FlowR2A: Learning Reward-to-Action Distribution for Multimodal Driving Planning

Xirui Li[1] **Zhe Liu**[1†] **Xiaoqing Ye**[2,3*] **Wenhua Han**[2]
Yifeng Pan[2] **Junyu Han**[2] **Hengshuang Zhao**[1*]
[1]The University of Hong Kong [2]Changan Automobile [3]Wuhan University
[†]project lead [*]corresponding author
https://lixirui142.github.io/flowr2a-ad

Abstract

Multimodal driving planning faces a long-standing tension between two paradigms: scoring-based methods benefit from dense reward supervision but are confined to a fixed action vocabulary, while anchor-based methods generate proposals dynamically yet suffer from sparse supervision constrained to a single ground-truth trajectory. In this work, we propose FlowR2A, which resolves this tension by reframing simulation-based rewards from discriminative targets into generative conditions. By learning the reward-conditioned action distribution from dense trajectory-reward pairs with a flow-matching decoder, FlowR2A unifies the dense supervision of scoring-based methods with the proposal generation of anchor-based methods in a single generative model, forcing the model to internalize the correlation between an action and its outcomes in safety, progress, comfort, and rule compliance. To balance hard safety constraints against soft progress objectives, we introduce fine-grained per-timestep reward conditioning and reward noise augmentation. The generative formulation naturally supports controllable test-time sampling via reward guidance and anchored sampling, producing high-quality proposals. FlowR2A achieves state-of-the-art results on the NAVSIM v1 and v2 benchmarks, with multimodal proposals of substantially higher quality than prior methods.

1 Introduction

End-to-end autonomous driving (E2E-AD) has emerged as a promising paradigm that maps raw sensor input directly to planning output through a differentiable model [20, 22, 7]. A typical E2E-AD model consists of a perception encoder that extracts scene observations from sensor input and a plan decoder that produces planning actions on top of these observations. Early efforts predict a single action that imitates the human trajectory [20, 22, 7, 6, 47, 30, 34]. Given the inherent uncertainty and multimodality of driving behavior, recent research has shifted toward multimodal planning [5, 32, 35, 17], where the final action is selected from multiple proposals.

Existing multimodal planners fall into two paradigms: scoring-based and anchor-based methods, illustrated in Fig. 1. Scoring-based methods [5, 32, 28, 49, 33] use a large, fixed action vocabulary as candidates and train a scorer to evaluate each candidate with simulation-based reward labels, selecting the best one as output. The key insight of this paradigm is that dense reward supervision over the entire action vocabulary provides a rich, comprehensive signal about the relationship between actions and their outcomes. However, scoring-based methods are fundamentally discriminative: they learn $p(r|a)$ to rank actions, but cannot transfer this knowledge to generate new proposals. Their output is therefore constrained to the fixed vocabulary, limiting adaptability to dynamic real-world scenarios.

Anchor-based methods [17, 35, 52, 24] address this rigidity by decoding proposals dynamically from a set of action anchors and applying a winner-takes-all loss that supervises only the proposal closest

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

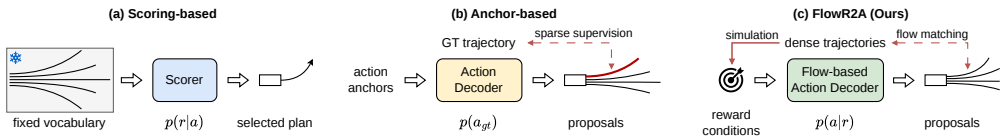


Figure 1: Comparison of multimodal planning paradigms. (a) Scoring-based methods select from a large fixed trajectory vocabulary. (b) Anchor-based methods decode multiple proposals from action anchors and sparsely supervise them with GT. (c) Our method FlowR2A learns the reward-conditioned action distribution $p(a|r)$ from dense trajectories, where a is the action and r is the reward. Red lines indicate training supervision.

to the single ground-truth (GT) trajectory. While this enables scene-adaptive proposals, the sparse GT supervision introduces two fundamental limitations. First, many anchors receive no training signal per scene, resulting in low-quality or degenerate proposals as shown in our experiments (Tab. 3, Fig. 6). Second, targeting the single GT trajectory inherits well-known imitation learning pathologies [9, 10], including shortcut learning from ego status and unawareness of action consequences.

The two paradigms reveal a common tension: dense supervision and generative proposal modeling have so far been mutually exclusive. Scoring-based methods enjoy dense action-reward supervision but are bounded by their discriminative nature. Anchor-based methods can generate proposals, yet are supervised by a single GT trajectory per scene. This raises a natural question: can we combine dense reward supervision with generative proposal modeling in a single framework?

In this work, we propose FlowR2A, a multimodal planning framework that learns the reward-conditioned action distribution $p(a|r)$ from dense trajectory-reward pairs, resolving the above tension with a simple reframing. A dense action vocabulary can be paired with simulation-based rewards characterizing safety, progress, comfort, and rule compliance, yielding dense action-reward pairs that span the action space. While prior work [32, 49] treats these rewards as discriminative targets to predict, we instead treat them as conditions to learn the generative reward-to-action distribution. Under this formulation, each action-reward pair becomes a valid training sample, and the model is forced to internalize the correlation between an action and its outcomes. We construct fine-grained reward signals from rule-based simulation [10] and train a flow-matching-based [38, 36, 12] action decoder to recover clean trajectories from noise under reward conditioning. At inference, this generative formulation enables classifier-free guidance conditioned on high rewards and anchored sampling from any reference trajectory, yielding diverse, high-quality proposals.

A practical challenge arises in conditioning on high rewards. High-reward actions often sit close to the boundary of the feasible region, requiring the generative decoder to approach it without crossing into the infeasible side. We address this through two designs. First, we replace key safety and compliance rewards with per-timestep counterparts, which provide a more general and finer-grained signal that sharpens the hard constraints. Second, we corrupt the continuous reward values with minor Gaussian noise during training, which acts as label smoothing and prevents the decoder from over-relying on them. Together, these designs effectively balance the conflicting objectives.

We evaluate FlowR2A on both NAVSIM v1 and v2 benchmarks [10, 3], achieving state-of-the-art performance. More importantly, our model produces multimodal proposals of substantially higher quality than prior methods, validating the benefit of modeling the entire conditional action distribution. Our contributions are:

- A new paradigm for multimodal driving planning that learns the reward-to-action distribution $p(a|r)$, unifying the dense supervision of scoring-based methods with the generation ability of anchor-based methods, and naturally supporting controllable test-time sampling.
- A reward construction recipe that effectively balances conflicting objectives with fine-grained per-timestep reward conditioning and reward noise augmentation.
- State-of-the-art results on the NAVSIM benchmark with higher proposal quality than previous methods, validating the advantage of learning the full conditional action distribution.

2 Preliminaries

Flow matching [36, 1, 38] provides a general perspective on generative modeling by defining a probability path between the noise distribution and the data distribution. It defines a forward process that linearly combines the data sample $\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$ and the noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$ into a noisy sample

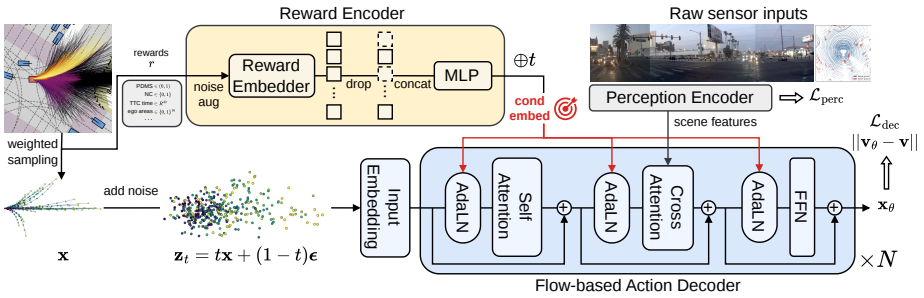


Figure 2: FlowR2A structure and training pipeline. We randomly sample action-reward pairs to produce noisy samples $\boldsymbol{z}_t$. The reward encoder embeds rewards r into a condition embedding. The flow-based action decoder predicts the clean sample based on scene features from the perception encoder and the condition injected via AdaLN, which is supervised by the velocity-matching loss.

$\boldsymbol{z}_t = a_t\boldsymbol{x} + b_t\boldsymbol{\epsilon}$, where a_t, b_t are noise schedules at time $t \in [0, 1]$. Following rectified flow [38], we use a straight path with a linear schedule,

$$\boldsymbol{z}_t = t\boldsymbol{x} + (1-t)\boldsymbol{\epsilon}, \tag{1}$$

so $\boldsymbol{z}_0 = \boldsymbol{\epsilon}$ is pure noise and $\boldsymbol{z}_1 = \boldsymbol{x}$ is the clean sample. The flow velocity is the time derivative of $\boldsymbol{z}_t$,

$$\boldsymbol{v} = \frac{d\boldsymbol{z}_t}{dt} = \boldsymbol{x} - \boldsymbol{\epsilon}. \tag{2}$$

Flow-based methods [12, 36] train a model $\boldsymbol{v}_\theta(\boldsymbol{z}_t, t)$ to match $\boldsymbol{v}$ via the velocity-matching loss

$$\mathcal{L} = \mathbb{E}_{t,\boldsymbol{x},\boldsymbol{\epsilon}} \|\boldsymbol{v}_\theta(\boldsymbol{z}_t, t) - \boldsymbol{v}\|^2. \tag{3}$$

At inference, we draw clean samples by solving the ODE $d\boldsymbol{z}_t = \boldsymbol{v}_\theta(\boldsymbol{z}_t, t)\, dt$ from $\boldsymbol{z}_0 \sim \mathcal{N}(0, \boldsymbol{I})$ at $t = 0$ to $\boldsymbol{z}_1$ at $t = 1$. We use a 20-step Euler solver [12, 13] in this work.

3 Method

FlowR2A is a multimodal planning method that learns the reward-conditioned action distribution $p(a|r)$ (with scene context s omitted from the condition for brevity). We construct fine-grained reward signals that capture safety, progress, comfort, and rule compliance, and pair them with a dense action vocabulary to form training samples (a, r) that span the action space. The end-to-end model couples a perception encoder that produces scene features, a reward encoder that maps reward signals into a condition embedding, a flow-based action decoder that generates trajectory proposals, and a mode selector that picks the final output. The model is trained with a flow-matching objective on dense (a, r) pairs and supports controllable test-time sampling via reward guidance and anchored sampling. Fig. 2 shows the model structure and training pipeline; Fig. 3 shows the inference pipeline.

3.1 Reward Condition

Dense Action-Reward Pairs. Following scoring-based methods [5, 32, 33], we discretize the continuous action space into a dense action vocabulary $\mathcal{V}_a$ containing 8192 four-second trajectories clustered from 700K nuPlan trajectories [2]. For every training scene, we simulate each vocabulary trajectory using the NAVSIM simulator [10] and record the resulting reward labels, yielding dense action-reward pairs (a, r) that span the action space and serve as training samples for our generative model of $p(a|r)$.

Fine-grained Reward Signals. NAVSIM evaluates a plan trajectory by closed-loop simulation and reports a PDM score that summarizes the overall quality [10]. A single scalar score is too coarse for conditioning. For example, given a zero-score condition, the model cannot tell whether the action collides with another agent or simply stays stationary. We therefore expose the underlying submetrics, namely no at-fault collisions (NC), drivable area compliance (DAC), driving direction compliance (DDC), traffic light compliance (TLC), ego progress (EP), time to collision (TTC), lane keeping (LK), and history comfort (HC). These metrics cover safety (NC, TTC), rule compliance (DAC, DDC, TLC), progress (EP), and comfort (LK, HC). Together with the PDM score, they form a fine-grained signal that enables the model to resolve the conditional action distribution with high fidelity.

Balancing Hard and Soft Objectives. Hard constraints and soft objectives conflict under naive high-reward conditioning. High-reward actions tend to sit on the decision boundary of binary hard constraints, which generative models smooth across in continuous action space. Naive conditioning on a high reward pushes the decoder toward aggressive proposals that sacrifice hard constraints to maximize progress. We confirm this empirically. Without our refinements, increasing the target reward at inference degrades overall performance rather than improving it (Fig. 5).

We address this with two designs that strengthen hard constraints and soften continuous signals. First, we replace the binary TTC and DAC labels with per-timestep arrays. The TTC-time array stores the projected collision time at each future timestep within a max detection horizon, and the ego-area array [17] marks whether the ego is on-road and on-route at each timestep. Both arrays preserve the hard constraint at higher temporal resolution, providing a stronger conditioning signal than a single binary label. Second, we corrupt the continuous rewards (EP and PDM score) with minor Gaussian noise during training, treating them as random variables $\tilde{r}_k \sim \mathcal{N}(r_k, \sigma^2)$ with noise scale σ, which acts as label smoothing and prevents the decoder from over-relying on these signals. The final reward set is $\mathcal{R} = \{r_k\}$, comprising the two per-timestep arrays and the seven other scalar metrics.

