TEST-TIME PLANNING FOR ROBUST IMITATION: LEARNING TO SEARCH FOR RECOVERY IN AUTONOMOUS DRIVING

Rajat Sahay[*1,2] **Volker Fischer**[2] **Thomas Brox**[1]
[1]University of Freiburg [2]Bosch Center for Artificial Intelligence

ABSTRACT

Behavioral cloning trains policies to mimic expert demonstrations, but provides no mechanism for recovery when the agent deviates from the training distribution at test time. We address this limitation through a new paradigm: test-time planning. Our approach learns a latent world model and reward model from expert demonstrations, then uses these components at inference to search for corrective actions when the base policy begins to fail. Concretely, we combine a hierarchical diffusion policy trained via imitation learning with Model-Predictive Control (MPC) in the learned latent space, enabling the ego vehicle to plan recovery trajectories during inference without additional human supervision. We evaluate our method on the nuPlan and CARLA planning benchmarks demonstrating that our test-time planning approach is consistently able to recover from distribution shifts that cause the base policy to fail. Our results suggest that integrating search-based planning with learned world models provides a robust framework for handling inference-time distribution shifts in embodied agents.

1 INTRODUCTION

Imitation learning offers a compelling paradigm for autonomous driving, rather than hand-engineering reward functions or rules (Dauner et al., 2023; Khanzada & Kwon, 2025), we learn policies directly from expert demonstrations. Behavioral cloning (BC), the simplest form of imitation learning, trains a policy to maximize the likelihood of expert actions at expert states (Torabi et al., 2018). Modern instantiations using expressive architectures like diffusion models have achieved strong performance on driving benchmarks Zheng et al. (2025); Tan et al. (2025) when the test distribution matches training. However, BC provides no mechanism for recovery when the agent inevitably deviates from the expert's state distribution at test time.

This limitation is fundamental. When an imitation policy makes even a small mistake—due to limited demonstrations, optimization errors, or distribution shift—it enters states unlike those seen during training, where it is likely to make further errors. This compounding error manifests as brittle behavior: a policy that performs well on average but fails catastrophically when perturbed from familiar states (Li et al., 2024; Jaeger et al., 2025). The dominant approaches to handling test-time distribution shift involve updating model parameters: test-time training adapts representations using self-supervised objectives (Tan et al., 2025), while test-time adaptation adjusts batch normalization statistics or lightweight adapters (Jia et al., 2023). However, for safety-critical autonomous driving, parameter updates raise concerns about stability—a modified policy may behave unpredictably across the entire state space, not just where adaptation was needed.

We explore an alternative: *test-time planning*. Rather than updating what the policy *is*, we adapt what the policy *does* by searching for corrective actions at inference time. This requires learning not just a policy, but the components needed for search: a world model that predicts action consequences, and a reward model that evaluates how "expert-like" a state is. Given these components, we perform local search during inference to correct the base policy's mistakes, planning recovery trajectories without modifying any parameters. Our approach builds on (Jain et al., 2025), which introduced this learning-to-search framework for robotic manipulation. At test time, when the ego vehicle deviates

