$ PDMS and is the primary driver of ReflectDrive-2's 91.0 PDMS on NAVSIM with camera-only input. We further showed that the decision–draft–reflect structure defines not only a modeling paradigm but also an efficient runtime with 31.8 ms per frame on NVIDIA Thor. But the current open-loop reinforcement learning lacks the ability to adjust actions based on environmental feedback, leading to error accumulation and poor robustness to perturbations, and the RL reward is a lightweight proxy for real-world driving objectives. As future work, we plan to extend the current open-loop setting to closed-loop reinforcement learning, investigate how world models can be incorporated into the algorithm to improve the model's bias correction capability.

Acknowledgements and Disclosures

Funding transparency statement: No funding was received for this work.

Competing interests: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025. URL `https://arxiv.org/abs/2503.09573`.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, pages 17981–17993, 2021.

Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst. In *Robotics: Science and Systems*, 2019.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Ling Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. LLaDA2.0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, Yihong Zhuang, et al. LLaDA2.1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *International Conference on Learning Representations*, 2024.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex H. Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. nuPlan: A closed-loop ML-based planning benchmark for autonomous vehicles. *arXiv preprint arXiv:2106.11810*, 2021.

Wei Cao, Marcel Hallgarten, Tianyu Li, Daniel Dauner, Xunjiang Gu, Caojun Wang, Yakov Miron, Marco Aiello, Hongyang Li, Igor Gilitschenski, Boris Ivanovic, Marco Pavone, Andreas Geiger, and Kashyap Chitta. Pseudo-simulation for autonomous driving. In *Conference on Robot Learning*, 2025.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. MaskGIT: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11315–11325, 2022.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023.

Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. TransFuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *IEEE International Conference on Robotics and Automation*, pages 4693–4700, 2018.

Chenxu Dang, Sining Ang, Yongkang Li, Haochen Tian, Jie Wang, Guang Li, Hangjun Ye, Jie Ma, Long Chen, and Yan Wang. DriveFine: Refining-augmented masked diffusion VLA for precise and robust driving. *arXiv preprint arXiv:2602.14577*, 2026.

Daniel Dauner, Matthias Hallgarten, Tianyu Li, Xinshuo Weng, Zhiyu Huang, Zetong Yang, Hongyang Li, Igor Gilitschenski, Boris Ivanovic, Marco Pavone, Andreas Geiger, and Kashyap Chitta. NAVSIM: Data-driven non-reactive autonomous vehicle simulation and benchmarking. In *Advances in Neural Information Processing Systems*, 2024.

Haoyang Fan, Fan Zhu, Changchun Liu, Liangliang Zhang, Li Zhuang, Dong Li, Weichuan Zhu, Jiangtao Hu, Hongye Li, and Qi Kong. Baidu Apollo EM motion planner. *arXiv preprint arXiv:1807.08048*, 2018.

Xiaoxin Feng, Ziyan Gao, Yuheng Kan, and Wei Wu. SMART: Scalable multi-agent real-time motion generation via next-token prediction. *arXiv preprint arXiv:2405.15677*, 2024.

Shengchao Hu, Li Chen, Penghao Wu, Hongyang Li, Junchi Yan, and Dacheng Tao. ST-P3: End-to-end vision-based autonomous driving via spatial-temporal feature learning. In *European Conference on Computer Vision*, pages 533–549, 2022.

Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862, 2023.

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, pages 9902–9915, 2022.

Yatai Ji, Teng Wang, Yuying Ge, Zhiheng Liu, Sidi Yang, Ying Shan, and Ping Luo. From denoising to refining: A corrective framework for vision-language diffusion model. *arXiv preprint arXiv:2510.19871*, 2025.

Bo Jiang, Shaoyu Chen, Qing Xu, Bencheng Liao, Jiajie Chen, Helong Zhou, Qian Zhang, Wenyu Liu, Chang Huang, and Xinggang Wang. VAD: Vectorized scene representation for efficient autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8340–8350, 2023a.

Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, and Dragomir Anguelov. MotionDiffuser: Controllable multi-agent motion prediction using diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9644–9653, 2023b.

Shinpei Kato, Shinpei Tokunaga, Yuya Maruyama, Shigeki Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *ACM/IEEE International Conference on Cyber-Physical Systems*, pages 287–296, 2018.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Yingyan Li, Shuyao Shang, Weisong Liu, Bing Zhan, Haochen Wang, Yuqi Wang, Yuntao Chen, Xiaoman Wang, Yasong An, Chufeng Tang, Lu Hou, Lue Fan, and Zhaoxiang Zhang. DriveVLA-W0: World models amplify data scaling law in autonomous driving. *arXiv preprint arXiv:2510.12796*, 2025a.

Yongkang Li, Kaixin Xiong, Xiangyu Guo, Fang Li, Sixu Yan, Gangwei Xu, Lijun Zhou, Long Chen, Haiyang Sun, Bing Wang, Guang Chen, Hangjun Ye, Wenyu Liu, and Xinggang Wang. ReCogDrive: A reinforced cognitive framework for end-to-end autonomous driving. *arXiv preprint arXiv:2506.08052*, 2025b.