3.2 Model Architecture

Perception Encoder. We adopt Transfuser [7] as the perception backbone to encode multi-view images and a bird's-eye-view (BEV) LiDAR feature map. We combine the backbone output with the encoded ego status and driving command to obtain a set of scene features $\boldsymbol{s}$ comprising context tokens and agent tokens. $\boldsymbol{s}$ is supervised with auxiliary losses on agent detection and BEV semantic segmentation. We also train an imitation learning head with GT trajectories, which provides an optional anchor at inference (Sec. 3.4). Together these supervisions form the perception loss $\mathcal{L}_{\text{perc}}$.

Reward Encoder. The reward encoder maps the heterogeneous reward signals in $\mathcal{R}$ into a single condition embedding $\boldsymbol{r}_c$ that is fed to the action decoder. Each reward r_k is independently mapped into a feature embedding $\boldsymbol{r}_k^{\text{emb}}$ by a per-reward embedder, and the embeddings are concatenated and passed through an MLP to produce $\boldsymbol{r}_c$. To enable test-time classifier-free guidance [19] and conditioning on arbitrary subsets of rewards, we randomly replace each $\boldsymbol{r}_k^{\text{emb}}$ with a per-reward null token $\boldsymbol{n}_k$ during training. Overall, the reward encoder is formulated as,

$$\begin{aligned} \boldsymbol{r}_k^{\text{emb}} &= \begin{cases} \boldsymbol{n}_k, & \text{if drop } r_k \\ \text{Embed}(r_k), & \text{otherwise} \end{cases} \\ \boldsymbol{r}_c &= \text{MLP}(\text{concat}[\boldsymbol{r}_k^{\text{emb}}]). \end{aligned} \tag{4}$$

The continuous-reward noise augmentation introduced above is applied to r_k before embedding.

Flow-based Action Decoder. The reward-conditioned formulation is agnostic to the choice of generative model. We adopt flow matching because it models continuous action distributions well and natively supports inference-time guidance, which enables reward-guided proposal sampling. We instantiate the action decoder as a flow-based generative model over the continuous action space. As described in Sec. 2, it learns to recover clean trajectories $\boldsymbol{x}$ from noisy inputs $\boldsymbol{z}_t$ during training and generates action proposals through iterative denoising at inference. Following JiT [29], we adopt $\boldsymbol{x}$-prediction, where the decoder predicts a clean sample $\boldsymbol{x}_\theta(\boldsymbol{z}_t, t, c)$ from the noisy input $\boldsymbol{z}_t$, time t, and the condition $c = (\boldsymbol{s}, \boldsymbol{r}_c)$ comprising scene features and the reward embedding. Training uses the velocity-matching loss,

$$\mathcal{L}_{\text{dec}} = \mathbb{E}_{t,\boldsymbol{x},\boldsymbol{\epsilon}} \|\boldsymbol{v}_\theta(\boldsymbol{z}_t, t, c) - \boldsymbol{v}\|^2, \tag{5}$$

where the predicted $\boldsymbol{x}_\theta$ is converted to velocity by $\boldsymbol{v}_\theta = (\boldsymbol{x}_\theta - \boldsymbol{z}_t)/(1-t)$ based on Eqs. 1 and 2.

The decoder embeds the noisy trajectory $\boldsymbol{z}_t$ into a sequence of tokens with sinusoidal positional encoding, then applies four stacked transformer blocks. Each block consists of self-attention across trajectory tokens, cross-attentions to the scene features $\boldsymbol{s}$ (context and agent tokens), and a feed-forward layer, with residual connections around each module. The reward condition embedding is injected via adaptive layer normalization (AdaLN) [23, 44, 43], using the concatenation of $\boldsymbol{r}_c$ and the time embedding $\boldsymbol{t}$ as the modulation signal. The final block outputs the predicted clean sample $\boldsymbol{x}_\theta(\boldsymbol{z}_t, t, c)$.

Mode Selector. The mode selector ranks the proposals from the action decoder and returns the highest-scoring trajectory as the final output. Following scoring-based methods [32, 49], it is a

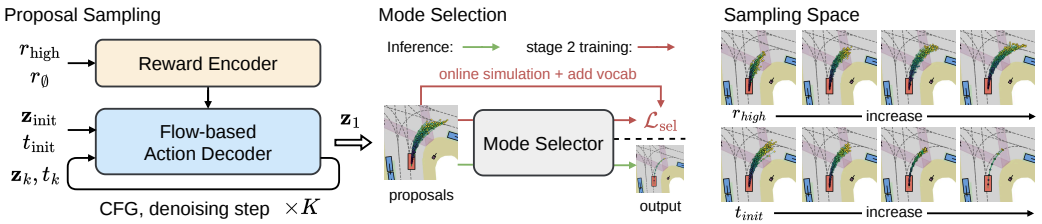


Figure 3: FlowR2A inference pipeline. (Left) The action decoder samples each proposal by denoising from a noisy anchor at t_{init} under CFG with the high-reward condition r_{high}, together producing multiple proposal candidates. (Middle) The mode selector ranks proposals at inference and is supervised by simulated GT labels during stage-2 training. (Right) Sampling space spanned by r_{high} and t_{init}. See Fig. 9 for the full version.

lightweight two-layer transformer that attends to the scene features and predicts a set of NAVSIM subscores through shallow heads, which are aggregated into a single ranking score. We supervise the heads with a multi-head prediction loss $\mathcal{L}_{\text{sel}}$ that matches the predicted subscores to their ground-truth values. See App. F.2.4 for full details.

3.3 Training

We train FlowR2A in two stages. The first stage trains the whole model end-to-end. The second stage finetunes the mode selector on proposals from the frozen decoder.

Training the Action Decoder (Stage 1). For each step, we sample action-reward pairs (a, r) from $\mathcal{V}_a$ with weight inversely proportional to the score density, since vocabulary scores are heavily skewed toward zero and rare high-quality samples would otherwise be drowned out. The trajectory is corrupted by Gaussian noise at $t \sim \text{Uniform}(0, 1)$, fed to the decoder with the reward condition, and supervised by $\mathcal{L}_{\text{dec}}$. We train the full model jointly with the perception and selector losses as auxiliary objectives that refine the scene features,

$$\mathcal{L}_{\text{train}} = \mathcal{L}_{\text{dec}} + w_{\text{perc}}\mathcal{L}_{\text{perc}} + w_{\text{sel}}\mathcal{L}_{\text{sel}}. \tag{6}$$

Training the Mode Selector (Stage 2). The stage-1 selector sees only vocabulary trajectories, which differ from decoder proposals at inference. In stage-2 training, we close this gap by freezing all other components and optimizing the selector with $\mathcal{L}_{\text{sel}}$ on online proposals labeled via simulation. Since the reward-guided decoder is biased toward high-quality samples, we mix in random vocabulary trajectories to keep the selector calibrated across the full quality range.

3.4 Inference

At inference, FlowR2A samples action proposals by solving the ODE $d\boldsymbol{z}_t = \boldsymbol{v}_\theta(\boldsymbol{z}_t, t, c)\, dt$ with the trained flow-based decoder. Built on the generative formulation, the decoder exposes a controllable test-time sampling interface over reward target and anchor noise level, instantiated via classifier-free guidance [19] and zero-shot editing [41]. Fig. 3 overviews the inference pipeline.

Reward Guidance. We use classifier-free guidance (CFG) [19] to steer the decoder toward the high-reward region of $p(a|r)$,

$$\boldsymbol{v}_g = \boldsymbol{v}_\theta(\boldsymbol{z}_t, t, r_\emptyset) + w_g\big(\boldsymbol{v}_\theta(\boldsymbol{z}_t, t, r_{\text{high}}) - \boldsymbol{v}_\theta(\boldsymbol{z}_t, t, r_\emptyset)\big), \tag{7}$$

where w_g is the CFG scale and $r_\emptyset$ is the empty condition obtained by setting every $\boldsymbol{r}_k^{\text{emb}} = \boldsymbol{n}_k$ in Eq. 4. The guided velocity $\boldsymbol{v}_g$ replaces $\boldsymbol{v}_\theta$ in the denoising step, so $w_g > 1$ amplifies the reward direction and pulls the sample toward high-quality actions. We instantiate r_{high} on a subset of $\mathcal{R}$ (App. F.4.3), and fix each reward entry to its maximal value except the target PDM score, which is left as a sampling control to be set by the strategy below. In what follows, sampling r_{high} refers to sampling this target score.

Anchored Sampling. The decoder also supports zero-shot anchored sampling [41]. Given any trajectory $\boldsymbol{x}_{\text{anchor}}$ as the anchor and an initial denoising time $t_{\text{init}} \in [0, 1]$, we form a noisy sample by reusing the forward process of Eq. 1,

$$\boldsymbol{z}_{\text{init}} = t_{\text{init}}\boldsymbol{x}_{\text{anchor}} + (1 - t_{\text{init}})\boldsymbol{\epsilon}, \tag{8}$$

Table 1: Results with closed-loop metrics on NAVSIM v1 `navtest` benchmark. The performance of our method is averaged over three inferences. Following recent works [52], we compare against methods using lightweight image backbones ResNet-34 [18] and V2-99 [26]. Methods are grouped by single-proposal and multi-proposal. For our single-proposal entry, r_{high} and t_{init} are fixed to representative values rather than randomly sampled (App. F.4.2).

Method	# Proposals	Img. Backbone	NC ↑	DAC ↑	TTC ↑	Comf. ↑	EP ↑	**PDMS ↑**
UniAD [20]	1	ResNet-34	97.8	91.9	92.9	100	78.8	83.4
Transfuser [7]	1	ResNet-34	97.7	92.8	92.8	100	79.2	84.0
PARA-Drive [47]	1	ResNet-34	97.9	92.4	93.0	99.8	79.3	84.0
DRAMA [50]	1	ResNet-34	98.0	93.1	94.8	100	80.1	85.5
ARTEMIS [14]	1	ResNet-34	98.3	95.1	94.3	100	81.4	87.0
FlowR2A (Ours)	1	ResNet-34	**98.6**	**97.3**	**95.3**	100	**84.9**	**90.0**
VADv2 [5]	8192	ResNet-34	97.2	89.1	91.6	100	76.0	80.9
Hydra-MDP [32]	8192	ResNet-34	98.3	96.0	94.6	100	78.7	86.5
Hydra-MDP++ [28]	8192	ResNet-34	97.6	96.0	93.1	100	80.4	86.6
Hydra-MDP [32]	8192	V2-99	98.4	97.8	93.9	100	86.5	90.3
Hydra-MDP++ [28]	8192	V2-99	98.6	**98.6**	95.1	100	85.7	91.0
DriveSuprim [49]	8192	ResNet-34	97.8	97.3	93.6	100	86.7	89.9
DiffusionDrive [35]	20	ResNet-34	98.2	96.2	94.7	100	82.2	88.1
WoTE [31]	256	ResNet-34	98.5	96.8	94.9	99.9	81.9	88.3
GoalFlow [48]	256	V2-99	98.4	98.3	94.6	100	85.0	90.3
DiffusionDriveV2 [52]	800	ResNet-34	98.3	97.9	94.8	99.9	87.5	91.2
iPad [17]	64	ResNet-34	98.6	98.3	94.9	100	88.0	91.7
FlowR2A (Ours)	60	ResNet-34	**98.8**	98.0	**96.0**	100	**90.1**	**92.8**

Table 2: Results with closed-loop metrics on NAVSIM v2 `navtest` benchmark. The performance of our method is averaged over three inferences. Following recent works [52], we compare against methods using lightweight image backbones ResNet-34 [18] and V2-99 [26].

Method	Img. Backbone	NC ↑	DAC ↑	DDC ↑	TLC ↑	EP ↑	TTC ↑	LK ↑	HC ↑	EC ↑	**EPDMS ↑**
Ego Status MLP	ResNet-34	93.1	77.9	92.7	99.6	86.0	91.5	89.4	98.3	85.4	64.0
Transfuser [7]	ResNet-34	96.9	89.9	97.8	99.7	87.1	95.4	92.7	98.3	**87.2**	76.7
Hydra-MDP++ [28]	ResNet-34	97.2	97.5	99.4	99.6	83.1	96.5	94.4	98.2	70.9	81.4
DriveSuprim [49]	ResNet-34	97.5	96.5	99.4	99.6	88.4	96.6	95.5	98.3	77.0	83.1
ARTEMIS [14]	ResNet-34	98.3	95.1	98.6	99.8	81.5	97.4	96.5	98.3	-	83.1
Hydra-MDP++ [28]	V2-99	98.4	98.0	99.4	99.8	87.5	97.7	95.3	98.3	77.4	85.1
DriveSuprim [49]	V2-99	97.8	97.9	**99.5**	**99.9**	90.6	97.1	**96.6**	98.3	77.9	86.0
FlowR2A (Ours)	ResNet-34	**98.9**	**98.1**	99.1	99.7	**91.5**	**98.5**	95.0	98.3	65.2	**88.9**

and start the denoising ODE from $\boldsymbol{z}_{\text{init}}$ at $t = t_{\text{init}}$ instead of from pure noise. The remaining denoising steps run under the reward condition, biasing the output toward the anchor's coarse structure. Unlike anchor-based methods [35] that train on a fixed anchor set, our decoder accepts any trajectory as anchor at inference without additional training. Anchored sampling is optional, and the decoder generates high-quality proposals from pure noise as well. By default, we use the IL head output as the anchor.

