FlashPlanner: Real-Time Goal-Conditioned Flow-Matching Planning for Autonomous Driving with Online RL Fine-Tuning

Qifeng Li[1,2,*], **Yubing Gao**[1,3,*], **Xiaosong Jia**[4], **Zhiliu Liu**[1,3], **Sizhuo Zhou**[3], **Wenlong Liao**[5], **Tao He**[5], **Junchi Yan**[1,3,†]
[1]Shanghai Jiao Tong University [2]Beijing Zhongguancun Academy
[3]Shanghai Innovation Institute [4]Fudan University
[5]COOWA
{liqifeng, gaoyubing, liuzhiliu, yanjunchi}@sjtu.edu.cn
{volans.liao, tommie.he}@cowarobot.com

Abstract

Diffusion and flow-matching models have emerged as expressive generative planners for autonomous driving, owing to their ability to model high-fidelity and multi-modal trajectory distributions. Nevertheless, existing generative planners are predominantly trained with imitation learning, which induces a fundamental mismatch between supervised trajectory fitting and closed-loop planning objectives. Direct RL fine-tuning faces challenges arising from the iterative generative sampling, high-dimensional trajectory space and mode collapse. In this paper, we propose *FlashPlanner*, a goal-conditioned flow-matching planner with online RL fine-tuning for autonomous driving. *FlashPlanner* introduces a continuous future goal point produced by a goal predictor, which is first pretrained and then optimized with RL using closed-loop feedback. The goal point serves as a conditioning interface for trajectory generation and a compact action space for RL fine-tuning, enabling effective policy optimization. To support efficient online reinforcement fine-tuning and real-time deployment, *FlashPlanner* adopts a data-prediction flow-matching objective and removes redundant architectural components in existing diffusion-based planners. Experiments on nuPlan and interPlan demonstrate that *FlashPlanner* achieves state-of-the-art planning performance while delivering 6× faster inference than the previous SOTA baseline.

1 Introduction

Autonomous driving demands planning systems that are safe, efficient, and robust across diverse real-world traffic scenarios. Learning-based planners [1–4], especially imitation-based planners [2–4], have demonstrated strong potential, offering scalable and flexible advantages over rule-based methods. Among them, diffusion-based planners have emerged as a promising paradigm due to their ability to model multi-modal trajectory distributions.

However, learning-based planners trained purely by imitation learning suffer from distributional shift and objective misalignment with closed-loop evaluation metrics. Reinforcement learning (RL) can mitigate these issues through simulator-assisted exploration and reward-oriented optimization. Yet directly applying RL to fine-tune trajectory-level generative models is challenging: the high-dimensional action space of full trajectories leads to poor **sample efficiency** and unstable training. Moreover, vanilla diffusion-based methods often suffer from **mode collapse**, where trajectories

*Equal contribution. †Corresponding author.

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

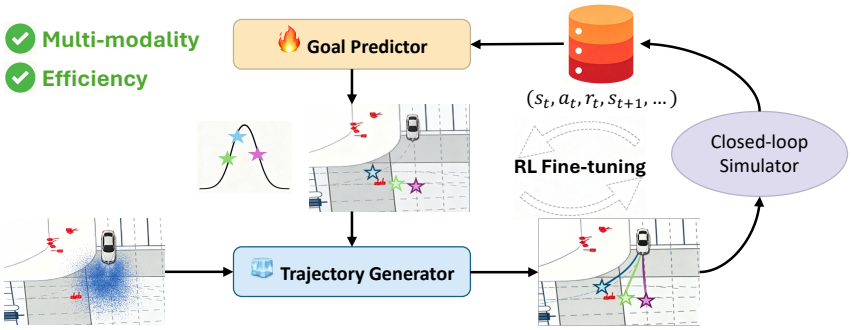

Figure 1: **Overview of FlashPlanner.** The Goal Predictor generates diverse goal points as conditions for the Trajectory Generator. During RL fine-tuning, only the Goal Predictor is updated.

generated from different noise inputs converge to nearly identical outputs during denoising [5–7], thereby limiting the exploration capability essential for RL. On the other hand, existing models inherit heavyweight components from conventional planners such as introducing prior anchors, imposing auxiliary losses, and jointly modeling the ego and other agents [6, 8], without validating their necessity in the generative setting. The resulting computational overhead renders inference speed insufficient for real-time deployment and efficient online RL fine-tuning.

In this paper, we present FlashPlanner, a goal-conditioned flow-matching planner with online reinforcement fine-tuning. Our key insight is to decompose the planning problem into two stages: a high-level *goal predictor* that determines *where to go*, and a low-level *trajectory generator* that determines *how to get there*. FlashPlanner first predicts a future goal point, then generates trajectories conditioned on both the environmental context and the predicted goal. During reinforcement fine-tuning, only the goal predictor is optimized to improve closed-loop planning performance, while the trajectory generator remains frozen. This decomposition yields three critical advantages. First, it enables **efficient** RL fine-tuning by restricting policy exploration to the compact, low-dimensional goal space rather than the full trajectory space. Second, it naturally supports **multi-modal** trajectory generation—different goal points induce distinct trajectory modes. Third, it provides an intuitive interface for **controllability**, as the goal point directly encodes high-level driving intent such as lane changes, turns, and merges. By systematically revisiting the design choices in existing diffusion based planners, we further remove redundant designs and devise a lightweight model architecture, which significantly improves trajectory generation efficiency.

FlashPlanner achieves superior closed-loop performance among learning-based baselines on the nuPlan, a large-scale real-world autonomous planning benchmark [9]. It maintains similarly competitive robustness on the interPlan benchmark [10], which focuses on rare and highly interactive long-tail scenarios. Especially in the reactive mode that more closely reflects real-world closed-loop driving, it exceeds PlannerRFT [11] by **4.40** on nuPlan Test14-hard split. FlashPlanner also shows outstanding inference efficiency, delivering **166 FPS** inference speed.

In summary, our contributions are as follows:

- We propose FlashPlanner, a goal-conditioned flow-matching planner that decouples high-level intent prediction from trajectory generation, enabling efficient multi-modal planning with explicit controllability.
- We introduce a reinforcement fine-tuning framework that optimizes the goal predictor via PPO in the low-dimensional goal space, effectively addressing distribution shift while improving training stability and sample efficiency.
- We systematically analyze the design choices in existing diffusion-based planners and prune redundant components, yielding a lightweight and robust architecture for real-time planning.
- FlashPlanner achieves SOTA performance on both the closed-loop nuPlan benchmark and the interPlan benchmark, while delivering a speedup of approximately 6× compared with the best existing approach [11].

2 Related Work

Imitation-based planner. Imitation-based AD planners [12–14] learn the driving policy by cloning control policies from expert demonstrations. With the abundance and affordability of driving data

today, this imitation learning (IL) paradigm achieves remarkable advances due to its scalability and ease of convergence, especially in the end-to-end pipelines [2, 15, 16]. However, IL methods are vulnerable to issues like covariate shift and mode collapse, which incur performance degradation, especially in closed-loop setting [17, 18]. Many techniques [3, 4] have been proposed to address these issues and push the limit of the imitation-based approach. Here we adopt the mid-to-end fashion, which utilizes post-perception outcomes as input, to focus on the planning task. Despite these efforts, most existing approaches still heavily rely on post-processing to refine or select appropriate trajectories, which means they fail to capture the expert driving policies to some extent.

Diffusion-based planner. The potential of diffusion and flow matching for planning tasks was first demonstrated in robotics [19, 20]. More recent works have highlighted diffusion models' strong expressiveness for autonomous driving tasks, including motion planning [8, 21, 22], traffic simulation [23–25], and end-to-end driving [6, 26–29]. Several works adapt diffusion models to advance IL-based AD planners but encounter unexpected obstacles such as mode collapse, trajectory divergence, and low efficiency [5–7]. These obstacles are alleviated through practices from conventional IL methods like introducing prior anchors and auxiliary losses, which result in inefficient training and inference. Some works [8] try to reduce the latency by using DPM-Solver [30], a smaller encoder, and jointly modeling ego and other agents, which achieves good performance. Nevertheless, these diffusion-based methods are built atop practices from conventional imitation-based methods, yet the principles for well-designed diffusion/flow-based planners are still under exploration.