Zhenxin Li, Kailin Li, Shihao Wang, Shiyi Lan, Zhiding Yu, Yishen Ji, Zhiqi Li, Ziyue Zhu, Jan Kautz, Zuxuan Wu, Yu-Gang Jiang, and José M. Álvarez. Hydra-MDP: End-to-end multimodal planning with multi-target hydra-distillation. *arXiv preprint arXiv:2406.06978*, 2024.

Bencheng Liao, Shaoyu Chen, Haoran Yin, Bo Jiang, Cheng Wang, Sixu Yan, Xinbang Zhang, Xiangyu Li, Ying Zhang, Qian Zhang, and Xinggang Wang. DiffusionDrive: Truncated diffusion model for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12037–12047, 2025. doi: 10.1109/CVPR52734.2025.01124.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*, 2024a.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*, 2024b. Also referred to as Score Entropy Discrete Diffusion.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Allen Z. Ren, Justin Lidard, Lars L. Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. In *International Conference on Learning Representations*, 2025. URL `https://arxiv.org/abs/2409.00588`.

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chunguang Xie, Ping Luo, Andreas Geiger, and Hongyang Li. DriveLM: Driving with graph visual question answering. In *European Conference on Computer Vision*, 2024.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Yonghui Wu, and Hao Zhou. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Chenxu Hu, Yang Wang, Kun Zhan, Peng Jia, Xianpeng Lang, and Hang Zhao. DriveVLM: The convergence of autonomous driving and large vision-language models. In *Conference on Robot Learning*, 2024.

Chenyu Wang, Paria Rashidinejad, DiJia Su, Song Jiang, Sid Wang, Siyan Zhao, Cai Zhou, Shannon Zejiang Shen, Feiyu Chen, Tommi S. Jaakkola, Yuandong Tian, and Bo Liu. SPG: Sandwiched policy gradient for masked diffusion language models. *arXiv preprint arXiv:2510.09541*, 2025a.

Guanghan Wang, Yair Schiff, Gilad Turok, and Volodymyr Kuleshov. d2: Improved techniques for training reasoning diffusion language models. *arXiv preprint arXiv:2509.21474*, 2025b.

Yuqing Wen, Hebei Li, Kefan Gu, Yucheng Zhao, Tiancai Wang, and Xiaoyan Sun. LLaDA-VLA: Vision language diffusion action models. *arXiv preprint arXiv:2509.06932*, 2025.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Zebin Xing, Xingyu Zhang, Yang Hu, Bo Jiang, Tong He, Qian Zhang, Xiaoxiao Long, and Wei Yin. GoalFlow: Goal-driven flow matching for multimodal trajectories generation in end-to-end autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1602–1611, 2025.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.

Yinan Zheng, Tianyi Tan, Bin Huang, Enguang Liu, Ruiming Liang, Jianlin Zhang, Jianwei Cui, Guang Chen, Kun Ma, Hangjun Ye, Long Chen, Ya-Qin Zhang, Xianyuan Zhan, and Jingjing Liu. Unleashing the potential of diffusion models for end-to-end autonomous driving. *arXiv preprint arXiv:2602.22801*, 2026.

Ziyuan Zhong, Davis Rempe, Yuxiao Chen, Boris Ivanovic, Yulong Cao, Danfei Xu, Marco Pavone, and Baishakhi Ray. Language-guided traffic simulation via scene-level diffusion. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 144–177. PMLR, 2023. URL `https://arxiv.org/abs/2306.06344`.

Zewei Zhou, Tianhui Cai, Seth Z. Zhao, Yun Zhang, Zhiyu Huang, Bolei Zhou, and Jiaqi Ma. AutoVLA: A vision-language-action model for end-to-end autonomous driving with adaptive reasoning and reinforcement fine-tuning. In *Advances in Neural Information Processing Systems*, 2025. URL `https://arxiv.org/abs/2506.13757`.

A Detailed Objective Formulations

This appendix provides the detailed formulations omitted from the main text for brevity. We expand two components: the drivable-area field loss used in supervised training, and the RL formulation for draft-and-edit rollouts. In both cases, the appendix contains not only the equations but also the motivation for why these constructions are useful.

A.1 Drivable-Area Field Loss

The supervised objectives $\mathcal{L}_{\mathrm{DLM}}$ and $\mathcal{L}_{\mathrm{SAP}}$ operate at the token level. They supervise trajectory generation and structured correction, but they do not explicitly encode drivable-area geometry in BEV space. As a result, a model can achieve good token-level likelihood while still assigning substantial probability mass to off-road regions. To inject spatial geometric bias, we add a field-based regularizer over the waypoint distribution induced by the coordinate-token logits.

Waypoint distribution induced by coordinate logits. At waypoint t, the model predicts discrete marginal distributions over the x- and y-coordinate vocabularies, denoted by $p_x^{(t)}$ and $p_y^{(t)}$. These marginals induce a spatial distribution over the BEV grid:

$$p_{xy}^{(t)}[i,j] = p_x^{(t)}[i]\, p_y^{(t)}[j]. \tag{12}$$

This factorized form converts token-level uncertainty into a dense probability map over spatial cells. It is also computationally convenient, since it avoids introducing a separate joint spatial head while still allowing field-based supervision in BEV.