Sampling Strategy. Reward guidance and anchored sampling together define a sampling space spanned by the target score and the initial noise level. The PDM score in r_{high} relates to the target progress level. The initial time t_{init} controls anchor adherence, ranging from $t_{\text{init}} = 0$ (pure-noise sampling) to $t_{\text{init}} = 1$ (output equal to the anchor). To produce diverse proposals, we sample both controls uniformly per proposal, drawing the target score from $\text{Uniform}(s_{\min}, s_{\max})$ and t_{init} from $\text{Uniform}(t_{\min}, t_{\max})$. The resulting candidates are scored by the mode selector, and the highest-scoring trajectory is returned as the final output.

4 Experiments

We evaluate FlowR2A on the NAVSIM v1 [10] and v2 [3] benchmarks against scoring-based and anchor-based multimodal planners to validate that learning the reward-conditioned action distribution $p(a|r)$ yields both higher overall performance and substantially better proposal quality.

Dataset. NAVSIM [10] is built upon OpenScene [8], a compact redistribution of the nuPlan dataset [2], and curates real-world non-trivial driving scenes where the future plan cannot be directly inferred

Table 3: Quantitative comparison of proposal quality. We report PDMS, EP, and TTC on `navtest` when selecting among different numbers of proposal candidates. Mean and standard deviation of all proposals' scores are reported to reflect the average quality of proposals. Ours generates 64 proposals in total to match iPad [17].

	1 Proposal			4 Proposals			All Proposals			Mean over All Proposals		
Method	PDMS	EP	TTC	PDMS	EP	TTC	PDMS	EP	TTC	PDMS	EP	TTC
DiffusionDrive [35]	69.7	65.9	78.6	85.3	79.0	93.6	88.1	82.2	94.7	60.3$\pm$33.1	57.5$\pm$31.6	73.1$\pm$28.4
iPad [17]	83.3	69.4	**95.7**	89.7	84.8	94.6	91.5	87.8	94.8	77.9$\pm$22.4	73.5$\pm$22.1	86.4$\pm$19.5
FlowR2A (Ours)	**89.4**	**84.2**	94.7	**91.8**	**87.9**	**95.7**	**92.9**	**90.1**	**96.0**	**89.4**$\pm$7.4	**84.1**$\pm$8.5	**94.9**$\pm$6.1

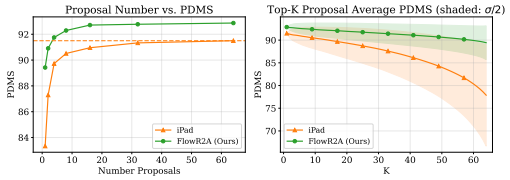


Figure 4: Comparing proposal quality with iPad [17]. (Left) PDMS under different proposal numbers. (Right) Average score of top-k proposals. Here σ indicates standard deviation.

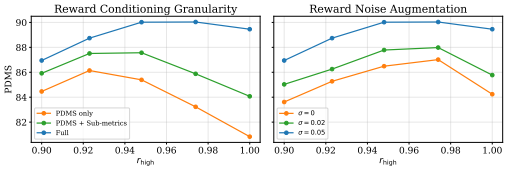


Figure 5: Ablation on reward conditioning. Evaluated on single proposal under different r_{high}. (Left) Reward condition granularity effect. (Right) Reward noise augmentation effect.

from history. NAVSIM is split into `navtrain`, containing 103k training frames, and `navtest`, containing 12k evaluation frames.

Metrics. NAVSIM [10, 3] computes closed-loop metrics for each planned trajectory by log-replay simulation. For NAVSIM-v1 [10], metrics include no at-fault collisions (NC), driving area compliance (DAC), time-to-collision (TTC), comfort, and ego progress (EP), aggregated into the Predictive Driver Model Score (PDMS). NAVSIM-v2 [3] further improves the evaluation using Extended PDMS (EPDMS), adding submetrics including driving direction compliance (DDC), traffic light compliance (TLC), lane keeping (LK), and extended comfort (EC).

Implementation Details. The perception backbone takes as input a front-view image stitched from the front, left, and right cameras together with a rasterized 2D BEV LiDAR feature map aggregating 4 recent frames for temporal context. We train end-to-end on `navtrain` for 100 epochs using AdamW (lr $= 3 \times 10^{-4}$, cosine annealing to 10^{-6}), with a total batch size of 64 across 4 NVIDIA H20 GPUs. The mode selector is trained for an additional two epochs in the second stage. At inference, we perform 20 denoising steps with CFG scale $w_g = 5$ and sample target score $r_{\text{high}} \in [0.9, 1.0]$, initial denoising time $t_{\text{init}} \in [0.5, 0.9]$, generating 60 proposals by default. See the appendix for full details.

4.1 Main Results

Results on NAVSIM-v1. Tab. 1 compares FlowR2A against existing methods on the NAVSIM-v1 [10] `navtest` split. FlowR2A achieves state-of-the-art 92.8 PDMS, outperforming all prior methods by $\geq$1.1 PDMS, with margins of $\geq$0.9 on TTC and $\geq$2.1 on EP. We highlight that our method leads in both safety (NC, TTC) and progress (EP) metrics, which are typically in tension. Even when sampling a single proposal, FlowR2A attains on par or better performance on safety metrics (NC, TTC) among multimodal methods, indicating consistently feasible proposals. We analyze this further in Sec. 4.2.

Results on NAVSIM-v2. On the NAVSIM-v2 [3] `navtest` split (Tab. 2), FlowR2A again leads with 88.9 EPDMS and the best scores on safety and progress submetrics (NC, TTC, EP). One submetric where FlowR2A underperforms is extended comfort (EC), which measures dynamic consistency across consecutive frames. Since our action decoder naturally produces multimodal proposals, enforcing such inter-frame consistency is the role of the mode selector, while our current selector scores each frame independently without modeling temporal correlation.

4.2 Proposal Quality

A core promise of learning $p(a|r)$ from dense supervision is that each sampled proposal should fall within the feasible action distribution, not just the one selected by the scorer. We test this claim against the strong anchor-based baselines, DiffusionDrive [35] and iPad [17].

Table 4: Diversity and worst-case proposal quality on `navtest`. Diversity follows the metric of DiffusionDriveV2 [52], computed over all proposals and over proposals with PDMS>0. Worst-case metrics are evaluated on the lowest-PDMS proposal of each scene. †Number taken from DiffusionDriveV2 [52].

	Diversity		Worst-Case Proposal					
Method	All ↑	PDMS>0 ↑	NC ↑	DAC ↑	TTC ↑	Comf. ↑	EP ↑	PDMS ↑
Diffusion Policy†	0.001	0.001	–	–	–	–	–	–
DiffusionDrive [35]	**0.419**	**0.328**	54.4	52.8	41.0	79.5	8.9	7.5
DiffusionDriveV2 [52]	0.385	0.279	52.9	39.3	43.9	76.0	0.6	0.4
iPad [17]	0.144	0.135	65.0	63.7	49.7	81.1	22.5	20.2
FlowR2A (Ours)	0.239	0.239	**93.6**	**89.6**	**79.8**	**99.5**	**63.9**	**69.3**

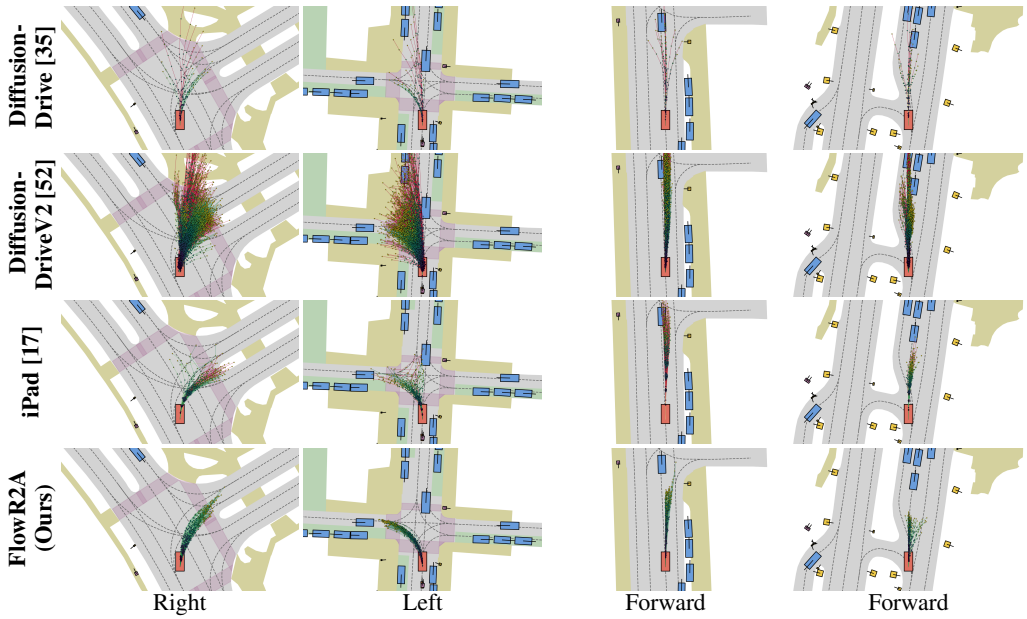


Figure 6: Qualitative comparison of proposal quality. Trajectories are colored by PDMS from 0 (red) to 1 (green). Driving command for each scene is labeled below. See App. D.4 for additional scenes.

Quantitative Comparison. Tab. 3 reports the performance across different proposal counts, and the mean±std over all proposals. We highlight two observations. First, FlowR2A reaches strong performance with very few proposals, surpassing iPad's full PDMS (64 proposals) within 4 proposals. Second, the gap is most pronounced when averaging performance over all proposals. FlowR2A obtains an average proposal PDMS exceeding iPad by +11.5 and DiffusionDrive by +29.1, with significantly lower standard deviation, indicating that our decoder produces consistent on-distribution proposals rather than scattered candidates. Fig. 4 visualizes this trend across K proposals. FlowR2A dominates iPad in PDMS at every generated proposal count (left) and in top-K average proposal score (right), with a markedly tighter spread. In addition, the 1-proposal column already matches the full system on safety, and the scorer contributes primarily to progress, showing that hard constraints are absorbed into $p(a|r)$ at sampling time, while selection only resolves soft trade-offs.

Diversity and Worst-Case Quality. Higher average quality could come from proposals collapsing around the GT. Tab. 4 rules this out. FlowR2A keeps reasonable diversity, far above a naive diffusion policy, and retains all of it after filtering out unsafe proposals (PDMS>0). In contrast, the diversity of DiffusionDrive and DiffusionDriveV2 drops substantially, since part of it comes from degenerate proposals. The worst proposal per scene shows the gap more clearly. Baselines degrade to near-zero worst-case PDMS, while FlowR2A raises it from 20.2 (iPad) to 69.3, with consistent gains on every submetric. Its remaining low-quality proposals are mostly low-progress rather than unsafe.

Qualitative Comparison. Fig. 6 compares BEV proposals on representative scenes. DiffusionDrive and DiffusionDriveV2 produce highly divergent proposals from fixed anchors, many of which receive negative scores. iPad noticeably tightens the proposal set, but still emits irregular low-quality trajectories on complex scenes. FlowR2A consistently concentrates proposals within a feasible action

Table 5: Ablation studies on reward conditioning. Evaluated on single proposal.

(a) Reward condition granularity

	Score-only	+Sub-rewards	+Per-timestep
EP ↑	83.3	**84.3**	84.0
TTC ↑	86.9	88.8	**94.9**
PDMS ↑	84.5	86.7	**89.4**

(b) Reward noise augmentation

noise scale σ	$\sigma=0$	$\sigma=0.02$	$\sigma=0.05$
EP ↑	79.1	82.8	**84.0**
TTC ↑	91.2	92.2	**94.9**
PDMS ↑	84.8	87.3	**89.4**

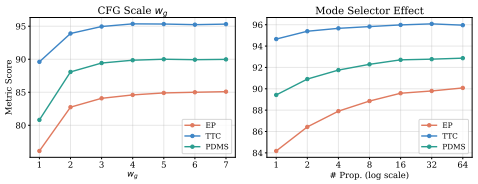


Figure 7: Ablation on CFG (left) and mode selector effect (right).