3 Preliminaries

Flow Matching. Flow Matching (FM) is a class of generative transport models that learns to transform samples from a known initial distribution π_0 to a target distribution π_1, by modeling the time-dependent velocity field of probability flows. Different from standard diffusion models that usually rely on complex noise schedules, FM is simulation-free and is easier for training [7, 31, 32]. The flow path is characterized by intermediate states $x_t \sim p_t(x)$ where $t \in [0, 1]$, with the instantaneous velocity defined as $v_t = dx_t/dt$. A neural network v_θ is trained to approximate this velocity field by minimizing the loss function:

$$L_v(\theta) = \mathbb{E}\left[\|v_\theta(x_t, t) - v_t\|^2\right]. \tag{1}$$

The learned velocity predictor v_θ enables sample generation by solving the ordinary differential equation (ODE) $\frac{dx_t}{dt} = v_\theta(x_t, t)$ starting from $x_0 \sim \pi_0$, yielding the solution:

$$x_1 = x_0 + \int_0^1 v_\theta(x_t, t)dt. \tag{2}$$

Task Definition. Autonomous driving motion planning takes processed environmental context as input and produces safe, feasible, and optimal future trajectories for the ego-vehicle to follow [3]. The trajectory is represented as a sequence of states $\mathbf{x} = \{(x^\tau, y^\tau, \theta^\tau)\}_{\tau=1}^T$, where T denotes the planning horizon, θ^τ is the heading angle, and (x^τ, y^τ) indicates the location at time τ in the current ego-vehicle coordinate system. We adopt the convention that subscripts index the continuous flow-matching time $t \in [0, 1]$, while superscripts index the discrete trajectory timesteps $\tau \in \{1, \ldots, T\}$. Leveraging the strong expressive power of generative models and further enhancing closed-loop performance via reinforcement learning, we propose FlashPlanner for this task, which generates multi-modal, high-quality trajectories in real-time.

4 Method

In this section, we present FlashPlanner, a goal-conditioned flow-matching planner with online RL fine-tuning for autonomous driving. Section 4.1 provides an overview of FlashPlanner. Section 4.2 details the goal predictor and trajectory generator, followed by the reinforcement fine-tuning procedure in Section 4.3. Section 4.4 discusses redundancy reduction strategies that yield a lightweight yet effective architecture, and Section 4.5 presents the detailed model design.

4.1 Overview of FlashPlanner

As illustrated in Figure 1, FlashPlanner first predicts a future goal point and then generates trajectories conditioned on both the environmental context and the predicted goal. During reinforcement fine-tuning, the goal predictor is optimized using PPO to improve closed-loop planning performance, while the trajectory generator remains frozen.

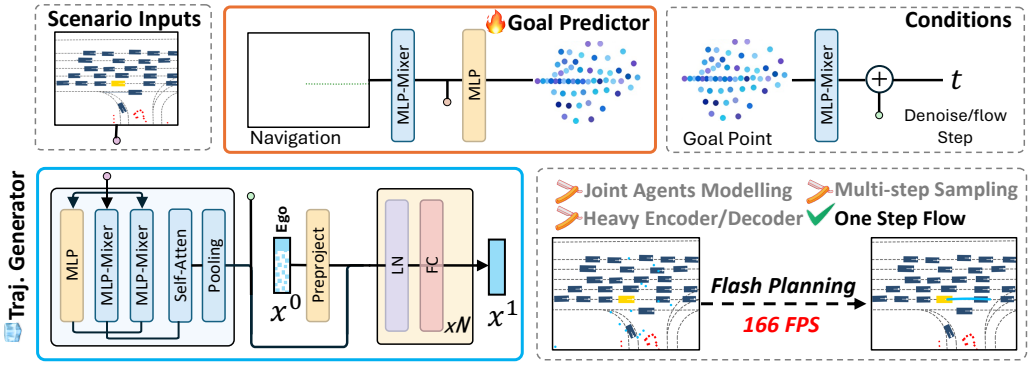

Figure 2: **Model Architecture of FlashPlanner.** Redundancy reduction strategies are employed to achieve real-time inference at 166 FPS.

4.2 Goal-Conditioned Flow-Matching Planner

Goal Predictor. We introduce a goal predictor to decouple "where to go" from "how to get there." Given the environmental context C, the goal predictor outputs a continuous goal point $g = h_\phi(C)$, where $h_\phi(\cdot)$ is a neural network parameterized by ϕ and g denotes the endpoint of the planning horizon in the ego-centric coordinate frame. The goal predictor is pretrained via supervised regression:

$$\mathcal{L}_{\text{goal}}(\phi) = \mathbb{E}\left[\|h_\phi(C) - g^*\|^2\right], \tag{3}$$

where g^* is obtained from the endpoint of the corresponding expert trajectory.

Prior work such as GoalFlow [7] constructs a dense goal vocabulary by clustering the training data into 4096/8192 discrete candidates. Although this discrete formulation provides strong constraints on trajectory generation, it introduces an inherent trade-off: too few candidates limit the performance ceiling, while too many enlarge the exploration space and impede reinforcement learning. To overcome this limitation, we adopt a continuous goal representation. The predicted goal g serves as a compact intent variable that encodes high-level motion decisions (e.g., lane following, turning, merging). Conditioning the planner on g improves **controllability** by enabling direct steering of generated behaviors, and naturally supports **multi-modality** by mapping different goals to distinct feasible trajectory modes.

Trajectory Generator. Conditioned on the context C and the predicted goal g, the planner directly predicts the clean trajectory x_1 from a partially noised sample x_t. The training objective is

$$\mathcal{L}_{\text{traj}}(\theta) = \mathbb{E}\Big[\|f_\theta(x_t, t, C, g) - x_1\|^2\Big] \; + \; \alpha\, \mathbb{E}\Big[\|\text{end}(f_\theta(x_t, t, C, g)) - g\|^2\Big], \tag{4}$$

where x_1 represents the expert trajectories used for supervision, and $\text{end}(\cdot)$ extracts the trajectory endpoint. The second term enforces consistency between the predicted endpoint and the goal, which is essential for ensuring controllability and multi-modality.

During pretraining, FlashPlanner is jointly optimized with

$$\mathcal{L}_{\text{pre}}(\theta, \phi) = \mathcal{L}_{\text{traj}}(\theta) + \beta\, \mathcal{L}_{\text{goal}}(\phi). \tag{5}$$

At inference time, FlashPlanner generates multi-modal trajectories by transporting samples from an initial noise distribution to the goal-conditioned trajectory distribution. We solve the corresponding flow ODE using the Euler method, yielding the final trajectory as

$$\mathbf{x}_1 = \mathbf{x}_0 + \sum_{i=1}^{n} \frac{1}{n} \cdot \frac{f_\theta(\mathbf{x}_{t_i}, t_i, \mathbf{C}, \mathbf{g}) - \mathbf{x}_{t_i}}{1 - t_i}, \tag{6}$$

where n is the total number of inference steps, and $t_i \in [0, 1)$ is the flow time at the i-th step. In practice, a single inference step ($n = 1$) is sufficient to generate high-quality trajectories.

4.3 Reinforcement Fine-Tuning

To further improve closed-loop planning performance, we perform online RL fine-tuning on the goal predictor using the PPO framework, while keeping the trajectory generator frozen. Instead of

optimizing over full trajectories, we restrict policy exploration to the low-dimensional goal space, which improves sample efficiency and training stability. The procedure is summarized in Algorithm 1.

Policy Formulation. At each simulation step k, the environmental context C_k serves as the observation. The goal predictor defines a stochastic policy over continuous goal points: $\pi_\phi(g_k \mid C_k) = \mathcal{N}\big(h_\phi(C_k), \Sigma\big)$, where $h_\phi(C_k)$ is the predicted mean and Σ is a learnable covariance. Given a sampled goal $g_k \sim \pi_\phi(\cdot \mid C_k)$, the frozen generator produces a trajectory conditioned on (C_k, g_k). Different goals induce diverse driving behaviors, enabling the policy to explore beyond the support of expert demonstrations and discover better driving strategies through reward-driven optimization.