Field-weighted penalty. Given a nonnegative BEV cost field $\mathcal{C} \in \mathbb{R}_{\geq 0}^{H\times W}$, we penalize probability mass assigned to high-cost cells using

$$\mathcal{L}_{\mathrm{field}} = \sum_{t=1}^{T} \sum_{i,j} -\log\Big(1 - p_{xy}^{(t)}[i,j]\Big)\, \mathcal{C}[i,j]. \tag{13}$$

The logarithmic barrier is weak when the assigned probability is small, but grows rapidly as $p_{xy}^{(t)}[i,j]$ approaches 1. This behavior is desirable: the model is not heavily penalized for small uncertainty mass near undesirable cells, but it is strongly discouraged from making confident off-road predictions.

Drivable-area compliance field. In our implementation, $\mathcal{C}$ is instantiated as a drivable-area compliance field. Let $\boldsymbol{b} \in \{0,1\}^{H\times W}$ be the drivable-area indicator, where $\boldsymbol{b}[i,j] = 1$ means cell (i,j) is drivable. We first define the outside distance

$$d_{\mathrm{out}}[i,j] = \begin{cases} 0, & \boldsymbol{b}[i,j] = 1, \\ r_{\mathrm{DAC}} \min\limits_{(u,v):\boldsymbol{b}[u,v]=1} \sqrt{(i-u)^2 + (j-v)^2}, & \boldsymbol{b}[i,j] = 0, \end{cases} \tag{14}$$

which measures how far a non-drivable cell is from the nearest drivable cell. The factor r_{DAC} converts grid distance into metric distance.

We then define the DAC cost field as

$$\mathcal{C}_{\mathrm{DAC}}[i,j] = \max(0, d_{\mathrm{out}}[i,j] - \epsilon_{\mathrm{safe}}), \tag{15}$$

where ϵ_{safe} specifies a tolerance band near the drivable-area boundary. This design gives zero cost inside the drivable region and also avoids over-penalizing cells immediately outside the boundary, which can be sensitive to map discretization, label noise, or coordinate quantization. Cells farther away from the drivable area receive larger penalties.

Interpretation. This field construction imposes a soft geometric prior rather than a hard feasibility constraint. It encourages the model to place probability mass in spatially valid regions, but still permits uncertainty near boundaries when the scene is ambiguous. Consequently, the field loss complements rather than replaces token-level supervision: $\mathcal{L}_{\mathrm{DLM}}$ teaches trajectory generation, $\mathcal{L}_{\mathrm{SAP}}$ teaches structured correction, and $\mathcal{L}_{\mathrm{field}}$ promotes compliance with drivable-area geometry.

A.2 RL over Draft-and-Edit Rollouts

The main text summarizes our RL stage as optimization over a composed draft-and-edit rollout. Here we formalize that process. The key idea is to assign reward to the final post-edit trajectory, while propagating policy-gradient signal to token transitions from both the drafting phase and the AutoEdit phase.

Why optimize the composed rollout. A single final trajectory in ReflectDrive-2 is produced in two stages: drafting first proposes a trajectory, and AutoEdit then revises it. These two stages are tightly coupled. The closed-loop score is assigned to the final post-edit trajectory, and is therefore determined jointly by the draft and the edits applied to it. In particular, AutoEdit should not merely reduce token-level uncertainty; it should make edits that improve the final closed-loop driving score. This motivates treating the two stages as one unified rollout for RL.

Candidate generation. For each scene, we sample N_g goal-conditioned starting points for planning and draw I drafts per goal, giving a total of $G = N_g I$ candidate rollouts. For candidate g, the token-state sequence is

$$\rho_g = \left(\boldsymbol{x}_g^0, \boldsymbol{x}_g^1, \ldots, \boldsymbol{x}_g^{S_{\text{draft}}+S_{\text{edit}}}\right), \tag{16}$$

where the first S_{draft} transitions correspond to drafting and the remaining S_{edit} transitions correspond to AutoEdit. The final trajectory is obtained by detokenizing the last state:

$$\tau_g = \text{Detok}\left(\boldsymbol{x}_g^{S_{\text{draft}}+S_{\text{edit}}}\right). \tag{17}$$

Terminal reward and group-relative advantage. We evaluate the final post-edit trajectory using a closed-loop planning score $R(\tau_g)$. Because the absolute scale of this score can vary across scenes, we use a group-relative baseline defined over the G candidates sampled from the same scene:

$$A_g = R(\tau_g) - \frac{1}{G}\sum_{j=1}^{G} R(\tau_j). \tag{18}$$

This advantage is positive when candidate g outperforms the within-scene average, and negative otherwise. Using a group-relative advantage reduces scene-level variance and emphasizes ranking among competing candidates under the same context.

Credit assignment over token transitions. The discrete-diffusion policy-gradient objective in Eq. (2) is applied to all token transitions in the composed rollout. To indicate whether token position p changes at step s, we define

$$\delta_{g,p}^s = \mathbf{1}_{\{x_{g,p}^{s+1} \neq x_{g,p}^s\}}, \qquad s = 0, \ldots, S_{\text{draft}} + S_{\text{edit}} - 1. \tag{19}$$

This unified transition indicator covers both token unmasking during drafting and token rewriting during AutoEdit. As a result, the same terminal reward is propagated through both phases under one objective.

Interpretation. Because reward is assigned only to the final post-edit trajectory, the drafting phase is optimized not only for plausible trajectories but jointly with AutoEdit for the final closed-loop outcome. Likewise, AutoEdit is optimized for corrections that improve the final closed-loop outcome rather than merely making local token predictions more likely. This is the main advantage of RL over the full draft-and-edit rollout: it aligns both stages with the same end-task metric.