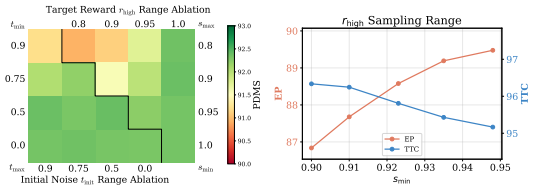


Figure 8: Ablation on sampling space (left) and inference-time objective balancing (right).

distribution across all scenes while preserving meaningful multimodality, demonstrating its ability to generate high-quality candidates at sampling time.

4.3 Ablation Studies

Ablation on Reward Condition Granularity. Tab. 5a ablates the reward condition granularity discussed in Sec. 3.1. Finer conditioning significantly improves proposal quality, most strikingly on TTC (88.8 to 94.9, +6.1). This confirms that exposing the model to where along the trajectory a constraint is violated is essential for the decoder to internalize hard safety constraints.

Ablation on Reward Noise Augmentation. Tab. 5b studies the effect of noise augmentation on continuous rewards. Without noise (σ=0), the decoder treats rewards as trajectory identifiers rather than quality indicators, causing PDMS to collapse when conditioned on high target scores at inference (Fig. 5 right). Larger noise scales force the model to map a band of high rewards to feasible high-quality actions, rather than relying on trajectory cues leaked by the precise reward value. Fig. 5 shows that the proposed per-timestep rewards and the noise augmentation together resolve the performance degradation with high reward conditioning.

Effect of Mode Selector. Fig. 7 right shows how the mode selector improves overall performance. As we increase the number of proposals it scores, the main gain is concentrated on progress (EP, 84 to 90), while safety metrics remain saturated. This confirms that the mode selector primarily refines soft objectives on top of an already-feasible proposal pool.

Effect of CFG. Fig. 7 left shows the effect of classifier-free guidance [19] (CFG) at inference. CFG is essential for high performance, consistent with the standard finding in conditional generative models. We use CFG scale 5, at which performance saturates.

Ablation on Sampling Strategy. Fig. 8 ablates the sampling strategies of Sec. 3.4. The left panel sweeps the sampling ranges of the target reward and initial noise. Given high rewards ($s_{\max} = 1.0$) with enough denoising steps ($t_{\min} \leq 0.5$), FlowR2A achieves consistently high performance. The right panel shows that adjusting the sampling range of r_{high} trades off between objectives at inference. With the same range length of 0.05, a lower $s_{\min}$ yields more conservative proposals, and vice versa.

5 Related Works

End-to-End Autonomous Driving. End-to-end autonomous driving maps raw sensor input directly to planning output through a single differentiable model. UniAD [20] pioneers this direction by integrating perception, prediction, and planning into one model. VAD [22] replaces dense BEV features with vectorized scene representations, and Transfuser [7] fuses multi-view images with LiDAR in the perception backbone. A line of follow-up works [6, 47, 30, 34, 46, 39, 51] continues to refine under the single-trajectory planning paradigm. Besides, some advanced methods [40, 27] adopt a vision-language-action model to obtain a promising trajectory.

Multimodal Driving Planning. Multimodal planners fall into two categories, scoring-based [5, 32, 33, 49] and anchor-based [35, 17, 24, 48]. Scoring-based methods select from a large fixed action vocabulary using a dedicated scorer. Starting from VADv2 [5], Hydra-MDP [32] and its extensions [28, 33] train learning-based scorers with fine-grained simulation-based supervision to enhance selection performance. DriveSuprim [49] refines scoring with a two-stage coarse-to-fine scheme.

Anchor-based methods decode proposals dynamically from a set of action anchors. DiffusionDrive [35] predicts proposals from fixed anchors via truncated diffusion. DiffusionDriveV2 [52] adds RL post-training and dense trajectory scoring. GoalFlow [48] pairs flow matching with goal-point anchors. iPad [17] uses a proposal-centric framework that iteratively refines proposals. Our method unifies the dense supervision of scoring-based methods with the generation ability of anchor-based methods by learning a reward-conditioned action distribution from dense trajectory-reward pairs.

Reward-Conditioned Policies and Offline RL. Our reward-to-action distribution learning is conceptually related to a line of offline RL methods that learn conditional policies from logged data, including reward-conditioned behavioral cloning [25, 45, 11], return-conditioned sequence modeling [4], goal-conditioned supervised learning [16], and diffusion-based planning [21]. These methods condition on a single scalar reward, return, or goal, whereas we condition on a per-timestep multi-signal reward vector covering safety, progress, comfort, and rule compliance, and apply this formulation to driving planning with rule-based simulation rewards. Recent works also explore multi-component reward parameterizations [42] and treat classifier-free guidance as a policy improvement operator [15].

6 Conclusion

In this paper, we propose a novel multimodal driving planning framework, FlowR2A, to learn the reward-conditioned action distribution. Our core contribution unifies dense reward supervision with generative action modeling through fine-grained reward signals. Experiments show that the resulting action decoder generates high-quality multimodal proposals consistently within the feasible action distribution. We hope this reward-conditioned generative modeling paradigm motivates future work that explores richer reward signals and extends to other policy learning scenarios beyond driving.

Acknowledgments and Disclosure of Funding

This work is supported by the Hong Kong Research Grant Council General Research Fund (No. 17213925), National Natural Science Foundation of China (No. 62422606), Shenzhen Loop Area Institute Fund (No. FPF10120250006), and HKU Shanghai Intelligent Computing Research Center.

References

[1] Michael Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *ICLR*, 2023.

[2] 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. In *CVPR workshop*, 2021.

[3] Wei Cao, Marcel Hallgarten, Tianyu Li, Daniel Dauner, Xunjiang Gu, Caojun Wang, Yakov Miron, Marco Aiello, Hongyang Li, Igor Gilitschenski, et al. Pseudo-simulation for autonomous driving. In *CoRL*, 2025.

[4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement learning via sequence modeling. *NeurIPS*, 2021.

[5] 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.

[6] Zhili Chen, Maosheng Ye, Shuangjie Xu, Tongyi Cao, and Qifeng Chen. PPAD: Iterative interactions of prediction and planning for end-to-end autonomous driving. In *ECCV*, 2024.

[7] 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.

[8] OpenScene Contributors. OpenScene: The largest up-to-date 3D occupancy prediction benchmark in autonomous driving. `https://github.com/OpenDriveLab/OpenScene`, 2023.

[9] Daniel Dauner, Marcel Hallgarten, Andreas Geiger, and Kashyap Chitta. Parting with misconceptions about learning-based vehicle motion planning. In *CoRL*, 2023.

[10] Daniel Dauner, Marcel Hallgarten, Tianyu Li, Xinshuo Weng, Zhiyu Huang, Zetong Yang, Hongyang Li, Igor Gilitschenski, Boris Ivanovic, Marco Pavone, et al. NAVSIM: Data-driven non-reactive autonomous vehicle simulation and benchmarking. *NeurIPS*, 2024.

[11] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. RvS: What is essential for offline RL via supervised learning? In *ICLR*, 2022.

[12] 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 *ICML*, 2024.

[13] Leonhard Euler. *Institutiones calculi integralis*. impensis Academiae imperialis scientiarum, 1792.

[14] Renju Feng, Ning Xi, Duanfeng Chu, Rukang Wang, Zejian Deng, Anzheng Wang, Liping Lu, Jinxiang Wang, and Yanjun Huang. ARTEMIS: Autoregressive end-to-end trajectory planning with mixture of experts for autonomous driving. *IEEE Robotics and Automation Letters*, 2025.

[15] Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Diffusion guidance is a controllable policy improvement operator. *arXiv preprint arXiv:2505.23458*, 2025.

[16] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. In *ICLR*, 2021.

[17] Ke Guo, Haochen Liu, Xiaojun Wu, Jia Pan, and Chen Lv. iPad: Iterative proposal-centric end-to-end autonomous driving. *arXiv preprint arXiv:2505.15111*, 2025.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[19] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[20] 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 *CVPR*, 2023.

[21] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *ICML*, 2022.

[22] 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 *ICCV*, 2023.

[23] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

[24] Ellington Kirby, Alexandre Boulch, Yihong Xu, Yuan Yin, Gilles Puy, Éloi Zablocki, Andrei Bursuc, Spyros Gidaris, Renaud Marlet, Florent Bartoccioni, et al. Driving on registers. In *CVPR*, 2026.

[25] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

[26] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. An energy and GPU-computation efficient backbone network for real-time object detection. In *CVPR workshop*, 2019.

[27] Jingyu Li, Junjie Wu, Dongnan Hu, Xiangkai Huang, Bin Sun, Zhihui Hao, Xianpeng Lang, Xiatian Zhu, and Li Zhang. Sgdrive: Scene-to-goal hierarchical world cognition for autonomous driving. *arXiv preprint arXiv:2601.05640*, 2026.

[28] Kailin Li, Zhenxin Li, Shiyi Lan, Yuan Xie, Zhizhong Zhang, Jiayi Liu, Zuxuan Wu, Zhiding Yu, and Jose M Alvarez. Hydra-MDP++: Advancing end-to-end driving via expert-guided hydra-distillation. *arXiv preprint arXiv:2503.12820*, 2025.

[29] Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. *arXiv preprint arXiv:2511.13720*, 2025.

[30] Yingyan Li, Lue Fan, Jiawei He, Yuqi Wang, Yuntao Chen, Zhaoxiang Zhang, and Tieniu Tan. Enhancing end-to-end autonomous driving with latent world model. In *ICLR*, 2025.

[31] Yingyan Li, Yuqi Wang, Yang Liu, Jiawei He, Lue Fan, and Zhaoxiang Zhang. End-to-end driving with online trajectory evaluation via BEV world model. In *ICCV*, 2025.

[32] Zhenxin Li, Kailin Li, Shihao Wang, Shiyi Lan, Zhiding Yu, Yishen Ji, Zhiqi Li, Ziyue Zhu, Jan Kautz, Zuxuan Wu, et al. Hydra-MDP: End-to-end multimodal planning with multi-target hydra-distillation. *arXiv preprint arXiv:2406.06978*, 2024.

[33] Zhenxin Li, Wenhao Yao, Zi Wang, Xinglong Sun, Joshua Chen, Nadine Chang, Maying Shen, Zuxuan Wu, Shiyi Lan, and Jose M Alvarez. Generalized trajectory scoring for end-to-end multimodal planning. *arXiv preprint arXiv:2506.06664*, 2025.

[34] Zhiqi Li, Zhiding Yu, Shiyi Lan, Jiahan Li, Jan Kautz, Tong Lu, and Jose M Alvarez. Is ego status all you need for open-loop end-to-end autonomous driving? In *CVPR*, 2024.

[35] 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 *CVPR*, 2025.

[36] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *ICLR*, 2023.

[37] Lin Liu, Guanyi Yu, Ziying Song, Junqiao Li, Caiyan Jia, Feiyang Jia, Peiliang Wu, and Yandan Luo. Beyond imitation: Constraint-aware trajectory generation with flow matching for end-to-end autonomous driving. *arXiv preprint arXiv:2510.26292*, 2025.

[38] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.

[39] Zhe Liu, Jinghua Hou, Xiaoqing Ye, Jingdong Wang, Hengshuang Zhao, and Xiang Bai. Unilion: Towards unified autonomous driving model with linear group rnns. *arXiv preprint arXiv:2511.01768*, 2025.

[40] Zhe Liu, Runhui Huang, Rui Yang, Siming Yan, Zining Wang, Lu Hou, Di Lin, Xiang Bai, and Hengshuang Zhao. Drivepi: Spatial-aware 4d mllm for unified autonomous driving understanding, perception, prediction and planning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3688–3698, 2026.

[41] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *ICLR*, 2022.

[42] Michal Nauman, Marek Cygan, and Pieter Abbeel. Reward-conditioned reinforcement learning. *arXiv preprint arXiv:2603.05066*, 2026.

[43] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

[44] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

[45] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[46] Wenchao Sun, Xuewu Lin, Yining Shi, Chuang Zhang, Haoran Wu, and Sifa Zheng. SparseDrive: End-to-end autonomous driving via sparse scene representation. In *ICRA*, 2025.

[47] Xinshuo Weng, Boris Ivanovic, Yan Wang, Yue Wang, and Marco Pavone. PARA-Drive: Parallelized architecture for real-time autonomous driving. In *CVPR*, 2024.

[48] 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 *CVPR*, 2025.

[49] Wenhao Yao, Zhenxin Li, Shiyi Lan, Zi Wang, Xinglong Sun, Jose M Alvarez, and Zuxuan Wu. DriveSuprim: Towards precise trajectory selection for end-to-end planning. In *AAAI*, 2026.