Algorithm 1 FlashPlanner: Reinforcement Fine-Tuning

Require: Pretrained goal predictor h_ϕ, frozen trajectory generator f_θ, value network V_ψ, closed-loop simulator $\mathcal{E}$, iterations N, rollout horizon T, update epochs K

for iteration $= 1, 2, \ldots, N$ **do**
  $s_0 \sim \mathcal{E}$.reset(), $\mathcal{B} \leftarrow \emptyset$
  for $k = 0, 1, \ldots, T-1$ **do**
    Extract context C_k from state s_k
    Sample goal: $g_k \sim \pi_\phi(\cdot \mid C_k) = \mathcal{N}(h_\phi(C_k), \Sigma)$
    Generate trajectory with one step: $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\mathbf{x}_1 = f_\theta(\mathbf{x}_0, 0, C_k, g_k)$
    Execute in simulator: $s_{k+1}, r_k \leftarrow \mathcal{E}$.step$(\mathbf{x}_1)$
    Store $(C_k, g_k, r_k, V_\psi(s_k), \log \pi_\phi(g_k \mid C_k))$ in $\mathcal{B}$
  end for
  Compute GAE advantages $\{\hat{A}_k\}$ using $\{r_k, V_\psi(s_k)\}$
  for epoch $= 1, 2, \ldots, K$ **do**
    Update ϕ, ψ via Equation (8)
  end for
  $\phi_{\text{old}} \leftarrow \phi$
end for
return Fine-tuned goal predictor h_ϕ

Closed-Loop Rollout and Policy Optimization. The first action of the generated trajectory is executed in the closed-loop simulator, advancing the environment state from s_k to s_{k+1} and yielding an immediate reward r_k. The transition tuple $(C_k, g_k, r_k, V_\psi(s_k), \log \pi_\phi(g_k \mid C_k))$ is stored in a replay buffer $\mathcal{B}$. The goal predictor is then updated by maximizing the PPO clipped surrogate objective:

$$\mathcal{L}^{\text{PPO}}(\phi) = \mathbb{E}_{(C_k, g_k)\sim\mathcal{B}}\left[\min\Big(\rho_k(\phi)\,\hat{A}_k,\ \text{clip}\big(\rho_k(\phi),\, 1-\epsilon,\, 1+\epsilon\big)\,\hat{A}_k\Big)\right], \tag{7}$$

where $\rho_k(\phi) = \frac{\pi_\phi(g_k|C_k)}{\pi_{\phi_{\text{old}}}(g_k|C_k)}$ is the importance sampling ratio and ϵ is the clipping threshold. The overall reinforcement fine-tuning objective is:

$$\mathcal{L}^{\text{RFT}}(\phi, \psi) = -\mathcal{L}^{\text{PPO}}(\phi) + c_1\,\mathcal{L}^{\text{value}}(\psi) - c_2\,\mathcal{H}\big[\pi_\phi(\cdot \mid C_k)\big], \tag{8}$$

where $\mathcal{L}^{\text{value}}(\psi)$ is the value function regression loss, c_1, c_2 are balancing coefficients, and the entropy term $\mathcal{H}[\cdot]$ encourages sufficient exploration in the goal space. Through iterative rollouts and PPO updates, the goal predictor progressively aligns with reward-oriented objectives, improving closed-loop stability and overall planning performance.

4.4 Redundancy Reduction Strategies

Existing diffusion-based planners often incorporate design choices inherited from conventional imitation-based methods [5, 8]. For example, Diffusion Planner [8] jointly models the traffic prediction and ego planning tasks as a trajectory generation task inspired by prior works [2, 33]. These involve additional computation and convergence difficulties, while their effectiveness has not been fully validated. We boldly remove a series of redundant designs and empirically verify their redundancy in diffusion-based planners. As a result, we obtain a lightweight yet powerful diffusion-based planner with real-time performance, which is sufficient to support efficient online RL fine-tuning and real-time deployment. Accordingly, we first describe the redundant structures we removed, and then present our lightweight architecture.

Planning-only objective. Trajectories of neighboring vehicles benefit planning, but joint modeling of traffic prediction and ego planning introduces unnecessary complexity and requires additional modules for multi-agent interaction representation. Inspired by [34], we focus exclusively on ego planning while leveraging other vehicles' trajectories as additional ego samples for training. Specifically, we densify supervision by treating other agents as ego vehicles via coordinate transformation. To ensure augmentation quality, we filter out trivial straight-driving samples and retain only trajectories with significant heading or velocity variations. This approach uses other agents' behaviors to create more diverse, challenging driving scenarios without extra data collection, thereby enhancing sample efficiency and generalization capability.

MLP v.s. Transformer. Prior diffusion-based planners typically employ heavy transformer-based decoders with multi-layer self-attention to capture interactions between the ego trajectory and the environment [5, 8]. However, recent advances in related domains [35] have demonstrated that appropriate objective design and architecture enable lightweight networks to deliver robust performance. Inspired by that, we developed a lightweight MLP-based decoder for FlashPlanner. It replaces the complex transformer structure and condition mechanism with several fully-connected layers and layer-norm layers. We empirically find that a compact MLP-based decoder is sufficient for high-quality trajectory generation while substantially reducing computational overhead.

Other reduction strategies. We further investigate more components, such as data augmentation, the depth of the encoder, and the structure of the predicted vector. Figure 3 provides an overview of the design choices introduced in FlashPlanner. Detailed analysis of each component is presented in Section 5.3 and Appendix Section E.

4.5 Lightweight Model Architecture

FlashPlanner presents a lightweight MLP-centric architecture, featuring a simple yet effective mechanism for multi-modal trajectory generation. Figure 2 illustrates an overview of the complete architecture, and a detailed description is provided as follows.

Encoder. For a driving scene at the current timestep, the encoder inputs consist of the historical trajectories of N_d dynamic agents over T_h timestamps, the status of N_s static obstacles, and the information of N_l lanes, denoted as $\mathbf{F}_d \in \mathbb{R}^{N_d \times T_h \times D_d}$, $\mathbf{F}_s \in \mathbb{R}^{N_s \times D_s}$, and $\mathbf{F}_l \in \mathbb{R}^{N_l \times P_l \times D_l}$, respectively, where P_l represents the number of waypoints per lane polyline. The feature dimension D_d contains agent coordinates, heading, velocity, size, and category. D_s is analogous but excludes velocity. D_l provides lane details encompassing waypoint coordinates, traffic light status, and speed limits. Following the encoder architecture of Diffusion Planner [8], each modality is initially embedded by modality-specific MLPs and processed with MixerBlocks [36] for feature extraction and temporal-spatial fusion. Subsequently, the encoded representations are concatenated across modalities and passed through a multi-head self-attention block to model cross-modal interactions, producing a comprehensive scene encoding $\mathbf{F} \in \mathbb{R}^{(N_d+N_s+N_l)\times D}$, where D is the hidden dimension. In contrast to the multi-layer stacking used in Diffusion Planner, we adopt a more efficient design featuring only one MixerBlock per modality and a single self-attention layer, achieving significant parameter reduction while maintaining representation capabilities. The scene encoding is further aggregated via mean pooling, yielding a compact scene representation $\mathbf{F}' \in \mathbb{R}^D$. The navigation information, which specifies the intended route, is denoted as $\mathbf{F}_r \in \mathbb{R}^{N_r \times P_l \times D_l}$ where N_r indicates the number of route lanes. We encode $\mathbf{F}_r$ through MLP and MixerBlock to obtain the route embedding $\mathbf{F}'_r \in \mathbb{R}^D$. The concatenation of $\mathbf{F}'$ and $\mathbf{F}'_r$ constitutes the environmental context $C_k = [\mathbf{F}', \mathbf{F}'_r]$ shared by downstream modules.

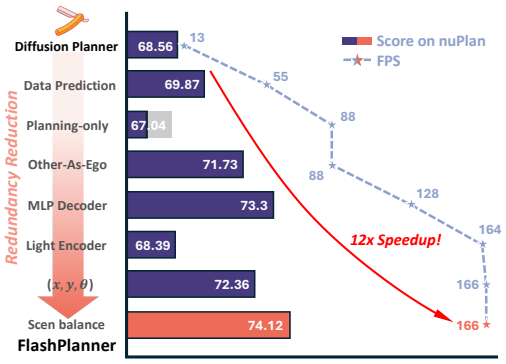

Figure 3: **Redundancy reduction roadmap.** Step-by-step evolution from Diffusion Planner (baseline) to FlashPlanner$_{\text{pre}}$, illustrating the cumulative impact of our design choices on nuPlan Test14-hard (R) performance and inference speed. Detailed description of each variant is provided in Appendix Section A. All experiments were conducted on the same device.

Goal Predictor. The goal predictor h_ϕ takes the environmental context C_k as input and produces a goal point $g_k \in \mathbb{R}^2$ representing the desired endpoint of the planned trajectory. Architecturally, it consists of a two-layer MLP that maps C_k to the goal coordinates. During reinforcement fine-tuning, the output serves as the mean of a Gaussian policy. In parallel, a value head V_ψ, implemented as a separate two-layer MLP, estimates the state-value $V(s_k)$ from the same context C_k to facilitate policy optimization.