B Supplementary Deployment Analysis

This appendix collectively presents supplementary deployment results, including the detailed inference optimization latency chain and comprehensive quality-gate evaluations. The latency breakdown quantifies individual inference optimizations on NVIDIA Thor, while quality validations verify that efficiency gains do not introduce noticeable planning degradation.

Table 4: **Inference optimization chain on NVIDIA Thor.** All latencies in ms, measured at the natural granularity of each optimization (per attention call, per FFN block, per unmask step, per decode stage). The final row reports the deployed end-to-end planner latency per frame. Local speedups should not be multiplied across rows.

Optimization	Latency: before → after	Speedup
Shared-prefix KV reuse	$0.28 \rightarrow 0.08$	$3.5\times$
Cache rewind & merged rewrite	$14.7 \rightarrow 11.5$	$1.28\times$
Action-expert FFN (D_{ff}: $4096 \rightarrow 1024$)	$2.47 \rightarrow 0.95$	$2.6\times$
Fused CUDA unmasking	$0.45 \rightarrow 0.06$	$7.5\times$
ASD temporal AutoEdit	$26.2 \rightarrow 7.6$	$3.4\times$
End-to-end planner (average frame)	$45.0 \rightarrow \mathbf{31.8}$	$\mathbf{1.42}\times$

B.1 Inference Latency Optimization Chain

All latency metrics are reported in milliseconds and measured at the natural granularity of each module optimization, including attention computation, FFN blocks, unmask operations, and decoding stages. Note that local speedup ratios are non-cumulative and should not be multiplied across individual improvements. Table 4 summarizes individual optimization gains as well as the overall end-to-end planner latency before and after full deployment.

B.2 Deployment Quality Gates

Most serving optimizations adopted in this work are semantics-preserving hardware and implementation-level improvements. Two core optimizations—the compact action-expert FFN and alternating step decode (ASD)—directly modify planner computation and thus require rigorous quality verification to guarantee reliable real-world performance.

Why quality gates are needed. Most serving optimizations in Section 5 are semantics-preserving implementation improvements, such as shared-prefix KV reuse, cache rewinding, and fused on-device token updates. In contrast, two optimizations modify the effective planner computation itself. First, the action-expert FFN replaces the original action-branch FFN with a smaller expert specialized to the compact trajectory-token vocabulary. Second, ASD changes the temporal execution pattern by alternating between full decision–draft–reflect inference and a short temporal AutoEdit pass on adjacent frames. These two changes therefore require explicit quality validation.

Action-expert FFN quality gate. Trajectory-token decoding operates over a constrained action vocabulary and a short fixed-length token block, which makes the original large FFN in the action branch unnecessarily expensive at inference time. We therefore reduce the action-branch FFN hidden dimension from 4096 to 1024 and evaluate the impact on trajectory quality. Results are reported in Table 5(b).

The compact action branch improves most reported trajectory-quality metrics. In particular, it reduces meanSADE from 1.710 to 1.539, improves path-meanFDE across all horizons, and improves path-minADE at 20, 40, and 80 steps. The only regression is a small increase in minSADE from 0.858 to 0.914. Overall, these results indicate that the action branch can be significantly compressed without harming, and in several metrics even improving, planning quality. This supports the use of the action-expert FFN in the deployed system.

ASD temporal AutoEdit quality gate. In streaming driving, adjacent frames share substantial scene context and often admit highly similar future plans. ASD exploits this redundancy by alternating between two execution modes: full-step frames run the complete decision–draft–reflect pipeline, while lite-step frames reuse the previous trajectory after temporal shifting and ego-frame transformation, followed by a short token-to-token AutoEdit refinement. Formally, the lite-step update is

$$\tilde{\tau}^{(t)} = \mathcal{T}_{t-1\rightarrow t}\Big(\text{Shift}\left(\hat{\tau}^{(t-1)}\right)\Big), \qquad \boldsymbol{x}^{(t,0)} = \text{Tok}\Big(\tilde{\tau}^{(t)}\Big), \qquad \hat{\boldsymbol{x}}^{(t)} = \text{T2TEdit}_{\theta}^{S'}\Big(\boldsymbol{x}^{(t,0)}, \mathbf{c}^{(t)}\Big), \tag{20}$$

Table 5: **Quality gates for deployment optimizations.** Δ denotes optimized $-$ baseline. Green marks the desired direction; red marks regressions. For driving scores (↑) higher is better; for trajectory errors (↓) lower is better.

(a) ASD temporal AutoEdit on the in-house deployment benchmark (↑)

Setting	NC	DAC	TTC	Comf.	EP	Overall
Full pipeline (baseline)	79.51	76.99	96.27	88.52	75.13	83.70
ASD	79.19	77.21	96.20	87.88	75.01	83.51
Δ	-0.31	$+0.22$	-0.07	-0.63	-0.12	-0.20

(b) Action-expert FFN trajectory quality (↓)

	Trajectory-level		path-meanFDE			path-minADE		
Setting	**minSADE**	**meanSADE**	**@20**	**@40**	**@80**	**@20**	**@40**	**@80**
Original FFN (baseline)	0.858	1.710	0.167	0.427	0.991	0.088	0.210	0.462
Action-expert FFN	0.914	1.539	0.145	0.346	0.910	0.084	0.177	0.406
Δ	$+0.056$	-0.171	-0.022	-0.081	-0.081	-0.004	-0.033	-0.056

where $\mathrm{Shift}(\cdot)$ removes the elapsed portion of the previous plan and $\mathcal{T}_{t-1\to t}$ maps the remaining waypoints from the previous ego frame to the current ego frame.