[50] Chengran Yuan, Zhanqi Zhang, Jiawei Sun, Shuo Sun, Zefan Huang, Christina Dao Wen Lee, Dongen Li, Yuhang Han, Anthony Wong, Keng Peng Tee, et al. DRAMA: An efficient end-to-end motion planner for autonomous driving with Mamba. *arXiv preprint arXiv:2408.03601*, 2024.

[51] Wenzhao Zheng, Ruiqi Song, Xianda Guo, Chenming Zhang, and Long Chen. GenAD: Generative end-to-end autonomous driving. In *ECCV*, 2024.

[52] Jialv Zou, Shaoyu Chen, Bencheng Liao, Zhiyu Zheng, Yuehao Song, Lefei Zhang, Qian Zhang, Wenyu Liu, and Xinggang Wang. DiffusionDriveV2: Reinforcement learning-constrained truncated diffusion modeling in end-to-end autonomous driving. *arXiv preprint arXiv:2512.07745*, 2025.

A Limitations and Future Directions

Limitations. The quality of the reward-conditioned action distribution $p(a|r)$ is bounded by the fidelity of the reward signals used as the condition, and in this work all signals come from the NAVSIM rule-based simulator. Inaccuracies in any subscore propagate into supervision. For instance, the NAVSIM ego-area check sometimes returns false detections due to gaps between adjacent area polygons, which enters training as label noise on the per-timestep ego-area array. A separate limitation lies in the mode selector, which is not a focus of this work. Our lightweight two-layer transformer occasionally promotes an unsafe proposal even when the action decoder produces a feasible majority (Fig. 11), and because it scores each frame independently it gives no explicit pressure for inter-frame consistency, which is reflected in the lower Two-Frame Extended Comfort score. Finally, evaluation is restricted to NAVSIM, since other planning datasets lack a comparable reward labeling pipeline.

Future Directions. A central question is how FlowR2A extends to settings where a NAVSIM-style rule-based simulator is unavailable. The generative formulation only requires a function from candidate action to reward vector, so the simulator can be replaced by a learning-based reward model, by hand-crafted proxy metrics, or by a pretrained trajectory scorer such as GTRS [33]. All of these alternatives are substantially faster than rule-based simulation and can support online labeling with random or model-proposed trajectories during training, rather than the one-time offline pass we use here. Beyond reward sourcing, designing more fine-grained rewards that capture aspects beyond the NAVSIM rule set, building a stronger mode selector with cross-frame context, and applying FlowR2A to broader planning datasets are natural next steps.

Broader Impacts. FlowR2A targets autonomous driving planning, where deployment carries direct safety implications. Bias or blind spots in the reward signals can propagate into the learned action distribution, so practical use would require independent runtime monitoring and human oversight. Our evaluation is restricted to closed-loop simulation on NAVSIM and we make no claims about real-world deployment.

B Comparison with Prior Generative Planners

Several recent end-to-end planners adopt a flow-matching or diffusion-based trajectory decoder [35, 48, 37]. FlowR2A shares this generative formulation but differs in what the decoder is trained to model. Prior methods supervise the decoder with a single ground-truth trajectory per scene and inject multimodality through external mechanisms such as fixed anchors, goal-point selection, or scorer outputs. FlowR2A instead supervises the decoder with dense action-reward pairs that span the action space, so the learned distribution $p(a|r)$ is multimodal by construction.

Naive Diffusion or Flow Policies. A baseline approach trains the generative decoder on the GT trajectory under a diffusion or flow-matching loss. Since the supervision target is deterministic given the scene, the decoder has no incentive to use the noise input as an information channel and learns to predict the GT directly from the scene features. The generative formulation collapses into single-trajectory regression, and proposals from different noise samples converge to one mode at convergence. DiffusionDrive [35] reports this collapse explicitly when applying a vanilla diffusion policy on top of Transfuser [7].

DiffusionDrive. DiffusionDrive addresses the collapse with anchored truncated diffusion. It clusters training trajectories into 20 anchors and uses each anchor as the start of a truncated diffusion schedule. Per-anchor reconstruction is supervised by a winner-takes-all loss in which only the anchor closest to the GT receives the reconstruction signal, while a separate classification head ranks anchors. This forces proposals to span a multimodal anchor space, which is the source of the reported diversity. The supervision per winning anchor, however, remains a single-GT objective. Each anchor selected as the winner is regressed to that scene's GT, so the per-anchor distribution can still collapse to one trajectory. The procedure spans modes across a fixed anchor set rather than within a scene, and the anchor set is shared across all scenes.

GoalFlow. GoalFlow [48] decouples multimodality from the flow decoder by introducing a separate goal-point construction module that selects a single high-scoring endpoint and conditions the flow on this goal. The flow decoder is then trained with the standard rectified-flow velocity-matching loss against the single GT. The flow itself is therefore subject to the same degeneration as the naive policy, and the diversity comes from varying the goal-point condition rather than from sampling the flow.

Table 6: Controllability of the reward condition on `navtest`. Evaluated on single proposal.

(a) Target score

Target score	NC ↑	TTC ↑	EP ↑	PDMS ↑
0.90	**98.9**	**96.4**	75.9	87.1
0.95	98.6	95.6	85.1	**90.3**
1.00	98.0	92.6	**88.4**	89.6

(b) Comfort condition

Comfort target	NC ↑	DAC ↑	HC ↑
1 (comfortable)	**98.1**	**96.6**	**98.3**
unset	98.0	**96.6**	96.2
0 (uncomfortable)	92.3	90.5	18.0

Table 7: Inference latency breakdown on NAVSIM `navtest`. Per-sample latency in milliseconds, measured on a single NVIDIA H20 GPU with batch size 1 over 1000 samples. N is the number of proposals, K is the total Euler solver steps over $[0, 1]$, and $t_{\min}$ is the lower bound of the initial noise sampling range. The first row is the default setting reported in the main paper.

N	K	$t_{\min}$	Component Latency (ms)				Total (ms)	PDMS ↑
			Perception	Reward Enc.	Denoising	Selector		
60	20	0.50	13.8	2.6	70.2	4.3	91.3	**92.8**
4	20	0.50	14.2	2.7	60.6	4.2	82.0	91.8
60	20	0.75	14.1	2.7	36.9	4.5	58.7	92.3
60	10	0.50	12.8	2.5	33.5	4.0	53.2	92.2
60	10	0.70	14.4	2.7	22.2	4.4	44.1	92.0

This view is consistent with the reported observation that even a single denoising step is sufficient at inference, indicating that the noise input carries little information.

CATG. CATG [37] adds richer conditioning, including a trajectory anchor and a target endpoint selected from the top-100 candidates of a pretrained GTRS [33] scorer, together with a per-trajectory ego-progress label. The flow decoder is again trained with a flow-matching loss against the single GT. The richer conditioning sharpens the conditional distribution, which makes per-condition mode collapse more likely rather than less. CATG mitigates the resulting drivable-area-compliance issues with three inference-time constraints over the velocity field and the flow start. The method effectively functions as a flow-matching post-processor over GTRS proposals, at the cost of 100 candidates and 100 sampling steps per scene.

Difference of FlowR2A. The common pattern across these methods is to train a generative trajectory decoder under single-GT supervision and to inject multimodality externally through anchors, goal points, or scorer outputs. Adding scene-conditional information does not resolve the underlying tension, since the supervision remains deterministic given the chosen condition. FlowR2A addresses this at the supervision level. The reward-conditioned distribution $p(a|r)$ is trained on dense action-reward pairs that span the action vocabulary in each scene. So the same condition maps to multiple possible actions instead of a single one, necessitating the generative formulation and forcing the decoder to rely on its noisy sample input. Multimodality is a property of the learned distribution rather than explicit anchors, and the trained decoder produces high-quality proposals through reward conditioning alone.

C Additional Experiments

C.1 Controllability

The reward condition directly steers proposal generation. Tab. 6a varies the target score: a lower target yields more conservative proposals with higher safety, and a higher target yields more aggressive proposals with higher progress (see also Fig. 9). Tab. 6b toggles the comfort condition, and the generated proposals follow it accordingly. We report the NAVSIM v2 history comfort (HC) metric here, since the v1 comfort metric labels almost all trajectories as comfortable.

C.2 Latency Analysis

Tab. 7 reports the per-component latency of FlowR2A. The denoising loop dominates total latency, exceeding 75% of the per-frame cost in the default 20-step setting, while perception, reward encoding, and the mode selector together stay below 22 ms across all configurations. The proposal count has

Table 8: PDMS vs. number of denoising steps on NAVSIM v1 `navtest`. Number of proposals N=60. Per-sample latency measured on a single NVIDIA H20 GPU with batch size 1 over 1000 samples.

K	Denoising (ms)	Total (ms)	PDMS ↑
10	33.5	53.2	92.2
20	70.2	91.3	**92.8**
30	104.3	125.1	92.7
40	140.0	161.4	92.7

Table 9: Effect of mode selector aggregation weights on NAVSIM v1 `navtest`. The weights $(w_{\mathrm{EP}}, w_{\mathrm{TTC}}, w_{\mathrm{HC}})$ are applied to the weighted subscores in Eq. 12. The first row is our default; the second row uses the official NAVSIM weights.

$(w_{\mathrm{EP}}, w_{\mathrm{TTC}}, w_{\mathrm{HC}})$	NC ↑	DAC ↑	EP ↑	TTC ↑	Comf. ↑	PDMS ↑
$(1, 1, 1)$ (default)	98.8	98.0	**90.1**	96.0	100	92.8
$(5, 5, 2)$	98.8	97.9	90.0	95.9	100	92.8
$(1, 2, 1)$	99.0	98.0	89.0	96.8	100	92.8
$(1, 3, 1)$	**99.1**	**98.1**	88.0	**97.1**	100	92.6

little effect on latency, with only a 9 ms reduction when going from 60 to 4 proposals, leaving the sequential denoising steps as the dominant factor. The generative formulation allows us to customize the denoising process for efficiency trade-off by changing the total Euler solver steps K and the minimum initial denoising time $t_{\min}$. K defines the discretization of the full $[0, 1]$ time interval, and only the steps from t_{init} onward are executed, so the realized number of forward passes per proposal is $\lceil K\,(1 - t_{\mathrm{init}})\rceil$. Raising $t_{\min}$ from 0.50 to 0.75 (rows 1, 3) drops latency from 91 to 59 ms at a 0.5 PDMS regression, halving K from 20 to 10 (rows 1, 4) gives a similar speed-up at 0.6 PDMS regression, and combining $t_{\min}$=0.70 with K=10 (row 5) reaches 44 ms per frame, more than $2\times$ faster than the default while staying within 0.8 PDMS. Both controls are chosen at inference time, so the same trained model can run at any of these speed-quality settings.

C.3 Number of Denoising Steps

Tab. 8 sweeps the number of denoising steps K under default sampling setting. PDMS saturates at K=20 and does not improve when K is raised further, while latency grows roughly linearly with K. Halving K to 10 gives a $1.7\times$ speed-up at a 0.6 PDMS regression. We therefore use K=20 as the default.

C.4 Mode Selector Aggregation Weights

Tab. 9 reports performance on `navtest` under different aggregation weights w_k in Eq. 12. The final PDMS is stable across settings, including the official NAVSIM weights $(w_{\mathrm{EP}}, w_{\mathrm{TTC}}, w_{\mathrm{HC}})$=$(5, 5, 2)$. Beyond robustness, the weights also serve as a free inference-time knob to balance EP and TTC. Increasing w_{TTC} relative to w_{EP} smoothly trades progress for safety without retraining, raising TTC from 96.0 to 97.1 at the cost of EP dropping from 90.1 to 88.0, while PDMS stays within 0.3.

D Qualitative Results

D.1 Sampling Space Visualization

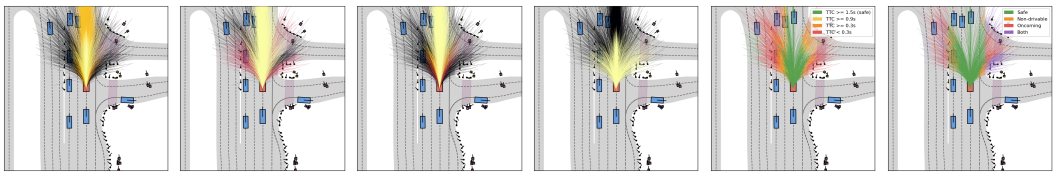
Fig. 9 sweeps the two sampling controls on a single scene. The column sweep varies r_{high} within the high-reward region, from 0.80 to 1.00. Within this range the proposals are already high-quality, so the visible effect is a shift toward higher ego progress, with trajectories reaching further along the route as r_{high} grows. Moving down the rows (t_{init} from 0.75 to 0.95) anchors the denoising trajectory closer to the IL head output, which reduces the spatial spread of proposals and converges the set toward a single mode.