[*]Correspondence to: rajat.sahay@de.bosch.com

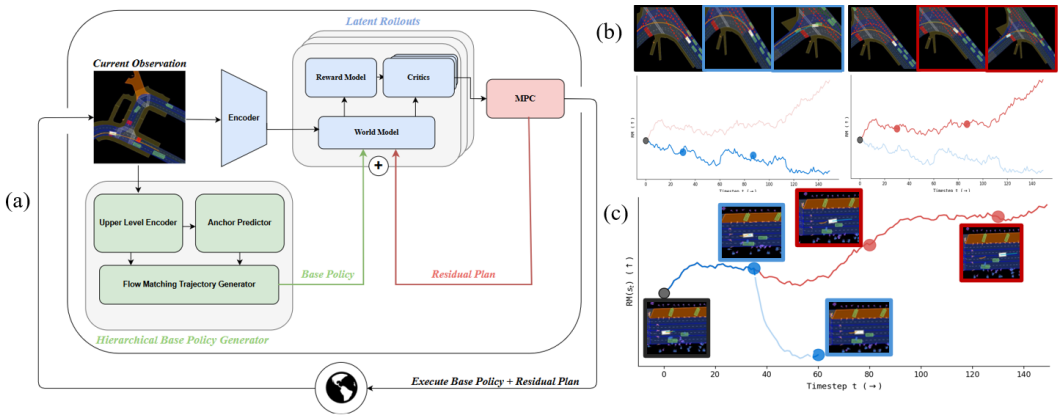

Figure 1: (a) An overview of how our model works during inference time. (b) By visualizing candidate rollouts within a learned world model, our method exposes how the reward function evaluates potential hazards. This transparency allows users to interpret why a base trajectory is rejected as unsafe (blue) and how the model identifies a corrective, high-reward path (red), providing crucial auditability for safety-critical planning. (c) Our learned reward model is able to identify nuanced failures (blue) and generate a new trajectory (red) to counterfactually avoid them.

from expert-like behavior, we use Model Predictive Control (MPC) (Kouvaritakis & Cannon, 2016) to search for residual corrections to the base policy's planned trajectory. The search operates in a learned latent space, evaluating candidates against the reward model and selecting actions that return the vehicle to the expert's state distribution.

This framing offers several advantages for autonomous driving. First, corrections are *local*—affecting only the current decision, not behavior elsewhere. Second, planning is *interpretable*—we can inspect candidate trajectories and their scores. Third, planning is *constraint-compatible*—we can reject unsafe trajectories before execution rather than hoping an adapted policy respects them.

We validate this approach on the widely-used nuPlan and CARLA benchmarks for autonomous planning. Our experiments demonstrate that by shifting the burden of correction from training-time updates to test-time search, we achieve higher safety and improved generalization. We show that even when the base policy fails, our planner effectively navigates the vehicle back to an expert-like distribution, confirming the efficacy of test-time optimization for safety-critical driving applications.

2 Background and Experimental Setup

Learning-based planning for autonomous driving spans imitation learning (Renz et al., 2022; Zheng et al., 2025; Tan et al., 2025), reinforcement learning (Zhang et al., 2021; Jaeger et al., 2025), and hybrid approaches (Cheng et al., 2024; Dauner et al., 2023). Imitation learning methods train neural planners via behavioral cloning on expert demonstrations, achieving strong performance with modern architectures like Transformers and diffusion models. However, these approaches struggle with compounding errors and recovery from off-distribution states. Reinforcement learning addresses some limitations by optimizing closed-loop objectives (Jaeger et al., 2025; Khanzada & Kwon, 2025), but requires extensive online interaction and careful reward shaping. Hierarchical formulations separate high-level behavior selection from low-level control, improving abstraction but remaining limited without explicit planning (Jin et al., 2025). Our work combines hierarchical planning with learning-to-search: we use a hierarchical diffusion policy as the base trajectory generator, then augment it with learned world and reward models that enable test-time planning for recovery (Jain et al., 2025). This integration preserves the structured representation of hierarchical approaches while adding the ability to recover from distribution shift at inference.

We evaluate our method against state-of-the-art planners on four benchmarks spanning real-world and simulated driving. From nuPlan (Dauner et al., 2023), we use Val14 (1118 scenarios across 14 types), Test14-random (200+ randomly selected scenarios), and Test14-hard (272 scenarios selected as worst-performing cases for rule-based planners, targeting dense traffic and complex intersections).

We also evaluate on CARLA Longest6 v2 (Chitta et al., 2022), which consists of 36 routes up to 2 km long with 5-21 safety-critical scenarios per route and background traffic driving up to 80 km/h.

3 TEST-TIME PLANNING VIA LEARNING TO SEARCH

Fig 1a shows our approach for enabling recovery from distribution shift at test time. We combine a hierarchical diffusion policy trained via imitation learning with a learning-to-search framework that performs local planning when the base policy begins to deviate from expert-like behavior.