In our deployment setting, full-step frames use $S = 1 + 3 + 3$ decision–draft–reflect steps, while lite-step frames use a shorter $S' = 1 + 1$ draft–reflect schedule. We compare ASD against a baseline that runs the full pipeline on every frame. The results are shown in Table 5(a). ASD changes the overall in-house deployment-benchmark score from 83.70 to 83.51, corresponding to only a -0.20 difference. Among the sub-metrics, drivable area compliance slightly improves ($+0.22$), while the remaining metrics show small regressions: NC -0.31, TTC -0.07, comfort -0.63, and EP -0.12. Slight fluctuations across safety metrics are inevitable in our internal closed-loop simulation system for three reasons: first, our simulator contains inherent randomness during runtime; second, all evaluation metrics are bounded by predefined threshold limits; third, temporal data playback mechanisms in the simulation pipeline also introduce minor result discrepancies across repeated runs. The slight declines on NC (-0.31) and TTC (-0.07) fall within the normal range of statistical variation caused by the above factors. Therefore, we judged these trivial regressions would not exert substantial negative impacts on overall model performance. Taken together, these results indicate that ASD preserves the planning quality of the full pipeline to a very close degree, while substantially reducing runtime.

Interpretation. These quality gates support the deployment recipe used in the main text. The action-expert FFN is not merely a latency optimization, but a compression of an over-parameterized action branch that remains well matched to the trajectory-token prediction task. Likewise, ASD reuses the same AutoEdit operator already employed for intra-frame refinement and extends it into a temporally aware inter-frame refiner. This makes the final serving stack efficient not only because each component is individually accelerated, but also because the planner computation is reorganized to better match the temporal structure of streaming autonomous driving.

C More Experiment Setup

C.1 Training parameters

Table 6 summarizes the training configurations used in both supervised fine-tuning (SFT) and reinforcement fine-tuning (RFT). We report the shared experimental settings across the two stages, together with the stage-specific hyperparameters adopted in each training phase.

Table 6: **Training parameters for SFT and RFT.**

Parameter	SFT	RFT
Shared Parameters		
Hardware	16 × NVIDIA A100-SXM4-80GB	
Spatial range	$X \in [-120\,\text{m}, 200\,\text{m}]$, $Y \in [-120\,\text{m}, 120\,\text{m}]$, res = 0.1 m	
Vehicle parameters	w = 2.297 m, l_f = 4.049 m, l_r = 1.127 m	
Optimizer	AdamW (weight decay = 0.01)	
Scheduler	Cosine Annealing (5 warmup epochs)	
Precision	bfloat16	
Stage-Specific Parameters		
Batch size	7	4
Learning rate	1×10^{-4}	5×10^{-6}
Training epochs	100	14
DAC Field loss	λ_{DAC} = 10, r_{DAC} = 0.3 m N = 3, ϵ_{safe} = 0.5 m	–
SAP Loss	λ_{SAP} = 4	–
Group Size	–	6 ($N_g = 3$, $I = 2$)
Steps	–	5 (1 AutoEdit)
λ_{KL}	–	0

C.2 Inference parameters

Table 7 presents the inference-time configuration used for discrete diffusion decoding. In particular, we employ a low-confidence remasking strategy and report the corresponding decoding budget and post-processing settings used in all experiments. For best-of-N evaluation, AutoEdit is only applied to the top-1 trajectory to preserve modal diversity.

Table 7: **Inference parameters.**

Parameter	Value
Steps	7 (3 AutoEdit)
Answer length	16
Block length	16
Remask strategy	low-confidence
NMS threshold	1.21 m

D Inference Hyperparameters Analysis

This appendix primarily analyzes the influence of inference hyperparameters on model performance.

Inference budget. Figure 6 sweeps generation steps and AutoEdit steps: performance improves and then plateaus around 3–5 steps, consistent with masked diffusion – a small number of rounds forms a coherent trajectory, while excess rewriting disturbs a good draft.

Goal-proposal parameters. Figure 7 varies the number of goal proposals and the NMS threshold in best-of-N. More proposals expose more behavior hypotheses; an NMS threshold near 1.2 m is optimal – smaller keeps duplicates, larger removes genuine alternatives.

E Reward Definition

Our RL reward is the NAVSIM Predictive Driver Model Score (PDMS) itself, so that the optimization target is directly aligned with the metric used for evaluation. Following the official NAVSIM definition, PDMS aggregates the no-at-fault-collision (NC), drivable-area-compliance (DAC), time-to-collision

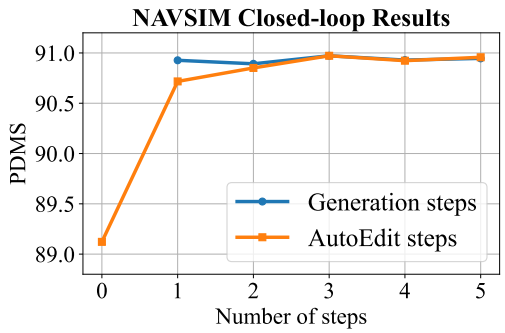

Figure 6: **Sensitivity to diffusion steps.** PDMS with different generation and AutoEdit step counts.