D.2 Reward Score Distribution

Fig. 10 visualizes the reward labels assigned to the full 8192-trajectory vocabulary for one scene. The distributions are highly non-uniform: most candidate trajectories receive low aggregate PDMS,



Figure 9: Sampling-space visualization of FlowR2A on a single `navtest` scene. The grid sweeps the high-reward target r_{high} across columns and the initial denoising time t_{init} across rows, with all other inference settings fixed. Each cell shows 60 proposals from a single sampling configuration.



(a) PDMS (b) NC (c) EP (d) HC (e) TTC-time (f) Ego-area

Figure 10: Reward score distribution over the action vocabulary on a single `navtest` scene. Each panel colors the 8192 vocabulary trajectories by one reward signal. PDMS, NC, EP, and HC use a brighter color to denote higher score; TTC-time and ego-area use a categorical palette and selectively show trajectories with violations.

while high-quality actions occupy a sparse subset of the vocabulary. Different reward signals also carve the action space in complementary ways. Scalar metrics such as EP and HC describe soft preferences over progress and smoothness, whereas the per-timestep TTC-time and ego-area labels localize hard safety and drivable-area violations along the trajectory. This heterogeneity motivates both our inverse-density sampling strategy during training and the fine-grained reward conditioning used by the decoder.

D.3 Failure Cases

Fig. 11 shows four representative failures of FlowR2A on `navtest`. From left to right, the cases are a collision with a vehicle that becomes visible only mid-trajectory and is missed by the perception

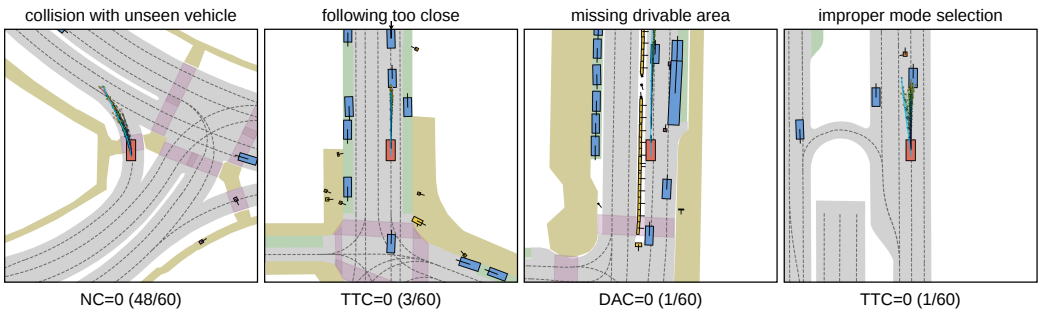


Figure 11: Failure cases of FlowR2A on `navtest`. For each scene, the top label names the failure mode and the bottom label reports the failed metric together with the count of failing proposals out of the 60 sampled proposals. Selected proposal is colored in blue.

encoder, following the lead vehicle too closely so fails TTC, drifting near the drivable area edge so fails DAC, and a mode-selection error that ranks an unsafe proposal above feasible alternatives.

These cases suggest that the current performance of FlowR2A is primarily bounded by perception coverage and mode selector quality. Of the two, the selector is the more actionable bottleneck. In scenes where the action decoder already produces a feasible majority, the selector still occasionally promotes one of the few failing proposals, leaving headroom that a stronger ranking model could recover. Improving the selector is a promising direction for future work.

D.4 Extended Comparisons

Fig. 12, Fig. 13 and Fig. 14 extend the qualitative comparison in the main paper to additional `navtest` scenes.

E NAVSIM Benchmark and PDM Score

We give a self-contained description of the NAVSIM [10, 3] benchmark and the closed-loop PDM score used for evaluation and reward construction in the main paper. Sec. E.1 describes the dataset and the simulation pipeline. Sec. E.2 lists every subscore. Sec. E.3 gives the aggregation formulas for PDMS (v1) and EPDMS (v2). Sec. E.4 summarizes the differences between the two versions.

E.1 Dataset and Simulation Pipeline

Dataset. NAVSIM is built on top of OpenScene [8], a compact redistribution of nuPlan [2]. It curates real-world non-trivial driving scenes where the future plan cannot be directly extrapolated from history, and is split into `navtrain` (103k frames) and `navtest` (12k frames). Each frame provides multi-view camera images, LiDAR point clouds, ego status, driving command, surrounding agent tracks, map elements, and a 4-second future ground-truth trajectory.

Simulation. A planned trajectory is evaluated by closed-loop log-replay simulation. The 4-second plan, given as eight ego-frame waypoints at 0.5 s intervals, is converted into a global interpolated trajectory and fed to an LQR tracker that drives a kinematic bicycle model at 10 Hz. The output is a sequence of 41 states at 0.1 s intervals. All subscores are computed on this simulated state sequence rather than on the raw planned waypoints.

PDM Reference Trajectory. The official scorer also evaluates a rule-based reference trajectory produced by the PDM-Closed planner [9]. The reference is used only to normalize ego progress (Sec. E.2).

E.2 Subscores

NAVSIM subscores split into multiplicative metrics that gate the final score on hard violations, and weighted metrics that contribute additively to a quality term.

No At-Fault Collisions (NC, multiplicative). For each timestep, the scorer checks whether the ego polygon intersects any tracked object. A collision is at-fault if it is a front collision, a stopped-object collision, or a lateral collision while the ego sits in multiple lanes or non-drivable area. NC is set to 0 on an at-fault collision with another road user, 0.5 on collisions with static objects, and is 1 otherwise. The minimum value across all timesteps is taken.

Drivable Area Compliance (DAC, multiplicative). DAC is 0 if any corner of the ego bounding box lies outside all drivable polygons at any timestep, otherwise 1.

Driving Direction Compliance (DDC). Multiplicative in v2, disabled in v1. It accumulates oncoming-direction motion per timestep, with the ego displacement set to 0 when the ego center is on a route lane or the ego is in an intersection. The maximum oncoming progress within a sliding window is then thresholded into a score of $\{0, 0.5, 1\}$.

Traffic Light Compliance (TLC, v2 only, multiplicative). TLC is 0 if the ego polygon intersects any red-light token at any timestep, otherwise 1.

Ego Progress (EP, weight 5). The raw progress is the centerline-projected displacement along the trajectory, clipped to be non-negative. It is normalized by a per-scene constant equal to the maximum raw progress across the agent and the PDM reference trajectory after multiplying by the multiplicative score, so that a trajectory that fails a multiplicative gate does not raise the normalization constant. EP defaults to 1 when the reference progress is too little to be informative.

Time-to-Collision (TTC, weight 5). At each timestep, the ego is forward-projected at constant velocity and heading over a 1.0 s window, and the resulting expanded polygon is checked against the same at-fault rules as NC. TTC is 0 if any projected polygon collides at fault, otherwise 1. A stopped ego and previously collided objects are skipped.

Comfort (C, v1 only, weight 2). Six dynamic quantities (longitudinal acceleration, lateral acceleration, total jerk, longitudinal jerk, yaw rate, yaw acceleration) are computed on the simulated states with a Savitzky-Golay filter and thresholded against fixed bounds. C is 1 only if all six quantities stay within bounds at every timestep.

History Comfort (HC, v2 only, weight 2). HC replaces v1's comfort. The simulated states are prepended with the past human ego states before applying the same six-quantity comfort check. HC penalizes discontinuities at the boundary between history and the planned trajectory, where a velocity or acceleration mismatch produces a jerk spike under the Savitzky-Golay filter.

Two-Frame Extended Comfort (EC, v2 only, weight 2). EC is computed across two consecutive scene frames. The overlapping portion of their simulated state sequences is used to compute root-mean-square differences in acceleration magnitude, jerk magnitude, yaw rate, and yaw acceleration. EC is 1 if all four RMS differences stay below their thresholds, otherwise 0. EC is undefined for the first frame in a scene, in which case the EC term is dropped from the weighted average.

Lane Keeping (LK, v2 only, weight 2). The lateral distance from the ego center to the lane centerline is computed at each timestep with intersections excluded. LK is 0 if the deviation exceeds a small threshold for a sustained duration, otherwise 1.

Human Penalty Filter (v2 only). A post-processing step that re-runs the human reference trajectory through the same scoring pipeline and overrides the agent's score to 1 on any subscore where the human also fails. This forgives failures that are unavoidable given the scene.

E.3 Aggregation

NAVSIM v1. The PDM score combines the multiplicative metrics NC and DAC with the weighted metrics EP, TTC, and C,

$$\mathrm{PDMS} = \mathrm{NC} \cdot \mathrm{DAC} \cdot \frac{5\,\mathrm{EP} + 5\,\mathrm{TTC} + 2\,\mathrm{C}}{12}. \tag{9}$$

DDC is technically present in v1 but enters with weight 0, so it does not contribute.

NAVSIM v2. The Extended PDM score promotes DDC to a multiplicative gate, adds TLC as a second new gate, replaces C with HC, and introduces LK and EC as additional weighted terms,

$$\mathrm{EPDMS} = \mathrm{NC} \cdot \mathrm{DAC} \cdot \mathrm{DDC} \cdot \mathrm{TLC} \cdot \frac{5\,\mathrm{EP} + 5\,\mathrm{TTC} + 2\,\mathrm{LK} + 2\,\mathrm{HC} + 2\,\mathrm{EC}}{16}. \tag{10}$$

When EC is undefined for a frame (no previous adjacent scene), the EC term is dropped from the numerator and the denominator becomes 14.

E.4 Differences Between v1 and v2

The two versions share most subscore definitions but differ in five places.

- DDC is promoted from a weight-0 weighted term in v1 to a multiplicative gate in v2, and additionally excludes intersections from oncoming progress accumulation.
- TLC is added in v2 as a second new multiplicative gate.
- Comfort (C) in v1 is replaced by History Comfort (HC) in v2, which prepends 1.5 s of past human ego states before the comfort check, and Two-Frame Extended Comfort (EC) is added as a separate weighted term that measures dynamic consistency across consecutive scene frames.
- Lane Keeping (LK) is added in v2 as a weighted term.
- Human penalty filter is added in v2.

EP and TTC keep the same form in both versions up to minor implementation differences in the normalization and the iteration bound.

F Implementation Details

This section provides the full implementation details of FlowR2A, organized in the same way as the method section. Sec. F.1: reward construction, Sec. F.2: model architecture, Sec. F.3: two-stage training recipe, Sec. F.4: inference configuration.

F.1 Reward Construction

Dense Action-Reward Pairs. We follow scoring-based methods [5, 32, 33] and use a dense action vocabulary $\mathcal{V}_a$ of 8192 four-second trajectories obtained by clustering 700K trajectories from nuPlan [2], each containing 8 ego-frame waypoints at 0.5 s spacing. For every `navtrain` scene, we simulate all vocabulary trajectories through the NAVSIM simulator and record the resulting reward labels, producing dense action-reward pairs (a, r) that span the action space.

Scalar Rewards. The scalar rewards in $\mathcal{R}$ are the NAVSIM v2 subscores NC, DDC, TLC, EP, LK, HC and the overall PDM score (Sec. E.2). Two-Frame Extended Comfort is excluded as it is not defined within a single scene. EP is replaced by a safety-oriented variant. We first gate it by a safety mask that is 1 when both NC$= 1$ and TTC$= 1$ and 0 otherwise, then divide by the maximum masked EP across $\mathcal{V}_a$ in the scene.

Per-Timestep Array Rewards. We replace the binary DAC and TTC subscores with per-timestep arrays that preserve the hard constraint at higher temporal resolution. The arrays are computed with the adapted NAVSIM v1 PDM scorer.

- **TTC-time array.** At each timestep, the ego is forward-projected at constant velocity over a 1.5 s detection horizon and we record the minimum projected collision time, set to the horizon when no collision is detected. The array is sampled at 40 timesteps, yielding a 40-dim array per trajectory.
- **Ego-area array.** At each timestep we record two binary flags, on-road (the ego polygon lies inside the drivable area) and on-route (the ego center is on the route lane consistent with the driving direction), sampled at 8 timesteps to yield an $(8, 2)$ array per trajectory.

Continuous-Reward Noise Augmentation. During training, we corrupt EP and the PDM score by adding Gaussian noise from $\mathcal{N}(0, \sigma^2)$ with $\sigma = 0.05$, acting as label smoothing on these dense scalars. No noise is added at inference.

Simulation Cost. Reward construction is a one-time offline cost. Following the GTRS [33] protocol, scoring $\mathcal{V}_a$ on `navtrain` for NAVSIM v2 scalar rewards takes around one day on 32 machines. We compute the per-timestep rewards on 4 nodes of 8 machines, taking approximately 8 hours. Simulation runs on CPU only and does not require GPU resources.