Trajectory Generator. It integrates scene context, predicted goal point, noised ego future trajectory, and flow timestep to generate ego future trajectory. Specifically, the goal point g_k is projected to a goal embedding $\mathbf{G} \in \mathbb{R}^D$ via MLP. The flow timestep t is mapped to a fixed-dimensional embedding $\mathbf{T} \in \mathbb{R}^D$ through sinusoidal positional encoding. The noised ego future trajectory $\mathbf{X}_t \in \mathbb{R}^{T_f \times 3}$ is

Table 1: **Closed-loop planning results on nuPlan Val14, Test14-hard, and interPlan.** The **best** and second-best results of each benchmark are highlighted in **bold** and underlined. "w/o" denotes "without". PlannerRFT has not been open-sourced and its interPlan results have not been reported.

Type	Planner	Val14		Test14-hard		interPlan
		NR	R	NR	R	R
Expert	Log-replay	93.53	80.32	85.96	68.80	25.40
Rule-based	IDM	75.60	77.33	56.15	62.26	37.82
	PDM-Closed	**92.84**	**92.12**	65.08	75.19	**43.61**
Learning-based	PDM-Open	53.53	54.24	33.51	35.83	11.47
	UrbanDriver	68.57	64.11	50.40	49.95	9.98
	GameFormer w/o refine.	13.32	8.69	7.08	6.69	6.24
	PlanTF	84.27	76.95	69.70	61.61	33.70
	PLUTO w/o refine.	88.89	78.11	70.03	59.74	22.69
	Diffusion Planner	89.76	82.56	75.67	68.56	32.28
	PlannerRFT	89.96	84.46	77.16	72.21	-
	FlashPlanner (Ours)	91.15	86.42	**78.65**	**76.61**	39.36

projected to $\mathbf{X}'_t \in \mathbb{R}^D$ using MLP. Finally, the four embeddings $[\mathbf{F}', \mathbf{G}, \mathbf{T}, \mathbf{X}'_t]$ are concatenated and processed through a lightweight MLP head to generate the denoised ego future trajectory.

5 Experiments

5.1 Experiment Setup

Dataset and Benchmark. We train and evaluate FlashPlanner on the nuPlan dataset [9], which provides the first publicly accessible, large-scale planning benchmark for autonomous driving with its associated closed-loop simulation framework. For pretraining, we employ a standardized split comprising 1 million frames sampled across all scenario types to ensure broad coverage of driving contexts (see Figure 7). For reinforcement fine-tuning, we follow Li et al. [11] and use low-score (less than 90) scenarios collected from nuPlan at 10 Hz sampling rate. For evaluation, we conduct extensive experiments on both the Test14-hard and Val14 benchmarks under non-reactive (NR) and reactive (R) modes. To further assess the robustness of FlashPlanner in long-tail scenarios, we also evaluate it on the interPlan benchmark [10], which is specifically designed to expose planners to rare and highly interactive scenarios that are not comprehensively covered in nuPlan.

Baseline. We conduct comparisons between FlashPlanner and existing state-of-the-art models on the nuPlan benchmark and the interPlan benchmark. The baselines are categorized into **Rule-based** methods that rely on manually engineered rules and **Learning-based** methods that employ neural networks to directly output the final planned trajectories. Detailed descriptions of each baseline and the implementation details of FlashPlanner are given in Appendix Section B.

5.2 Main Results

Closed-loop Planning Performance. We conduct evaluations of FlashPlanner with state-of-the-art planning methods [3, 4, 8, 11, 14, 37–39] on the nuPlan benchmark and the interPlan benchmark, with the quantitative results presented in Table 1. Among learning-based approaches, FlashPlanner achieves superior overall performance across key metrics, with particularly pronounced gains in the reactive setting. While both FlashPlanner and PlannerRFT employ generative modeling frameworks enhanced through RL fine-tuning and deliver comparable non-reactive results, FlashPlanner exhibits substantial improvements in reactive scenarios, outperforming PlannerRFT by +1.96 on Val14 and +4.40 on Test14-hard, indicating enhanced robustness to interaction-induced disturbances. A similar trend appears on the interPlan benchmark, which only provides reactive evaluation: despite the overall score drop caused by its long-tail, highly interactive scenarios, FlashPlanner maintains a competitive level of robustness within the learning-based group.

Although PDM-Closed performs competitively on Val14, it exhibits limited generalization on the more challenging Test14-hard split. In contrast, FlashPlanner maintains consistently strong performance across both nominal and interaction-heavy scenarios. We attribute these gains to the model's ability to

Table 2: **Efficiency comparison.** PlannerRFT enhances Diffusion Planner's performance through RL fine-tuning while preserving the original model architecture and replacing the DPM-solver denoising with a 5-step DDIM sampler. Inference latency and training time are measured on the same device.

Paradigm	Method	Schedule	Test14-hard R	Inference FPS	Training Time (h)
Pretrain	Diffusion Planner	500 epochs	68.56	13.7	38.3
	$\text{FlashPlanner}_{\text{pre}}$	100 epochs	74.12	166.7	2.5
	Speedup	-	-	**12.2**×	**15.3**×
RL Fine-tune	PlannerRFT	40M env. steps	72.21	28.2	-
	FlashPlanner	10M env. steps	76.61	166.7	8
	Speedup	-	-	**5.9**×	**-**

generate high-quality trajectories that are further enhanced by closed-loop RL fine-tuning, effectively bridging the performance gap between learning-based and rule-based approaches.

Efficiency. Table 2 reports the efficiency comparison. Additional comparisons with learning-based planners are provided in Appendix Table 8. Benefiting from our architectural design, FlashPlanner achieves 166.7 FPS inference speed and completes pretraining in only 2.5 hours. Compared to Diffusion Planner, $\text{FlashPlanner}_{\text{pre}}$ delivers 12.2× faster inference and 15.3× faster training while achieving superior planning performance. For RL fine-tuning, FlashPlanner attains a higher closed-loop score with 4× fewer environment steps than PlannerRFT. Notably, PlannerRFT conducts closed-loop rollouts in nuMax [11], a GPU-parallel simulator that achieves 10× faster rollout than native nuPlan, whereas our RL training uses the nuPlan environment accelerated by [1]. Due to this difference in simulation infrastructure, a direct comparison of training time is omitted. These results show the substantial efficiency of FlashPlanner, making it practical for real-world deployment.

Qualitative Results. We provide qualitative results of FlashPlanner's closed-loop planning performance on challenging scenarios in Figure 4. Each row represents a different scenario captured at 0, 5, 10, and 15 seconds intervals. The visualizations demonstrate that FlashPlanner generates smooth, collision-free trajectories that closely follow the ground-truth expert behavior across diverse driving situations, including lane changes, intersections, and interactions with surrounding vehicles.

5.3 Ablation Study

Based on the ablation results in Table 3 and Table 4, we analyze the contribution of our design choices. Ablations on decoder architecture, prediction type and data preprocessing are conducted at the pretraining stage, using $\text{FlashPlanner}_{\text{pre}}$ as the baseline. In contrast, goal horizon and reward type are evaluated at the RL fine-tuning stage, using FlashPlanner as the baseline. More ablation studies are provided in Appendix Section E.

Decoder Architecture. Despite the wide use of Transformer-based decoders in learning-based motion planning, our ablations indicate that such a choice is unnecessary and inefficient in this setting. Under identical configurations with 100 training epochs, the MLP decoder consistently outperforms its Transformer-based counterpart across both prediction types. It also demonstrates faster convergence: the MLP attains strong performance with 100 epochs, whereas the Transformer requires extended training up to 500 epochs for comparable results. This efficiency advantage is particularly evident on velocity prediction, where the MLP at 100 epochs exceeds the Transformer even at 500 epochs, underscoring the superior efficiency of our simplified architecture for planning.
Prediction Type. As established in Equation (1), the standard flow-matching parameterizes a neural network as a velocity predictor $v_\theta(x_t, t)$. An alternative is data prediction, wherein a network directly predicts the clean endpoint x_1. For a given flow path, these two parameterizations are theoretically equivalent. However, this reparameterization alters the gradient scaling across different timesteps, leading to different optimization and numerical behaviors in practice. For trajectory generation in autonomous driving, data prediction demonstrates superior efficiency in both training and inference, achieving SOTA planning performance with minimal computational cost. Velocity prediction attains an acceptable score with 10 flow steps, yet still underperforms data prediction's single-step results. Training velocity prediction is also more challenging, especially with Transformer-based decoders:

Table 3: **Ablation of decoder architecture and prediction type on nuPlan Test14-hard (NR).**

Decoder	Prediction	Epochs	Steps	Score
MLP	data	100	1	76.46
	velocity	100	1	66.56
			10	75.57
Transformer	data	100	1	74.93
		500	1	76.08
	velocity	100	1	27.83
			10	36.26
		500	1	56.63
			10	66.07

Table 4: **Ablation on nuPlan Test14-hard.**

Component	Setting	NR	R
Baseline	$\text{FlashPlanner}_{\text{pre}}$	76.46	74.12
Preprocessing	w/o data augmentation	68.96	61.82
	w/o scenario balancing	71.54	72.52
	w/o other-as-ego	77.38	68.34
Baseline	FlashPlanner	78.65	76.61
Goal Horizon	4s	79.23	75.13
	5s	78.01	73.08
	8s	75.64	71.01
Reward Type	Survival	76.05	75.90
	Simple	77.69	75.23

extending training from 100 to 500 epochs increases the score from 27.83 to 56.63, indicating that substantially longer training is needed to approach convergence.