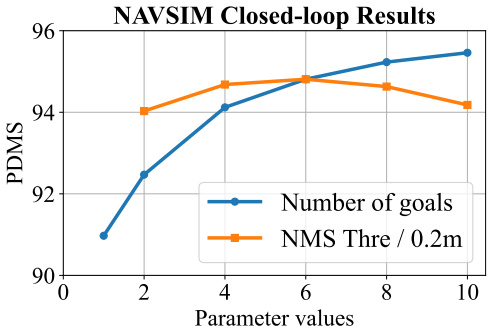

Figure 7: **Sensitivity to goal-proposal parameters.** PDMS vs. number of proposals and NMS threshold.

(TTC), comfort (C), and ego-progress (EP) sub-scores as

$$\text{PDMS} = \text{NC} \times \text{DAC} \times \frac{5\,\text{TTC} + 5\,\text{EP} + 2\,\text{C}}{12}. \tag{21}$$

The multiplicative NC and DAC terms act as hard safety gates, while the weighted average over TTC, EP, and C balances safety, progress, and comfort.

F More Results

F.1 Reward Re-weighting for the Safety–Progress Trade-off

Under the default reward, RL optimizes the aggregate PDMS and markedly improves EP, but the safety-related sub-metrics NC and TTC are not the highest and can decline relative to the supervised model (Table 3). This is a possible consequence of reward maximization rather than a defect of the method: RL maximizes the total reward, so when progress and safety trade off, the policy is free to concede a small amount of safety margin as long as the aggregate score increases. Crucially, this balance is controllable through the reward itself rather than through any change to the algorithm.

To demonstrate this, we keep the model, data, and training procedure fixed and only re-weight the ratio between the EP and TTC sub-terms in the RL reward from 1:1 to 1:20, obtaining a conservative variant denoted ReflectDrive-2 (cons.). As shown in Table 8, the conservative reward recovers the safety margin: every sub-metric of the conservative model matches or improves over the supervised model, and the aggregate PDMS equals that of the default RL model. The safety–progress trade-off is therefore a tunable property of the reward weighting, not an intrinsic limitation of reward-coupled AutoEdit.

Table 8: **Effect of re-weighting the EP:TTC ratio in the RL reward (navtest).** Only the reward weighting is changed; the model, data, and algorithm are identical. The conservative variant matches or improves every sub-metric over the supervised model while matching the aggregate PDMS of the default RL model.

Model	**NC↑**	**DAC↑**	**TTC↑**	**Comf.↑**	**EP↑**	**PDMS↑**
ReflectDrive-2 (SFT)	98.0	96.8	93.8	**100.0**	82.3	88.0
ReflectDrive-2 (RL, EP:TTC = 1:1)	97.3	98.1	92.5	**100.0**	**89.4**	**91.0**
ReflectDrive-2 (cons., EP:TTC = 1:20)	**98.3**	**98.5**	**95.0**	**100.0**	85.9	**91.0**

F.2 Evaluation on NAVHard

To further validate ReflectDrive-2, we additionally evaluate on NAVHard [Cao et al., 2025], a harder, pseudo-simulation benchmark that collects long-tail, reactive, and out-of-distribution scenarios and reports a two-stage EPDMS that is far from saturated. This setting directly probes robustness

and error-correction behavior under challenging conditions. We compare ReflectDrive-2 against ReCogDrive and DiffusionDrive under the same protocol, and also evaluate the conservative variant from Section F.1. We additionally report both variants with AutoEdit disabled.

Table 9: **Evaluation on NAVHard.** S1 and S2 denote the two evaluation stages. EPDMS extends PDMS with driving-direction compliance (DDC), traffic-light compliance (TLC), lane keeping (LK), history comfort (HC), and extended comfort (EC). Results of ReCogDrive and DiffusionDrive are taken from DriveFine [Dang et al., 2026]. AE indicates whether AutoEdit is enabled (✓) or disabled (×). The conservative variant (cons.) with AutoEdit leads on the aggregate EPDMS in both stages and on the safety sub-metrics (NC, TTC) in the harder S2 stage.