F.2 Model Architecture

F.2.1 Perception Encoder

We adopt the Transfuser [7] backbone without modification. The backbone outputs a set of BEV feature tokens, which we concatenate with an encoded status feature that embeds the ego status and driving command. Positional encodings are added to the concatenated tokens to form the context tokens. A set of agent tokens is then queried from the context tokens through cross-attention. Together, the context tokens and agent tokens form the scene features $\boldsymbol{s}$ used by the downstream action decoder.

Two auxiliary losses follow the Transfuser recipe. A BEV semantic segmentation loss supervises the backbone BEV features, and an agent prediction loss is applied to the queried agent tokens. We keep the same imitation-learning (IL) head of Transfuser, an MLP supervised with the L1 loss against the GT trajectory, that produces the optional inference-time anchor (Sec. 3.4). These three losses sum into $\mathcal{L}_{\text{perc}}$.

F.2.2 Reward Encoder

The reward encoder maps each reward in $\mathcal{R}$ into a 256-dim embedding using a per-reward embedder selected by reward type. Discrete rewards use learnable embedding tables, scalar continuous rewards use sinusoidal positional encoding, and the per-timestep array rewards (TTC-time, ego-area) use an MLP. Each reward also has a learned null token $\boldsymbol{n}_k$ that replaces its embedding when the reward is dropped. The resulting embeddings are concatenated and passed through an MLP to produce the 256-dim condition embedding $\boldsymbol{r}_c$.

Reward Dropout Policy. We use a three-mode dropout schedule during training. With probability 0.5 we keep all rewards, with probability 0.1 we drop all rewards together to form the empty condition $r_\emptyset$ used at inference, and with probability 0.4 we drop each reward independently with per-reward probability 0.5. The all-drop event ensures that $r_\emptyset$ is observed often enough for the model to learn the unconditional distribution required by classifier-free guidance, while the independent dropout exposes the model to a wide range of partial conditions for arbitrary subset conditioning at inference.

F.2.3 Flow-based Action Decoder

Trajectory Representation. The decoder operates on 4-second trajectories represented as 8 waypoints (x, y, θ) where θ is the ego heading. The (x, y) is normalized to $[-2, 2]$, and the heading is decoupled into $(\sin\theta, \cos\theta)$. So the shape of a trajectory is 8×4.

Architecture. We stack four transformer blocks with hidden dim 256, FFN dim 1024, and 8 attention heads. The input trajectory is encoded by a sinusoidal positional embedding followed by an MLP projected to hidden dim, and a learned positional embedding is added along the waypoint dimension. Each block applies, in order, self-attention, cross-attention to the context tokens, cross-attention to the agent tokens, and an FFN, with residual connections around each module. AdaLN modulation is applied before every attention and FFN module, and dropout is added around the agent cross-attention to reduce overfitting. After the final block, a linear head projects each waypoint feature back to 4 to produce the predicted clean trajectory.

Conditioning. The time t is embedded by sinusoidal positional encoding followed by a 2-layer MLP. The AdaLN modulation input is the concatenation of the time embedding and the reward embedding $\boldsymbol{r}_c$, projected per block into the scale and shift parameters.

F.2.4 Mode Selector

Architecture. The mode selector is a lightweight two-layer transformer that scores trajectory proposals from the action decoder. It first encodes the input trajectory with sinusoidal positional embeddings, then applies the same transformer block as the action decoder with three modifications. We drop the trajectory self-attention module, replace AdaLN with ordinary layer normalization, and add a grid-sample-based cross-attention module that attends to the BEV features. The output feature is mapped to a set of subscore predictions through shallow MLP heads.

Prediction Heads. We attach one prediction head per NAVSIM subscore, covering NC, DAC, TTC, EP, and HC. Two auxiliary heads predict the per-timestep TTC-time and ego-area arrays used in the reward condition (Sec. 3.1). The auxiliary heads increase the resolution of the selector's awareness

of safety and rule compliance. The auxiliary heads are enabled only in stage 2 (App. F.3.4); stage 1 trains the five subscore heads only (App. F.3.3). Our selector score follows the NAVSIM PDMS aggregation formulation rather than replicating it exactly. In particular, we predict v2 HC instead of v1 Comf., as the v1 Comf. computation has a known issue that labels almost all candidate trajectories as 1 and provides little ranking signal, whereas v2 HC corrects this by prepending past ego states before the comfort check.

Continuous TTC Label. The official TTC subscore in NAVSIM is binary and gives no gradient near the decision boundary. We replace it with a continuous label $\bar{s}_{\mathrm{TTC}} = \min(ttc_{\min}, t_{\mathrm{bound}})/t_{\mathrm{bound}}$, where $ttc_{\min}$ is the minimum projected collision time across the trajectory horizon and t_{bound} is the maximum detection bound. The label takes value 0 at an actual collision and 1 when no collision is detected within t_{bound} at any timestep. We set $t_{\mathrm{bound}} = 2$ seconds in practice.

Training Loss. The five subscore heads (NC, DAC, TTC, EP, HC) and the per-timestep ego-area auxiliary head are supervised with binary cross-entropy (BCE) loss. The per-timestep TTC-time auxiliary head is supervised with MSE since its target is not bounded in $[0, 1]$. The total selector loss is the sum of all head losses,

$$\mathcal{L}_{\mathrm{sel}} = \sum_{k} \mathcal{L}_k, \tag{11}$$

where $\mathcal{L}_k$ is the BCE or MSE term for head k.

Final Score Aggregation. Following the official NAVSIM aggregation, the final ranking score is

$$s_{\mathrm{final}} = \Big(\prod_{k \in \mathcal{M}} \hat{s}_k \Big) \cdot \Big(\sum_{k \in \mathcal{W}} w_k \hat{s}_k \Big), \tag{12}$$

where $\mathcal{M} = \{NC, DAC\}$ are the multiplicative metrics and $\mathcal{W} = \{EP, TTC, HC\}$ are the weighted metrics. We set all weights w_k to 1 to equally treat all weighted scores. The final performance of FlowR2A is stable under different weight choices (App. C.4).

F.3 Training Details

F.3.1 Input Construction

The image input is a 1024×256 stitched view from the front, front-left, and front-right cameras. The LiDAR input aggregates four frames at 0.5 Hz, covering a 2-second history from -1.5 s to the current frame. History points are transformed into the current ego frame and rasterized into 2D BEV histograms based on local point statistics. We use both above-ground and below-ground points, producing an 8-channel BEV grid as the LiDAR input.

F.3.2 Trajectory Sampling

The dense action vocabulary contains many trajectories that fail safety or compliance checks in any given scene, so the empirical PDM-score distribution over all (a, r) pairs is heavily skewed toward zero. Uniform sampling from this set under-represents the rare high-score samples that the decoder needs to model the high-quality region of $p(a|r)$. To rebalance, we sample (a, r) pairs with weight inversely proportional to the score density,

$$w(s) \propto \frac{1}{p(s)^{\alpha}}, \tag{13}$$

where $p(s)$ is the empirical density of the PDM score s, estimated per scene with a 1D Gaussian KDE over the vocabulary scores, and $\alpha \in [0, 1]$ controls the strength of the rebalancing. A small constant is added to $p(s)$ for numerical stability before inverting. At $\alpha = 0$, w reduces to a constant and sampling is uniform across all trajectories. At $\alpha = 1$, the score density is fully canceled and sampling is uniform across score bins. We use $\alpha = 0.6$ in practice, which lifts the rare high-score samples while still retaining enough common low-score actions to model the full distribution.

At each training step, we separately sample 20 trajectories for the action decoder and the mode selector using this weighting.

F.3.3 Stage 1: End-to-End Training

We train all components jointly on `navtrain` for 100 epochs with AdamW (default β, weight decay 10^{-4}). The learning rate follows a cosine schedule from 3×10^{-4} to 10^{-6} with a 3-epoch warmup. We use a total batch size of 64 across 4 NVIDIA H20 GPUs.

The full stage-1 objective is a weighted sum of per-component losses with weights chosen to balance their gradient scales. The velocity-matching loss $\mathcal{L}_{\text{dec}}$ has weight 40. Inside $\mathcal{L}_{\text{perc}}$, the agent class and box prediction losses have weights 10 and 1, the BEV semantic loss has weight 14, and the IL head L1 loss has weight 10. The selector loss $\mathcal{L}_{\text{sel}}$ has weight 10.

In stage 1, we mainly treat the mode selector supervision as an auxiliary signal to refine scene features, and use a reduced version of selector loss. The per-timestep TTC-time and ego-area heads are not included, and the mode selector is trained to predict v2 subscores NC, DAC, EP, TTC, and HC.

Numerical Stability of $\mathcal{L}_{\text{dec}}$. The velocity conversion $\boldsymbol{v}_\theta = (\boldsymbol{x}_\theta - \boldsymbol{z}_t)/(1-t)$ in Sec. 3.2 diverges as $t \to 1$. To stabilize training, we clip the denominator to a minimum of 0.05, computing $\boldsymbol{v}_\theta = (\boldsymbol{x}_\theta - \boldsymbol{z}_t)/\max(1-t, 0.05)$ when sampling t near 1. The clip is loss-only and does not affect the inference ODE solver.

F.3.4 Stage 2: Mode Selector Finetune

In stage 2, we freeze every component except the mode selector and finetune the selector to close the gap between vocabulary trajectories seen at stage-1 and decoder proposals seen at inference. We train with a batch size of 256 across 8 H20 GPUs for 2 epochs. Other settings match stage 1.

For each training scene, we generate decoder proposals using the same sampling configuration as inference, and combine them with 32 random vocabulary trajectories drawn under inverse-density weighting. Generated proposals are scored online by the NAVSIM simulator to obtain ground-truth subscores. The selector is then trained to predict these subscores together with the per-timestep TTC-time and ego-area arrays under $\mathcal{L}_{\text{sel}}$ defined in App. F.2.4.

F.4 Inference

F.4.1 Default Sampling Configuration

Unless otherwise stated, all experiments share the following sampling configuration. We use the Euler scheduler with 20 denoising steps, discretizing time $t \in [0, 1]$. The sampling uses the CFG scale w_g=5 and takes the IL head output as the anchor for zero-shot anchored sampling. The two sampling controls r_{high} and t_{init} are drawn independently and uniformly per proposal, with the target score in $[s_{\min}, s_{\max}]$=$[0.9, 1.0]$ and the initial noise level in $[t_{\min}, t_{\max}]$=$[0.5, 0.9]$. We generate 60 proposals and rank them with the mode selector under unit aggregation weights w_k=1 (App. F.2.4). The same model trained once on `navtrain` is used for both NAVSIM v1 and v2 evaluation.

F.4.2 Per-Experiment Sampling Settings

A few experiments deviate from the default. The differences are limited to the number of proposals, the $t_{\min}$ lower bound, and how the single-proposal setting in Tab. 1 is constructed. We summarize the settings in Tab. 10.

The single-proposal column of Tab. 1 is the only configuration that fixes r_{high} and t_{init} to representative values, since random sampling would inject noise into a single-shot evaluation. Every other single-proposal experiment draws the two controls from the default ranges, matching the behavior at multi-proposal inference. Tab. 3 uses 64 proposals to be comparable to iPad [17].

F.4.3 Reward Subset for Classifier-Free Guidance

The high-reward condition r_{high} used for classifier-free guidance covers a subset of $\mathcal{R}$ rather than all reward signals. The subset comprises three scalar rewards (NC, HC, and the target PDM score) and the two per-timestep rewards (TTC-time and ego-area). All entries except the target score are set to their maximal values, signaling no collision, full history comfort, on-road and on-route at every timestep, and no near collision at every timestep. The target score is randomized over $[s_{\min}, s_{\max}]$

Table 10: Inference settings per experiment. Default refers to the configuration in App. F.4.1.

Experiment	# proposals	r_{high}	t_{init}
Tab. 1 (multi-proposal)	60	default	default
Tab. 1 (single-proposal)	1	fixed 0.95	fixed 0.5
Tab. 2	60	default	$[0.75, 0.9]$
Tab. 3	64	default	default
Tab. 5a, Tab. 5b	1	default	default
Fig. 4	64	default	default
Fig. 5	1	default	default
Fig. 6	64	default	default
Fig. 7	1	default	default
Fig. 8 (left)	10	varies (sweep)	varies (sweep)
Fig. 8 (right)	16	varies (sweep)	default

and acts as the diversification axis described in Sec. 3.4. The remaining rewards in $\mathcal{R}$ are dropped from r_{high} by replacing their embedding with the per-reward null token $\boldsymbol{n}_k$ in Eq. 4.

The choice of subset reflects what we can confidently prescribe at inference time. The optimal value of most rewards is scene-dependent and cannot be set in advance, so we leave them empty and let the model balance them through the learned $p(a|r)$. The PDM target score serves as an overall indicator for high-quality actions. We additionally pin the rewards that encode hard constraints, namely NC and the two per-timestep arrays for safety and drivable-area compliance, to enforce them across all proposals. HC is included as a stable, easy-to-satisfy regularizer that smooths the dynamics of the resulting trajectories.