Data Preprocessing. For the planning-only objective, we augment the 1M training frames by extracting 250K trajectories from neighboring vehicles with significant heading or velocity changes, treating them as pseudo-ego trajectories. This strategy leverages diverse agent behaviors to create more challenging training samples without additional data collection, demonstrating improved generalization capability. We apply data augmentation [8], including ego-centric transformation, z-score normalization, and state perturbation with future trajectory interpolation. Removing data augmentation yields poor performance, highlighting its role in mitigating out-of-distribution shift and enhancing generalization. Additionally, balancing the number of scenarios per type (see Figure 7) in the trainset mitigates dataset skew and consistently improves planning performance.

Goal Horizon. A goal horizon of 2 seconds achieves the best overall performance. As the horizon increases, compounding uncertainty in goal prediction leads to performance degradation, particularly in reactive mode. The 2-second horizon is short enough to ensure accurate goal prediction, yet sufficiently long to convey meaningful directional intent for trajectory generation.

Reward Type. For the reward design, we compare three formulations. The survival reward provides a non-zero dense signal at every non-terminal timestep, encouraging the planner to optimize toward trajectories that delay failure events. The simple multiplicative reward [1] primarily optimizes route completion as a single intuitive term. FlashPlanner uses a multi-objective weighted reward combining progress, time-to-collision, speed, and comfort: $r = 5 \cdot r_{\text{progress}} + 5 \cdot r_{\text{ttc}} + 4 \cdot r_{\text{speed}} + 2 \cdot r_{\text{comfort}}$. This formulation aligns with the main components of the nuPlan closed-loop score. However, directly applying it leads to reward hacking: the ego vehicle can repeatedly alternate between forward and reverse motion within the same route segment to collect the progress reward multiple times. We address this issue by enforcing monotonic route progress, which achieves the strongest overall performance. This observation is also consistent with CaRL's Appendix E.2, where the weighted reward outperforms the simple reward on nuPlan Val14 by 2.6/2.5 points in NR/R modes.

Continuous Goal-Space PPO. We further disentangle the contribution of continuous goal-space PPO from architectural and data-pipeline improvements, as shown in Table 5. Our lightweight

Table 5: **Ablation of continuous goal-space PPO on nuPlan Test14-hard.**

Configuration	NR	R
Diffusion Planner (IL)	75.67	68.56
+ Continuous Goal-Space PPO	77.90	71.63
$\text{FlashPlanner}_{\text{pre}}$ (IL)	76.46	74.12
+ Continuous Goal-Space PPO	78.65	76.61
Discrete Goal Space	74.50	70.35
Continuous Goal Space	78.65	76.61

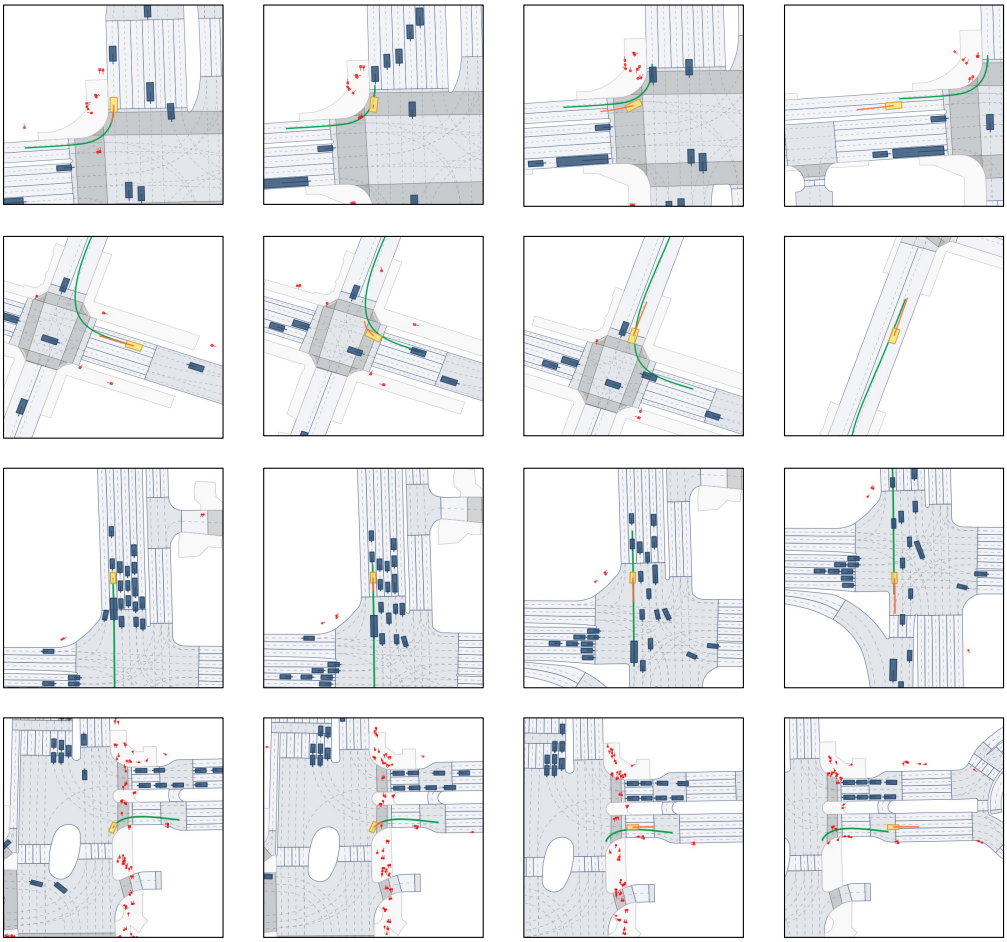

Figure 4: **Visualization of FlashPlanner closed-loop planning results on hard cases.** Each row represents a scenario at 0, 5, 10, and 15 seconds intervals. Each frame includes the future planning of the ego vehicle and the ground truth ego trajectory. FlashPlanner generates smooth and safe trajectories that closely align with expert driving behavior across diverse challenging scenarios.

architecture and data pipeline improve the IL baseline by 0.79/5.56 points in NR/R, with individual components ablated in Table 4 and Figure 3. Continuous goal-space PPO further improves the IL-pretrained FlashPlanner by 2.19/2.49 points in NR/R. Applying the same PPO framework to the original Diffusion Planner backbone and data pipeline also yields improvements, demonstrating that the RL gains are not specific to our architectural or data-pipeline modifications. Furthermore, continuous goal-space PPO outperforms a discrete variant based on K-means clustering of 1M expert endpoints by 4.15/6.26 points in NR/R, highlighting the advantage of continuous goal optimization over discrete goal selection.

6 Conclusion

In this work, we propose FlashPlanner, a goal-conditioned flow-matching planner with online RL fine-tuning for closed-loop autonomous driving. By decomposing the planning problem into a high-level goal predictor and a low-level trajectory generator, FlashPlanner enables efficient RL optimization in a compact goal space, naturally supports multi-modal trajectory generation, and provides an intuitive interface for controllability. We also provide a detailed analysis of existing design choices in diffusion-based methods and identify inherent redundancies, offering practical insights for model design. FlashPlanner achieves state-of-the-art closed-loop performance on both the nuPlan and interPlan benchmarks while delivering 166 FPS inference speed, demonstrating a more efficient framework for generative planning in autonomous driving.

Acknowledgements

This work was partly supported by National Science and Technology Major Project (2025ZD1606203) and Zhongguancun Academy (02012401).

References

[1] Bernhard Jaeger, Daniel Dauner, Jens Beißwenger, Simon Gerstenecker, Kashyap Chitta, and Andreas Geiger. Carl: Learning scalable planning policies with simple rewards. *arXiv preprint arXiv:2504.17838*, 2025.

[2] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17853–17862, 2023.