Method	Stage	AE	NC↑	DAC↑	DDC↑	TLC↑	EP↑	TTC↑	LK↑	HC↑	EC↑	EPDMS↑
ReCogDrive [Li et al., 2025b]	S1	–	**97.1**	80.2	98.4	**100.0**	83.6	95.3	93.6	**97.6**	76.0	68.9
DiffusionDrive [Liao et al., 2025]	S1	–	96.8	86.0	**98.8**	99.3	84.0	**95.8**	**96.7**	**97.6**	**79.6**	66.7
ReflectDrive-2	S1	×	93.0	90.9	98.3	99.6	**90.9**	92.0	94.7	95.8	56.0	74.0
ReflectDrive-2	S1	✓	93.4	92.2	97.4	99.8	90.1	92.7	94.4	97.3	61.3	76.2
ReflectDrive-2 (cons.)	S1	×	95.6	91.8	97.3	**100.0**	85.7	95.1	94.2	96.9	63.6	76.9
ReflectDrive-2 (cons.)	S1	✓	95.6	**92.7**	98.1	**100.0**	85.0	95.1	94.9	**97.6**	66.2	**78.2**
ReCogDrive [Li et al., 2025b]	S2	–	78.2	69.3	83.9	98.2	86.6	73.9	44.1	96.4	72.0	37.8
DiffusionDrive [Liao et al., 2025]	S2	–	80.1	72.8	84.4	**98.4**	85.9	76.6	46.4	96.3	**72.8**	40.5
ReflectDrive-2	S2	×	77.9	72.2	80.5	97.6	**95.0**	73.7	**47.2**	94.3	47.9	37.6
ReflectDrive-2	S2	✓	78.4	73.8	81.0	97.8	**95.0**	73.7	46.3	95.3	48.5	39.4
ReflectDrive-2 (cons.)	S2	×	79.3	75.3	82.2	98.1	91.2	74.6	43.3	96.1	56.2	40.3
ReflectDrive-2 (cons.)	S2	✓	**81.8**	**76.9**	**85.1**	98.2	90.3	**77.7**	44.3	**97.1**	59.6	**44.0**

As shown in Table 9, on the far-from-saturated EPDMS ReflectDrive-2 clearly outperforms both baselines in S1 (76.2 vs. 68.9 and 66.7), with the advantage concentrated in EP, consistent with the navtest picture. Applying only the reward re-weighting of Section F.1 without any algorithmic change, the conservative variant leads on the aggregate EPDMS in both stages and, in the harder S2 stage, also dominates the safety sub-metrics NC and TTC. Unlike the thin $+0.2$ margin on navtest, the improvement over ReCogDrive on this harder benchmark is substantial ($+9.3$ EPDMS in S1 and $+6.2$ in S2), providing stronger evidence that the reward-coupled AutoEdit mechanism remains effective in long-tail, reactive, and out-of-distribution scenarios.

Comparing the rows with and without AutoEdit, AutoEdit consistently improves EPDMS for both variants in both stages ($74.0 \rightarrow 76.2$ and $76.9 \rightarrow 78.2$ in S1; $37.6 \rightarrow 39.4$ and $40.3 \rightarrow 44.0$ in S2), with the largest gain for the conservative variant in the harder S2 stage. The improvement is most visible in the safety-related sub-metrics (S2, cons.: NC $79.3 \rightarrow 81.8$, DAC $75.3 \rightarrow 76.9$, TTC $74.6 \rightarrow 77.7$), at a small cost in progress (EP $91.2 \rightarrow 90.3$).

F.3 Qualitative Analysis: Pre-/Post-RL Drafts and AutoEdit

To visualize how full-rollout RL jointly improves draft generation and AutoEdit refinement, we plot four trajectories on the same scene: the SFT draft, the SFT AutoEdit output, the RL draft, and the RL AutoEdit output (Figure 8). Blue and red curves denote the SFT and RL models respectively; dashed and solid curves denote the draft and the post-AutoEdit trajectory; titles report per-sub-metric gains.

The four cases consistently show the progression "RL draft $>$ SFT AutoEdit, and RL AutoEdit $>$ RL draft": in case 1 the RL draft is already more efficient than the SFT AutoEdit output and AutoEdit further removes the residual TTC risk; in case 2 the RL draft corrects the SFT trajectory's drivable-area violation and AutoEdit then improves EP; in cases 3 and 4 (longitudinal and lateral) the RL draft turns a direct collision left by SFT AutoEdit into a mere TTC risk, which AutoEdit subsequently eliminates. In contrast, the SFT draft and its AutoEdit output are nearly overlapping, mirroring the small $+0.3$ supervised AutoEdit gain in Table 1. Together, these examples show that RL improves the draft itself while AutoEdit still contributes additional safety and efficiency gains.

F.4 Representative Failure Cases after AutoEdit

For completeness we also analyze scenarios that remain unsatisfactory after AutoEdit (Figure 9). Blue and red curves denote the draft and the post-AutoEdit trajectory; endpoint numbers are per-trajectory scores and the titles state the failure cause.

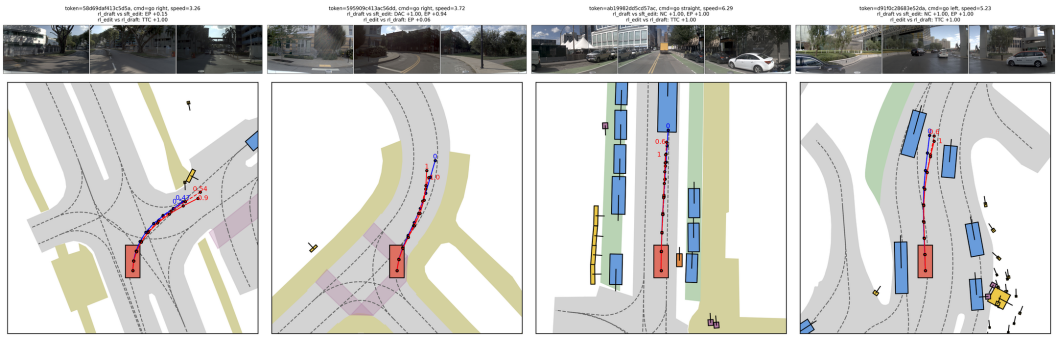

Figure 8: **Pre-/post-RL drafts and their AutoEdit refinements on the same scenes.** Blue/red: SFT/RL model; dashed/solid: draft/post-AutoEdit. Endpoint numbers are per-trajectory scores; titles report per-sub-metric gains.