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 contributions stated in the abstract and introduction are supported by the method section and the experiment section.

   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: Limitations are discussed in App. A.

   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 present new 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 method section describes the architecture, training objective, and inference procedure, and App. F provides full implementation details.

   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: Code and checkpoints are released at `https://github.com/hku-sail/FlowR2A`. App. F provides the implementation details needed to reproduce the results.

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: Training and inference details are provided in App. F.3 and App. F.4.

   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: Following standard practice on NAVSIM, we report scores from a single trained model.

   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: Training compute is reported in App. F.3 and inference latency in App. C.2.

   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: We have reviewed the NeurIPS Code of Ethics and the research conforms to it.

   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: We discuss broader impacts in App. A.

    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 data or models with high misuse risk.

    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: All existing assets used in the paper are properly cited and used under their public research licenses.

    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 submission does not release new 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.

    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 crowdsourcing or research with 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: LLMs are not used as a component of the core method.

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.

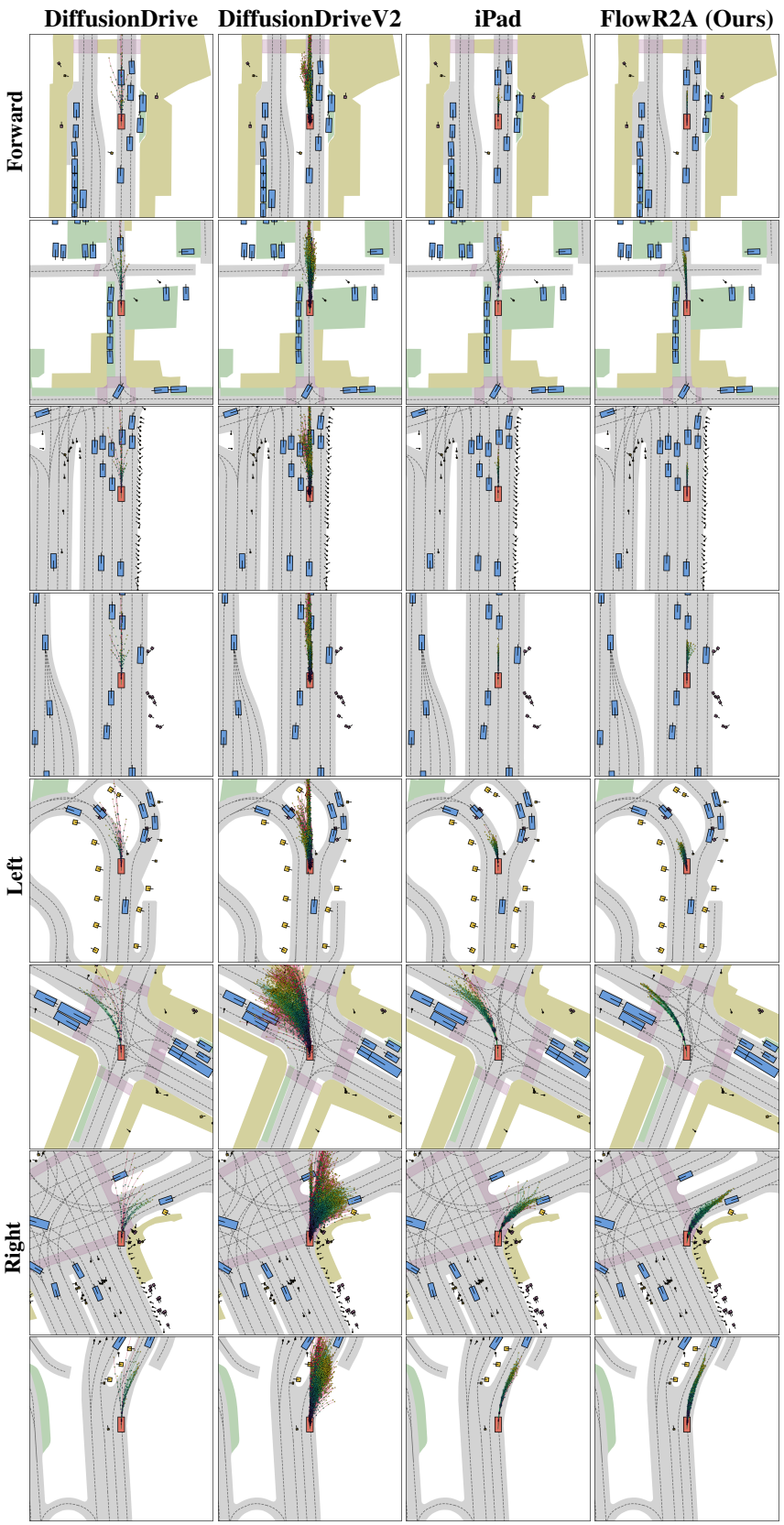


Figure 12: Full qualitative comparison part 1 of 3. Trajectories are colored by PDMS from 0 (red) to 1 (green). Scenes are grouped by driving command labeled left.

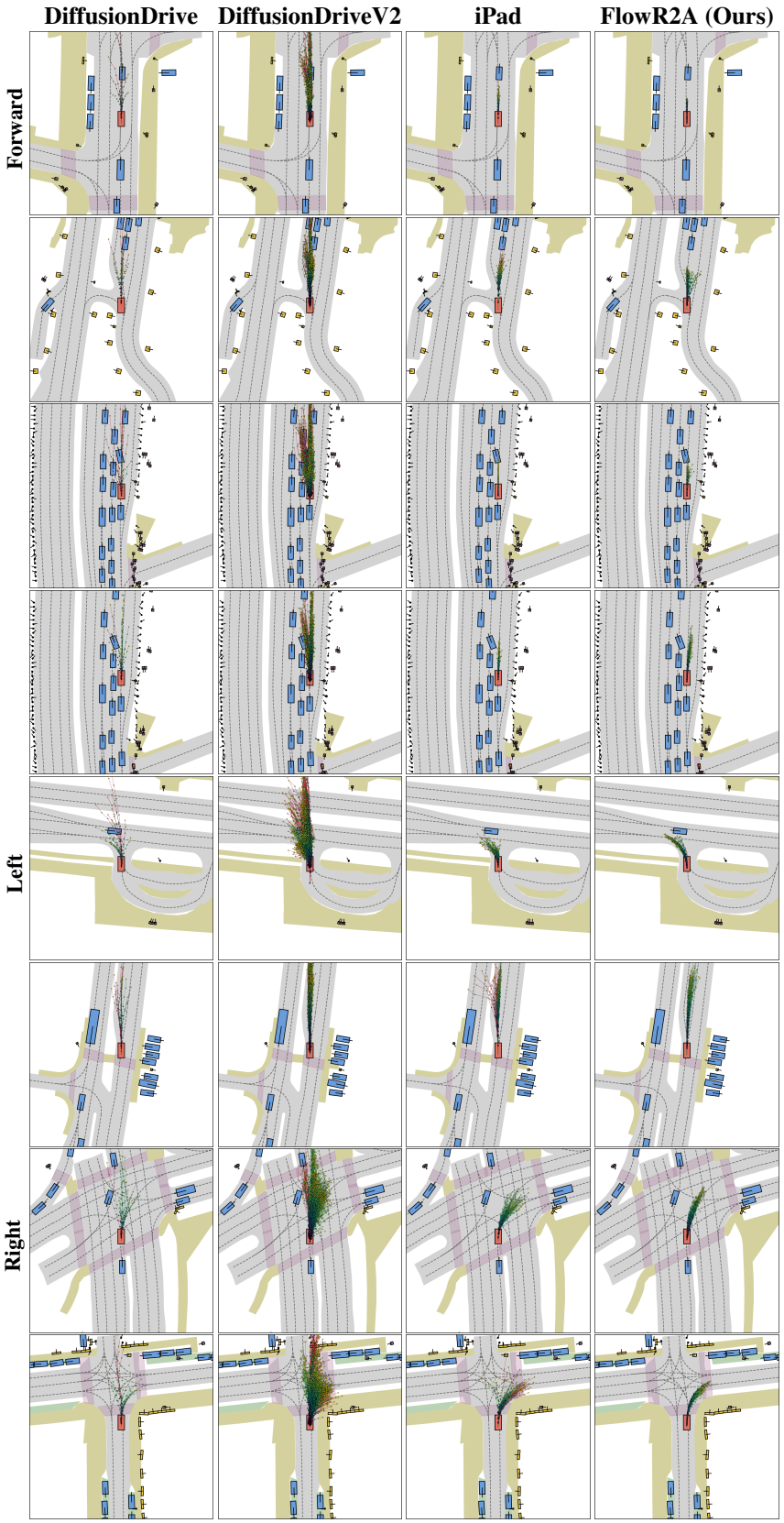


Figure 13: Full qualitative comparison part 2 of 3. Trajectories are colored by PDMS from 0 (red) to 1 (green). Scenes are grouped by driving command labeled left.

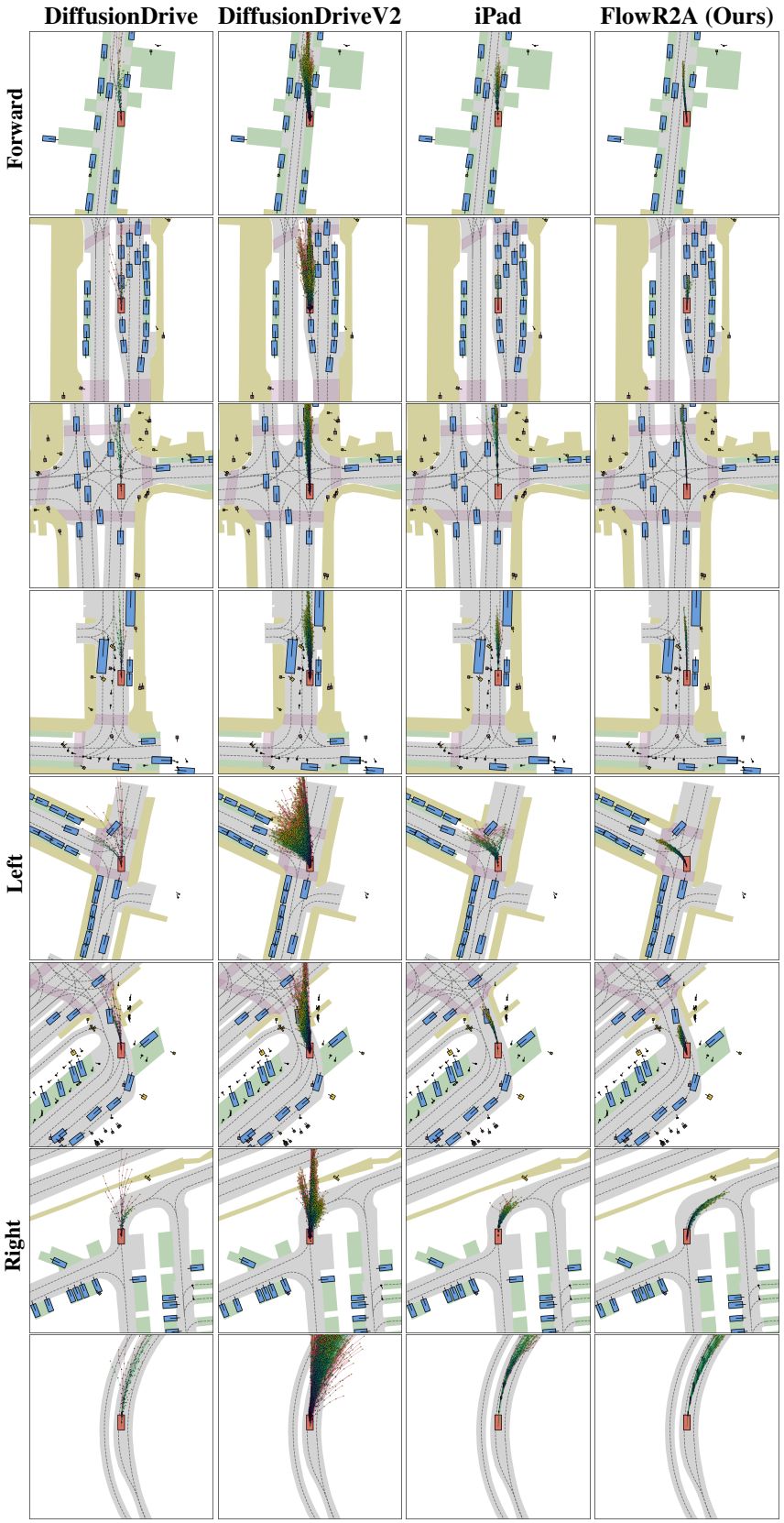


Figure 14: Full qualitative comparison part 3 of 3. Trajectories are colored by PDMS from 0 (red) to 1 (green). Scenes are grouped by driving command labeled left.