[3] Jie Cheng, Yingbing Chen, Xiaodong Mei, Bowen Yang, Bo Li, and Ming Liu. Rethinking imitation-based planners for autonomous driving. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14123–14130. IEEE, 2024.

[4] Jie Cheng, Yingbing Chen, and Qifeng Chen. Pluto: Pushing the limit of imitation learning-based planning for autonomous driving. *arXiv preprint arXiv:2404.14327*, 2024.

[5] Brian Yang, Huangyuan Su, Nikolaos Gkanatsios, Tsung-Wei Ke, Ayush Jain, Jeff Schneider, and Katerina Fragkiadaki. Diffusion-es: Gradient-free planning with diffusion for autonomous driving and zero-shot instruction following. *arXiv preprint arXiv:2402.06559*, 2024.

[6] Bencheng Liao, Shaoyu Chen, Haoran Yin, Bo Jiang, Cheng Wang, Sixu Yan, Xinbang Zhang, Xiangyu Li, Ying Zhang, Qian Zhang, et al. Diffusiondrive: Truncated diffusion model for end-to-end autonomous driving. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 12037–12047, 2025.

[7] 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 Computer Vision and Pattern Recognition Conference*, pages 1602–1611, 2025.

[8] Yinan Zheng, Ruiming Liang, Kexin Zheng, Jinliang Zheng, Liyuan Mao, Jianxiong Li, Weihao Gu, Rui Ai, Shengbo Eben Li, Xianyuan Zhan, et al. Diffusion-based planning for autonomous driving with flexible guidance. *arXiv preprint arXiv:2501.15564*, 2025.

[9] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex 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.

[10] Marcel Hallgarten, Julian Zapata, Martin Stoll, Katrin Renz, and Andreas Zell. Can vehicle motion planning generalize to realistic long-tail scenarios? In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5388–5395, 2024.

[11] Hongchen Li, Tianyu Li, Jiazhi Yang, Haochen Tian, Caojun Wang, Lei Shi, Mingyang Shang, Zengrong Lin, Gaoqiang Wu, Zhihui Hao, et al. Plannerrft: Reinforcing diffusion planners through closed-loop and sample-efficient fine-tuning. *arXiv preprint arXiv:2601.12901*, 2026.

[12] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[13] Matt Vitelli, Yan Chang, Yawei Ye, Ana Ferreira, Maciej Wołczyk, Błażej Osiński, Moritz Niendorf, Hugo Grimmett, Qiangui Huang, Ashesh Jain, et al. Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 897–904. IEEE, 2022.

[14] Zhiyu Huang, Haochen Liu, and Chen Lv. Gameformer: Game-theoretic modeling and learning of transformer-based interactive prediction and planning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3903–3913, 2023.

[15] 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*, 45(11):12878–12895, 2022.

[16] Shaoyu Chen, Bo Jiang, Hao Gao, Bencheng Liao, Qing Xu, Qian Zhang, Chang Huang, Wenyu Liu, and Xinggang Wang. Vadv2: End-to-end vectorized autonomous driving via probabilistic planning. *arXiv preprint arXiv:2402.13243*, 2024.

[17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[18] Xiaosong Jia, Zhenjie Yang, Qifeng Li, Zhiyuan Zhang, and Junchi Yan. Bench2drive: Towards multi-ability benchmarking of closed-loop end-to-end autonomous driving. *Advances in Neural Information Processing Systems*, 37:819–844, 2024.

[19] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[20] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[21] Tianyi Tan, Yinan Zheng, Ruiming Liang, Zexu Wang, Kexin Zheng, Jinliang Zheng, Jianxiong Li, Xianyuan Zhan, and Jingjing Liu. Flow matching-based autonomous driving planning with advanced interactive behavior modeling. *arXiv preprint arXiv:2510.11083*, 2025.

[22] Lingguang Wang, Ömer Şahin Taş, Marlon Steiner, and Christoph Stiller. Flowdrive: moderated flow matching with data balancing for trajectory planning. *arXiv preprint arXiv:2509.21961*, 2025.

[23] Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, Dragomir Anguelov, et al. Motiondiffuser: Controllable multi-agent motion prediction using diffusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9644–9653, 2023.

[24] Max Jiang, Yijing Bai, Andre Cornman, Christopher Davis, Xiukun Huang, Hong Jeon, Sakshum Kulshrestha, John Lambert, Shuangyu Li, Xuanyu Zhou, et al. Scenediffuser: Efficient and controllable driving simulation initialization and rollout. *Advances in Neural Information Processing Systems*, 37:55729–55760, 2024.

[25] Yunsong Zhou, Naisheng Ye, William Ljungbergh, Tianyu Li, Jiazhi Yang, Zetong Yang, Hongzi Zhu, Christoffer Petersson, and Hongyang Li. Decoupled diffusion sparks adaptive scene generation. *arXiv preprint arXiv:2504.10485*, 2025.

[26] Yu Gao, Anqing Jiang, Yiru Wang, Wang Jijun, Hao Jiang, Zhigang Sun, Heng Yuwen, Wang Shuo, Hao Zhao, and Sun Hao. Diffvla++: Bridging cognitive reasoning and end-to-end driving through metric-guided alignment. *arXiv preprint arXiv:2510.17148*, 2025.

[27] Anqing Jiang, Yu Gao, Zhigang Sun, Yiru Wang, Jijun Wang, Jinghao Chai, Qian Cao, Yuweng Heng, Hao Jiang, Yunda Dong, et al. Diffvla: Vision-language guided diffusion planning for autonomous driving. *arXiv preprint arXiv:2505.19381*, 2025.

[28] Yongkang Li, Kaixin Xiong, Xiangyu Guo, Fang Li, Sixu Yan, Gangwei Xu, Lijun Zhou, Long Chen, Haiyang Sun, Bing Wang, et al. Recogdrive: A reinforced cognitive framework for end-to-end autonomous driving. *arXiv preprint arXiv:2506.08052*, 2025.

[29] Shu Liu, Wenlin Chen, Weihao Li, Zheng Wang, Lijin Yang, Jianing Huang, Yipin Zhang, Zhongzhan Huang, Ze Cheng, and Hao Yang. Bridgedrive: Diffusion bridge policy for closed-loop trajectory planning in autonomous driving. *arXiv preprint arXiv:2509.23589*, 2025.

[30] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in neural information processing systems*, 35:5775–5787, 2022.

[31] Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 14754–14762, 2025.

[32] Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. *arXiv preprint arXiv:2502.02538*, 2025.

[33] Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, et al. Scene transformer: A unified architecture for predicting multiple agent trajectories. *arXiv preprint arXiv:2106.08417*, 2021.

[34] Dian Chen and Philipp Krähenbühl. Learning from all vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17222–17231, 2022.

[35] Chen Chen, Rui Qian, Wenze Hu, Tsu-Jui Fu, Jialing Tong, Xinze Wang, Lezhi Li, Bowen Zhang, Alex Schwing, Wei Liu, et al. Dit-air: Revisiting the efficiency of diffusion model architecture design in text to image generation. *arXiv preprint arXiv:2503.10618*, 2025.

[36] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34: 24261–24272, 2021.

[37] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

[38] Daniel Dauner, Marcel Hallgarten, Andreas Geiger, and Kashyap Chitta. Parting with misconceptions about learning-based vehicle motion planning. In *Conference on Robot Learning*, pages 1268–1281. PMLR, 2023.

[39] Oliver Scheel, Luca Bergamini, Maciej Wolczyk, Błażej Osiński, and Peter Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *Conference on Robot Learning*, pages 718–728. PMLR, 2022.

[40] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

[41] Zhengyang Geng, Mingyang Deng, Xingjian Bai, Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. *Advances in Neural Information Processing Systems*, 38: 75460–75482, 2026.

[42] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

A Design choices of FlashPlanner

We summarize the design choices of FlashPlanner, built upon Diffusion Planner, as follows.

- *Data Prediction*: Replace the diffusion training and inference paradigm with flow matching using a data-prediction objective and single-step inference.
- *Planning-only*: Focus exclusively on ego planning, removing traffic prediction network modules and their corresponding loss terms from the training objective.
- *Other-As-Ego*: Augment the training dataset with trajectory samples of non-ego vehicles extracted from existing scenarios.
- *MLP Decoder*: Replace the Transformer-based decoder with a compact MLP.
- *Light Encoder*: Reduce the number of self-attention layers in the encoder to one.
- (x, y, θ): Represent ego future states as (x, y, θ) instead of $(x, y, \cos\theta, \sin\theta)$.
- *Scen Balance*: Rebalance training scenarios to ensure sufficient coverage across diverse scenario types.