In cases 1 and 2 the draft contains a longitudinal collision risk; AutoEdit correctly decelerates but the adjustment is insufficient to fully remove the risk. In case 3 the generated goal point lies well inside a non-drivable region, an error too large for AutoEdit's local adjustment to recover. A likely explanation is the *locality* of AutoEdit: during RL its edit magnitude is sampled from a policy distribution that becomes concentrated as training converges, so at inference AutoEdit mainly performs small, local refinements and cannot repair large draft errors. Case 4 shows a rare instance where the score decreases after AutoEdit; such cases occur when the trajectory lies right at a scoring threshold (e.g. the drivable-area boundary), where even a small learned edit can cross the threshold. As a learned local operator, AutoEdit cannot be guaranteed error-free. These failure modes motivate future work on improving AutoEdit's accuracy and adaptive magnitude and on reducing rare harmful edits.

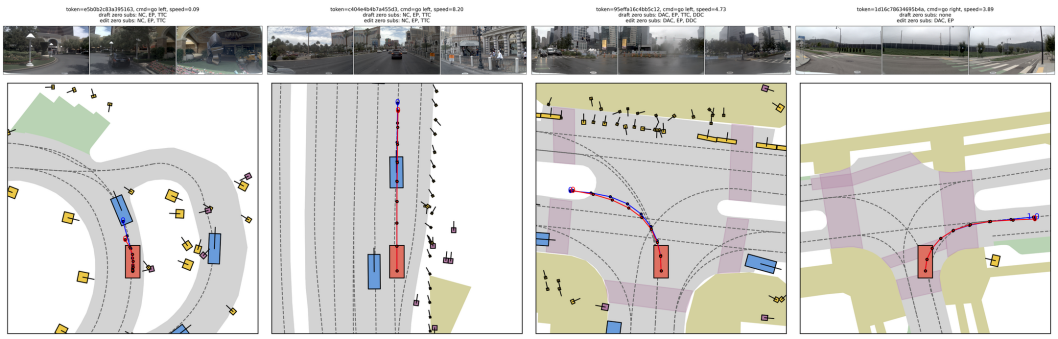

Figure 9: **Representative failure cases after AutoEdit.** Blue/red: draft/post-AutoEdit trajectory. Endpoint numbers are per-trajectory scores; titles give the failure cause.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state three contributions. These claims are supported by experimental results in Section 6 (Tables 1–3 and Figures 4–5), with deployment analysis in Section 5 and Appendix B.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.

- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 7 ("Limitations & Future Work") discusses the limitations of open-loop reinforcement learning and the use of a lightweight reward as a proxy for real-world driving objectives. It outlines future work on closed-loop reinforcement learning and world models.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not claim formal theoretical results (theorems or lemmas). It presents formulations of loss functions and RL objectives (Sections 3–4), which are standard instantiations of masked diffusion and policy-gradient methods with all assumptions stated in context.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.

- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 6.1 specifies the dataset (NAVSIM navtrain/navtest splits), model architecture (0.7B language backbone + 0.1B ViT), input format (two temporal frames from three cameras), output format (16 discrete coordinate tokens for 8 waypoints), training procedure (supervised fine-tuning then RL with PDMS reward), and evaluation metric (PDMS with sub-metrics). The method sections provide all loss formulations and the RL objective.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

Justification: The source code and model weights cannot be publicly released due to internal project constraints and proprietary model assets. All dataset resources used in this work are publicly available, including NAVSIM and nuPlan benchmarks. The paper provides complete training configurations, inference hyper-parameters, loss formulations, and detailed deployment latency measurements in the main text and appendices (Appendix B, C, D), which contain sufficient information for researchers to reproduce the described algorithm.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 6.1 provides data splits, appendix C lists the training and inference parameters, appendix D shows the inference hyperparameters analysis, and the training objectives and rl objectives are shown in section method.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper does not report error bars or confidence intervals for the main results. NAVSIM evaluation is deterministic given a fixed model and fixed goal-selection strategy, but variability across training seeds is not reported due to the computational cost of training multiple 0.7B-parameter models.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies the deployment hardware (NVIDIA Thor) and reports detailed latency measurements for each component of the inference stack in Table 4. The training uses a 0.7B language model and 0.1B ViT, providing model-size context for compute requirements.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: The paper does not include a dedicated broader-impact discussion. The work targets autonomous driving, which has clear positive societal implications.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release pre-trained models or datasets. The model weights are proprietary and not publicly available, and the evaluation benchmark (NAVSIM) is already publicly released by its original authors.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites all datasets (NAVSIM, nuPlan) and baseline methods used in the evaluation.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new datasets, code, or model checkpoints as public assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects. All evaluations are conducted automatically on the NAVSIM benchmark.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The paper does not involve human subjects research. All experiments use pre-collected driving datasets and automated evaluation.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method uses a masked discrete diffusion model (0.7B parameters) that is not a pre-trained LLM repurposed for this task. The backbone is a proprietary model trained specifically for trajectory generation, not a general-purpose language model.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.