Hierarchical Base Policy Generator Our base policy generator employs a hierarchical architecture that decomposes trajectory generation into three stages. First, an upper-level encoder processes ego history, neighboring agent trajectories, and vectorized map features through temporal transformers and cross-modal attention, producing a context representation f_{up}. Second, an anchor predictor outputs K sparse probabilistic waypoints via a Mixture Density Network (MDN) (Bishop, 1994), capturing multi-modal intentions over the planning horizon. Finally, a conditional flow matching model generates dense trajectories by learning a vector field $v_\theta(\tau_t, t, \mathcal{W}, c, f_{\text{up}})$ that transports samples from a Gaussian prior to the trajectory distribution, conditioned on anchors $\mathcal{W}$ and context f_{up}. The model is trained in two stages: supervised anchor prediction with NLL loss and joint flow matching training with anchor consistency and smoothness regularization. At inference, we sample diverse trajectory candidates by varying anchor samples and flow prior noise, which are then evaluated by downstream components. Full architectural and training details are provided in Appendix A.

Test-Time Planning for Recovery While the hierarchical base policy provides diverse trajectory candidates, it lacks a mechanism for recovery when these candidates deviate from the expert distribution. We address this by learning three additional components from a mixture of expert demonstrations $\mathcal{D}_{\text{expert}}$ and base policy rollouts $\mathcal{D}_{\text{policy}}$: a latent world model, a reward model, and a critic. The world model follows an RSSM architecture with an encoder enc that maps observations to latent states $z_t = \text{enc}(z_{t-1}, a_{t-1}, o_t)$ and a dynamics model f that predicts transitions $\hat{z}_{t+1} \sim f(z_t, a_t)$. The reward model $\text{RM} : \mathcal{Z} \to \mathbb{R}$ scores latent states by how expert-like they are, trained as a discriminator between expert and learner rollouts:

$$\mathcal{L}_{\text{RM}} = \mathbb{E}_{(z,a,o)\sim\mathcal{D}_{\text{policy}}}[\text{RM}(\text{enc}(z,a,o))] - \mathbb{E}_{(z,a,o)\sim\mathcal{D}_{\text{expert}}}[\text{RM}(\text{enc}(z,a,o))] \tag{1}$$

The critic $V : \mathcal{Z} \to \mathbb{R}$ estimates long-horizon returns beyond the planning window. We use an ensemble of critics with an uncertainty penalty to encourage conservative planning. At inference, we use an MPC algorithm to search for residual corrections to the base policy's plan. Given a state representation and nominal plan $a^{\text{base}}_{t:t+k}$ from the hierarchical policy, we sample candidate residuals from a proposal distribution, roll them out in the world model, and score the resulting trajectories:

$$\Delta^\star_{t:t+k} = \arg\max_{\Delta_{t:t+k}} \sum_{h=0}^{k-1} \gamma^h \text{RM}(z_{t+h}) + \gamma^k V(z_{t+k}) \tag{2}$$

Here, z_{t+h} denotes the latent state obtained by rolling out the corrected actions $a^{\text{base}}_{t:t+k} + \Delta_{t:t+k}$ in the learned world model. The sampling distribution is iteratively refined toward higher-scoring residuals over multiple iterations, after which we execute the first corrected action $a^\star_t = a^{\text{base}}_t + \Delta^\star_t$ and re-plan at the next timestep. This residual formulation is conservative, which means that when the base policy performs well, MPC converges to near-zero corrections. Additional training and implementation details are provided in Appendix B.

4 EXPERIMENTS

This section shows the advantages of our method over several other planning approaches on nuPlan (Dauner et al., 2023), and CARLA longest6 v2 (Chitta et al., 2022). We provide more results, including qualitative examples, in Appendix C.

nuPlan Table 1 reports official closed-loop scores (0-100) in both non-reactive (NR) and reactive (R) settings. We outperform pure imitation (Tan et al., 2025), hybrid (Zheng et al., 2025), and hierarchical (Jin et al., 2025) baselines. To isolate the contribution of test-time planning, we ablate

Table 1: Closed-loop planning performance on nuPlan benchmarks. **Bold** indicates best performance, underline indicates second best.