B Experimental Details

Baseline. We compare FlashPlanner against the following baselines.

- **Log Replay** tracks human trajectories using an LQR controller as the expert baseline.
- **IDM** [37] represents a classic car-following model widely used in traffic simulations.
- **PDM** [38] offers three variants: PDM-Closed generates trajectory proposals using IDM policies with varying hyperparameters and selects the optimal one through rule-based scoring; PDM-Open employs MLPs conditioned on centerlines; PDM-Hybrid combines both approaches.
- **GameFormer** [14] models interactive planning and prediction based on level-k game theory, with outputs refined through nonlinear optimization.
- **PLUTO** [4] extends PDM-Open with contrastive learning for enhanced environmental understanding, followed by rule-based post-processing.
- **UrbanDriver** [39] utilizes policy gradient optimization with PointNet-based polyline encoders and Transformers.
- **PlanTF** [3] represents a strong imitation learning baseline built on a Transformer architecture with efficient designs.
- **Diffusion Planner** [8] applies a diffusion transformer to generate ego trajectories conditioned on vectorized scene representations, with optional refinement.
- **PlannerRFT** [11] is a reinforcement fine-tuning framework that enhances Diffusion Planner's closed-loop performance through dual-branch reward-oriented optimization while preserving the original model architecture.

Metric. For evaluation, we conduct extensive experiments on both the Test14-hard and Val14 benchmarks under non-reactive (NR) and reactive (R) modes. The closed-loop score is adopted as our primary metric, which is a composite measure that aggregates progress, time-to-collision, speed-limit compliance, and comfort into a unified score ranging from 0 to 100, with higher values indicating better planning performance. Additionally, to further assess the robustness of FlashPlanner in long-tail scenarios, we also evaluate it on the interPlan benchmark [10], which is specifically designed to expose planners to rare and highly interactive scenarios that are not comprehensively covered in nuPlan. We follow the official interPlan split (interPlan10), which includes 80 scenarios. Based on the nuPlan framework, interPlan also adopts a 0 to 100 closed-loop score but extends the metric with penalties tailored for long-tail situations.

Pretrain. FlashPlanner is trained on 4 NVIDIA A100 80GB GPUs with a batch size of 2048 over 100 epochs, including a 5-epoch warmup phase. We employed the AdamW optimizer with a learning rate of 1e-3. The detailed hyperparameter configuration is presented in Table 6.

Table 6: Hyperparameters of FlashPlanner.

Parameter	Symbol	Value
Num. past timestamps	T_h	21
Num. future timestamps	T_f	4
Num. neighboring vehicles	N_d	32
Dim. neighboring vehicles	D_d	11
Num. static objects	N_s	5
Dim. static objects	D_s	10
Num. lanes	N_l	70
Num. points per polyline	P_l	20
Dim. lanes	D_l	12
Num. navigation lanes	N_r	25
Dim. hidden layer	D	192
Time sampler	-	Uniform
Inference step	n	1

RL Fine-tuning. FlashPlanner is fine-tuned on 8 NVIDIA A100 80GB GPUs over 10M environment steps. Hyperparameters for PPO optimization are provided in Table 7.

Table 7: Hyperparameters for PPO optimization.

Parameter	Value
Samples	10M
Initial Learning Rate	2.5×10^{-4}
Learning Rate Schedule	Cosine decay
Number of Envs.	1024
Env. Steps per Iteration	16
Batch Size	16384
Mini-batch Size	4096
Steps per Epoch	4
Epochs	2
Discount Factor	0.99
GAE λ	0.95
Clip Range ϵ	0.2
Max Gradient Norm	0.5

Inference. To ensure fair comparison and evaluate computational efficiency under practical deployment conditions, all inference experiments were performed on the same device with a single NVIDIA RTX A6000 GPU and an AMD EPYC 7542 32-Core CPU.

C Impact Statement

This work provides a stable generative planner for real-time decision making in autonomous driving. Autonomous driving is a composite system that integrates multiple modules, such as perception, prediction, and planning. Upstream components, especially perception, often dominate the computational budget and operate at relatively low throughput. In practical industrial deployments, accelerating the planning module with our proposed FlashPlanner may reduce the overall inference cost, freeing compute resources for other bottleneck components and improving end-to-end system efficiency.

D Limitation and Future Work

While FlashPlanner achieves state-of-the-art closed-loop performance with real-time inference speed (166 FPS) in the planning setting, several limitations remain. First, our current framework operates on post-perception representations rather than raw sensor inputs. Extending FlashPlanner to an

end-to-end setting—where the goal predictor and trajectory generator are jointly optimized with perception modules—remains an important direction for future work. Second, our reinforcement fine-tuning relies on the nuPlan closed-loop simulator, which inevitably introduces a sim-to-real gap. The reactive agent models in simulation may not fully capture the complexity and diversity of real-world driver behaviors, potentially limiting the transferability of learned goal policies to deployment. We also plan to investigate more realistic simulation environments and domain adaptation techniques to narrow the sim-to-real gap for RL fine-tuning.

E Additional Results

E.1 Efficiency Comparison

We compare the computational efficiency of FlashPlanner against other learning-based planners in Table 8. FlashPlanner achieves the smallest model size (4.60 MB), the lowest computational cost (0.48 GFLOPs), and the highest throughput (166 FPS), outperforming all baselines by a significant margin. Compared to Diffusion Planner, FlashPlanner is approximately $5\times$ smaller in model size, requires $3.2\times$ fewer FLOPs, and achieves $12\times$ higher throughput, demonstrating its suitability for real-time deployment.

Table 8: **Efficiency comparison of learning-based planners.** FlashPlanner achieves the best efficiency across all metrics, enabling real-time inference with minimal computational overhead.

Method	Model Size (MB)	FLOPs (G)	Throughput (FPS)
PlanTF	7.72	0.67	97
PLUTO w/o refine.	16.18	0.73	47
Diffusion Planner	23.05	1.54	13
FlashPlanner (Ours)	4.60	0.48	166

E.2 Ablation on Ego State Representation and Time Sampler

We ablate the ego state representation and time sampler choices in Table 9.

Ego State. Prior works represent the ego's future state as $(x, y, \cos\theta, \sin\theta)$, whereas we use (x, y, θ). As shown in Table 9, although $(x, y, \sin\theta, \cos\theta)$ achieves a marginally higher NR score (76.96 vs. 76.46), it leads to a notable degradation in the reactive setting (71.32 vs. 74.12). We attribute this to the redundant trigonometric constraint ($\sin^2\theta + \cos^2\theta = 1$), which is brittle under distribution shift in closed-loop reactive simulations.

Time Sampler. We evaluate three time samplers for flow matching: Uniform $\mathcal{U}(0, 1)$, LogNormal(-0.4, 1.0) that emphasizes intermediate times [40], and Beta(1.0, 1.5) that biases toward early times [19]. As reported in Table 9, the results show broad robustness across samplers, with the simplest uniform sampling achieving the best overall performance, particularly in the reactive setting (74.12).

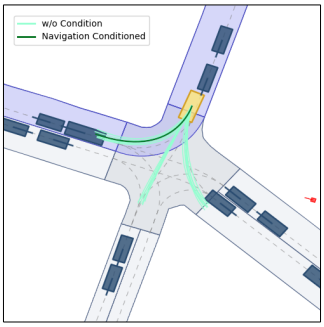

Figure 5: **Multi-modal planning behaviors of FlashPlanner.** Without navigation guidance, FlashPlanner generates diverse trajectory distributions at an intersection, covering left turn, right turn, and straight maneuvers. With navigation constraints, the planner reliably follows the specified route.

Table 9: **Ablation of Ego State and Time Sampler on nuPlan Test14-hard.** The baseline FlashPlanner$_{\text{pre}}$ uses (x, y, θ) representation with uniform time sampling.

Component	Setting	NR	R
Baseline	FlashPlanner$_{\text{pre}}$	76.46	74.12
Ego State	$(x, y, \sin\theta, \cos\theta)$	76.96	71.32
Time Sampler	LogNormal(-0.4, 1.0)	76.06	74.06
	Beta(1.0, 1.5)	75.51	70.76

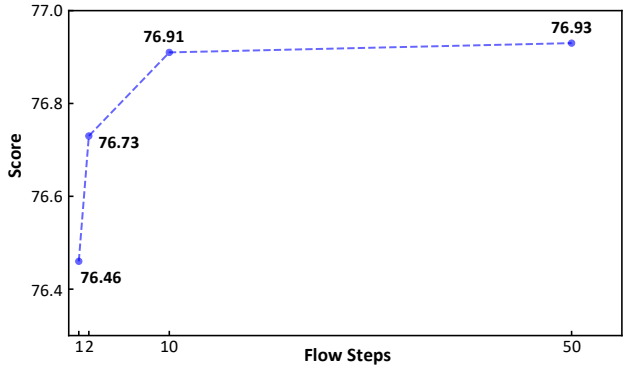

Figure 6: **Ablation study on inference steps (Test14-hard NR).** FlashPlanner achieves strong performance with a single inference step and exhibits monotonic improvement as more steps are used, demonstrating effective single-step generation with optional refinement capability.