Method	Type	Val14		Test14-hard		Test14	
		NR	R	NR	R	NR	R
Log Replay (LQR)	Human	93.53	80.32	85.96	68.80	94.03	75.86
Flow Planner	IL	90.43	83.31	76.47	70.42	89.88	82.93
Diff. Planner w/refine	Hybrid	94.26	92.90	78.87	82.00	94.80	91.75
HiDrive	Hierarchical	93.62	**93.15**	81.41	83.18	93.71	92.31
Ours	–	91.29	84.93	78.19	72.62	90.13	84.39
Ours (w/L2S)	–	**95.81**	90.44	**81.89**	**85.47**	**95.61**	**90.16**

Table 2: Closed-loop performance on CARLA Longest6 v2. DS: Driving Score (↑), RC: Route Completion (↑), Ped: Pedestrian collisions (↓), Veh: Vehicle collisions (↓), MS: Min-speed infractions (↓), Time: Average completion time (↓). **Bold** indicates best performance.

Planner	Type	DS ↑	RC ↑	Ped ↓	Veh ↓	MS ↓	Time ↓
PlanT	IL	62	96	0.07	0.43	3.18	18
Think2Drive	RL	7	39	0.97	2.55	18.05	**6**
LBC	TTA	36	52	0.39	3.78	5.93	10
Roach	RL + TTA	22	77	0.45	2.42	7.12	7
CaRL	RL	64	82	**0.01**	0.36	1.71	8
Ours	–	57	73	0.38	1.95	4.07	7
Ours (w/L2S)	–	**69**	**87**	0.06	**0.28**	**1.24**	11

our method with and without the learning-to-search (L2S) module. The substantial improvement with L2S demonstrates the effectiveness of our method, and shows that test-time planning effectively recovers from situations where the hierarchical base policy alone would fail.

CARLA Table 2 reports results on CARLA Longest6 v2 (Chitta et al., 2022), where Driving Score (DS) multiplies Route Completion (RC) by a penalty for infractions. We compare against a different set of baselines here, as methods evaluated on nuPlan and CARLA typically come from distinct research communities. We restrict comparison to methods operating solely in BEV space, excluding approaches that additionally use sensor observations as input, which would constitute an unfair comparison given the additional modality unavailable to our model. Our method achieves 69 DS, outperforming prior RL and IL approaches while drastically reducing pedestrian and vehicular collisions. We also compare against two state-of-the-art test-time adaptation (TTA) methods and find that our approach outperforms both, highlighting the advantage of learning the components for search rather than adapting policy parameters at inference. We attribute the safety improvements to test-time trajectory re-evaluation: by reasoning about the dynamics of surrounding agents in the learned world model, our planner updates its trajectories to avoid collisions.

5 Discussion

We presented an approach for robust imitation learning in autonomous driving that performs test-time planning to improve the base policy during inference. By learning a world model and reward model from expert demonstrations and base policy rollouts, we enable recovery from distribution shift through search in learned latent space. Our results show consistent improvements over the base hierarchical policy, particularly in challenging scenarios where behavioral cloning alone fails.

Our framework is also architecturally compatible with foundation models for both policy and world modeling. The base policy could be replaced with a vision-language-action model pretrained on diverse driving data, potentially improving generalization to novel scenarios. Similarly, the world model could leverage video prediction foundation models trained on internet-scale data, providing broader coverage of possible state transitions. We hypothesize that such integrations could amplify the benefits of test-time planning- a more general base policy would require corrections less frequently, while a more general world model would enable recovery from a wider range of failures. We leave this integration as a promising direction for future work.

REFERENCES

Christopher M Bishop. Mixture density networks. 1994.

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

Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE transactions on pattern analysis and machine intelligence*, 45(11):12878–12895, 2022.

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

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models, 2022. URL https://arxiv.org/abs/2010.02193.

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

Arnav Kumar Jain, Vibhakar Mohta, Subin Kim, Atiksh Bhardwaj, Juntao Ren, Yunhai Feng, Sanjiban Choudhury, and Gokul Swamy. A smooth sea never made a skilled sailor: Robust imitation via learning to search. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Xiaosong Jia, Yulu Gao, Li Chen, Junchi Yan, Patrick Langechuan Liu, and Hongyang Li. Driveadapter: Breaking the coupling barrier of perception and planning in end-to-end autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7953–7963, 2023.

Xuanjin Jin, Chendong Zeng, Shengfa Zhu, Chunxiao Liu, and Panpan Cai. Hi-drive: Hierarchical pomdp planning for safe autonomous driving in diverse urban environments. *IEEE Robotics and Automation Letters*, 2025.

Feeza Khan Khanzada and Jaerock Kwon. Indrive: Reward-free world-model pretraining for autonomous driving via latent disagreement. *arXiv preprint arXiv:2512.18850*, 2025.

Basil Kouvaritakis and Mark Cannon. Model predictive control. *Switzerland: Springer International Publishing*, 38(13-56):7, 2016.

Qifeng Li, Xiaosong Jia, Shaobo Wang, and Junchi Yan. Think2drive: Efficient reinforcement learning by thinking with latent world model for autonomous driving (in carla-v2). In *European Conference on Computer Vision*, pp. 142–158. Springer, 2024.

Allen Z Ren, Justin Lidard, Lars L Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

Katrin Renz, Kashyap Chitta, Otniel-Bogdan Mercea, A Koepke, Zeynep Akata, and Andreas Geiger. Plant: Explainable planning transformers via object-level representations. *arXiv preprint arXiv:2210.14222*, 2022.

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

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

Zhejun Zhang, Alexander Liniger, Dengxin Dai, Fisher Yu, and Luc Van Gool. End-to-end urban driving by imitating a reinforcement learning coach. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 15222–15232, 2021.

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

A HIERARCHICAL BASE POLICY GENERATOR

This section provides additional details on the hierarchical base policy generator summarized in the main text.

A.1 OVERVIEW

The hierarchical policy decomposes trajectory generation into two stages: sparse anchor prediction followed by dense trajectory generation via conditional flow matching. This decomposition enables efficient coverage of the multi-modal trajectory distribution while maintaining interpretability through explicit subgoal representations. The architecture consists of four modules: (1) an upper-level encoder that extracts scene context, (2) an anchor predictor that generates sparse probabilistic waypoints, and (3) a lower-level trajectory generator that produces dense trajectories conditioned on anchors.

A.2 PROBLEM FORMULATION

Let $\mathcal{S}_t = \{E_t, \mathcal{A}_t, \mathcal{M}\}$ denote the scene context at time t, where $E_t = \{e_{t-T_h+1}, \ldots, e_t\}$ is the ego vehicle state history over T_h timesteps with $e_\tau = (x_\tau, y_\tau, \theta_\tau, v_\tau)$ representing position, heading, and velocity; $\mathcal{A}_t = \{A_t^{(i)}\}_{i=1}^{N_t}$ is the set of neighboring agent histories; and $\mathcal{M} = \{\ell_1, \ldots, \ell_L\}$ is the local map represented as lane polylines. Our goal is to learn a generative model $p_\theta(\tau|\mathcal{S}_t)$ over future ego trajectories $\tau = \{e_{t+1}, \ldots, e_{t+T_f}\}$ spanning prediction horizon T_f.

A.3 ARCHITECTURE

A.3.1 UPPER-LEVEL ENCODER

The upper-level encoder extracts a joint context representation $f_{\text{up}} \in \mathbb{R}^d$ from the scene context $\mathcal{S}_t$ using modality-specific encoders followed by cross-modal fusion.