E.3 Comparison of Generation Paradigms

We compare our data prediction formulation against alternative single-step generation paradigms in Table 10. Specifically, we evaluate MeanFlow [41], which learns the mean velocity field for one-step generation, and Consistency Models [42], which enforce self-consistency along the probability flow ODE. All methods share the same architecture and are trained for 100 epochs with one-step inference. As shown in Table 10, our data prediction formulation consistently outperforms both alternatives across all benchmarks and evaluation modes.

Table 10: **Performance Comparison on nuPlan.** All methods were trained for 100 epochs and evaluated with one-step generation. Data prediction achieves the best performance across all settings.

Method	Val14		Test14-hard	
	NR	R	NR	R
MeanFlow	86.26	77.38	70.53	65.00
Consistency Models	80.32	73.88	69.22	61.61
Data Prediction	89.19	84.6	76.46	74.12

E.4 Effect of Inference Steps

Inference Steps. To numerically solve ODEs, we use the Euler method, which requires a pre-specified number of steps (Equation (6)). As shown in Figure 6, FlashPlanner enables one-step generation of high-quality trajectories for autonomous driving, while supporting monotonic refinement with additional inference steps. The closed-loop score on Test14-hard NR improves steadily as the number of inference steps increases, demonstrating that our data-prediction formulation preserves single-step efficiency while benefiting from multi-step refinement when computational budget permits.

E.5 Scenario Balancing

As discussed in Appendix Section A, we rebalance training scenarios to ensure sufficient coverage across diverse scenario types. Figure 7 illustrates the distribution of scenario types in the training set after balancing. The original nuPlan training set exhibits a highly imbalanced distribution, with common scenarios (e.g., lane following) dominating and rare scenarios (e.g., U-turns, unprotected left turns) being severely underrepresented. Our scenario balancing strategy upsamples rare scenario types and downsamples over-represented ones, resulting in a more uniform distribution that improves the planner's generalization to diverse and challenging driving situations.

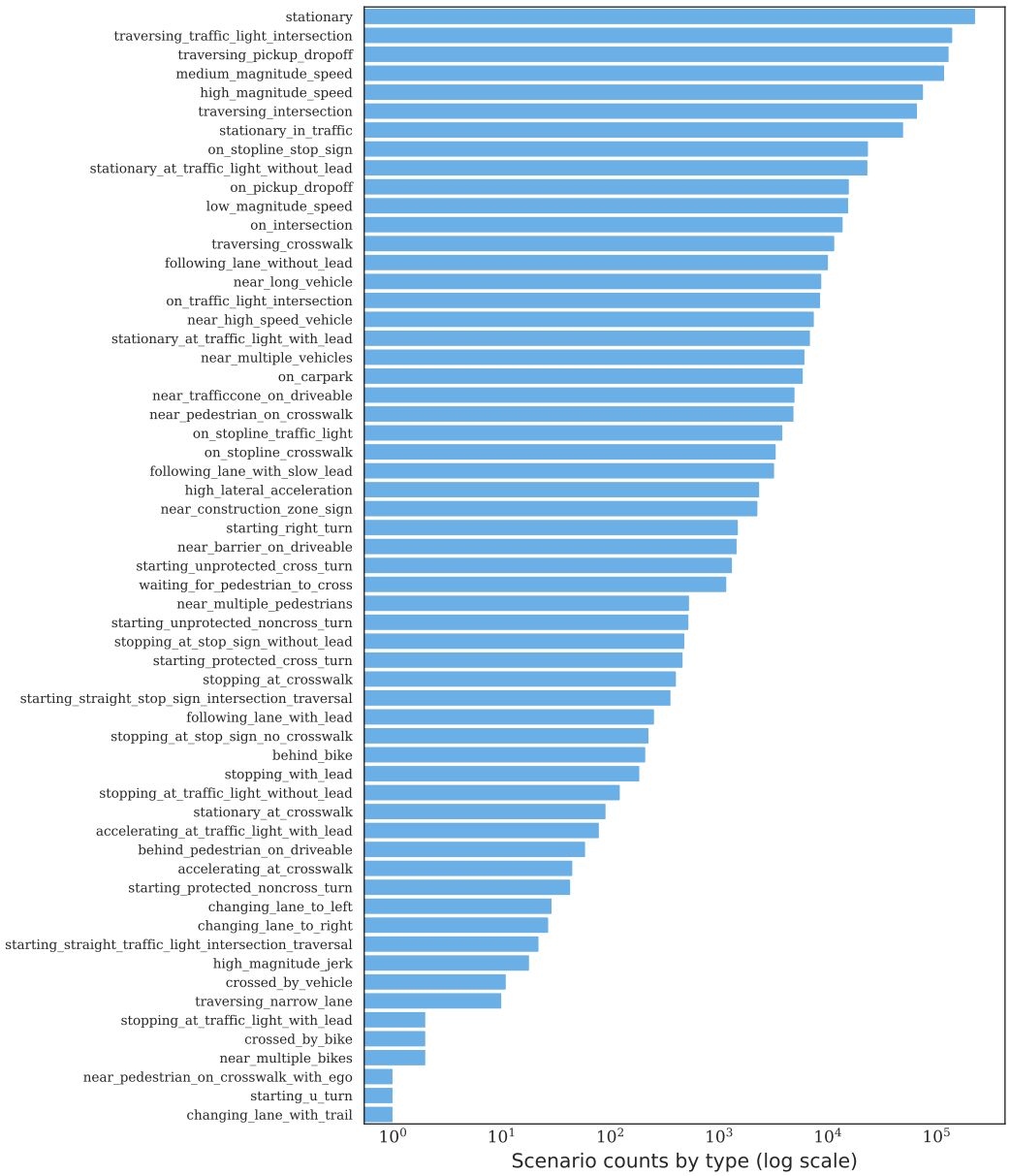

Figure 7: **Distribution of scenario types in the training set.** The original nuPlan training set exhibits a highly imbalanced distribution across scenario types. Our scenario balancing strategy creates a more uniform distribution, ensuring sufficient coverage of rare and challenging scenarios during training.

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 claim four main contributions: (1) a goal-conditioned flow-matching planner that decouples intent prediction from trajectory generation, (2) a reinforcement fine-tuning framework optimizing the goal predictor via PPO in low-dimensional goal space, (3) systematic analysis and pruning of redundant components in diffusion-based planners, and (4) SOTA performance on nuPlan and interPlan benchmarks with 6× speedup. These claims are supported by experimental results in Table 1 (SOTA performance), Table 2 (efficiency comparison), Tables 3-4 (ablation studies), and the method description in Section 4.

   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: Refer to Section D.

   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 include formal theoretical results.

   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: The paper provides detailed architectural descriptions (Section 4.5), hyperparameters for both pretraining (Table 6) and RL fine-tuning (Table 7), training infrastructure (4 A100 GPUs for pretraining, 8 A100 GPUs for RL), optimizer choices (AdamW with lr=1e-3), dataset details (1M frames for pretraining, low-score scenarios for RL), and the evaluation protocol.

   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: [Yes].

   Justification: We will open-source the code upon acceptance.

   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: The paper provides comprehensive experimental details in Section 5.1 and Appendix B.

   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: [Yes].

   Justification: Refer to Section 5

   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 training on 4 NVIDIA A100 80GB GPUs for pretraining (2.5 hours) and 8 NVIDIA A100 80GB GPUs for RL fine-tuning (8 hours, 10M env steps). Inference experiments use a single NVIDIA RTX A6000 GPU with AMD EPYC 7542 32-Core CPU.

   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 uses publicly available datasets (nuPlan), does not involve human subjects, and focuses on improving autonomous driving planning in simulation environments.

   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: [Yes].

Justification: Refer to Section C

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: No pretrained LLMs are used.

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 properly cites the nuPlan dataset, interPlan benchmark, and all baseline methods used for comparison.

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: No new assets are introduced.

    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: We have no human subjects.

    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: We have no human subjects.

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: LLM is not a core component in this paper.

    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.