Ego Encoder. We encode the ego history using a temporal transformer:

$$h_{\text{ego}} = \text{TemporalTransformer}\left(\{W_e \cdot e_\tau + \text{PE}(\tau)\}_{\tau=t-T_h+1}^{t}\right) \tag{3}$$

where $W_e \in \mathbb{R}^{d \times d_e}$ projects ego states to the model dimension and $\text{PE}(\cdot)$ denotes sinusoidal positional encoding. The final ego embedding is obtained via mean pooling: $f_{\text{ego}} = \text{MeanPool}(h_{\text{ego}})$.

Agent Encoder. Each neighboring agent trajectory is first transformed to ego-centric coordinates via $\tilde{a}_\tau^{(i)} = R_t^{-1}(a_\tau^{(i)} - e_t)$ where R_t is the rotation matrix corresponding to ego heading θ_t. Agent trajectories are encoded independently and aggregated via self-attention:

$$f_{\text{agent}}^{(i)} = \text{MeanPool}\left(\text{TemporalTransformer}\left(\{W_a \cdot \tilde{a}_\tau^{(i)} + \text{PE}(\tau)\}\right)\right) \tag{4}$$

$$f_{\text{agents}} = \text{MultiHeadSelfAttn}\left(\{f_{\text{agent}}^{(i)}\}_{i=1}^{N_t}\right) \tag{5}$$

For scenes with variable numbers of agents, we apply masking and use a maximum agent count $N_{\max}$ with zero-padding.

Map Encoder. Each lane polyline $\ell_j = \{p_1^{(j)}, \ldots, p_{M_j}^{(j)}\}$ is encoded using a polyline transformer. Lane point features include ego-centric position, tangent direction, and lane type embeddings. Lane embeddings are aggregated via self-attention:

$$f_{\text{map}} = \text{MultiHeadSelfAttn}\left(\{\text{MeanPool}(h_{\text{lane}}^{(j)})\}_{j=1}^{L}\right) \tag{6}$$

The final context representation is obtained via cross-attention fusion: $f_{\text{up}} = \text{CrossAttn}(Q = f_{\text{ego}}, K = V = [f_{\text{agents}}; f_{\text{map}}])$. To stabilize training, we maintain an exponential moving average (EMA) copy of the encoder parameters with momentum $\alpha = 0.999$. The EMA encoder produces $f_{\text{up}}^{\text{EMA}}$, which is used for conditioning downstream modules during training.

A.3.2 ANCHOR PREDICTOR

The anchor predictor generates K sparse probabilistic waypoints $\mathcal{W} = \{w_1, \ldots, w_K\}$ at uniformly spaced time indices $\{t_k\}_{k=1}^K$ where $t_k = t + k \cdot \lfloor T_f/K \rfloor$. We model anchors using a Mixture Density Network (MDN) with M Gaussian components per anchor:

$$p(w_k | f_{\text{up}}^{\text{EMA}}) = \sum_{m=1}^{M} \pi_{k,m} \cdot \mathcal{N}(w_k; \mu_{k,m}, \Sigma_{k,m}) \tag{7}$$

The MDN head outputs mixture weights $\boldsymbol{\pi}_k \in \Delta^{M-1}$ via softmax, component means $\mu_{k,m} \in \mathbb{R}^2$, and diagonal covariances $\Sigma_{k,m} = \text{diag}(\sigma_{k,m}^2)$ with $\sigma = \exp(\log \sigma)$ to ensure positivity. To sample an anchor, we first sample a component index $m^* \sim \text{Categorical}(\boldsymbol{\pi}_k)$, then sample position $w_k \sim \mathcal{N}(\mu_{k,m^*}, \Sigma_{k,m^*})$. For multi-modal trajectory generation, we sample N_{samples} independent anchor sets.

A.3.3 LOWER-LEVEL TRAJECTORY GENERATOR

The lower-level generator produces dense trajectories $\tau = \{e_{t+1}, \ldots, e_{t+T_f}\}$ conditioned on sampled anchors $\mathcal{W}$ and context f_{up} using conditional flow matching.

Flow Matching Formulation. Let $\tau_0 \sim \mathcal{N}(0, I)$ denote a sample from the prior and $\tau_1 = \tau^{\text{GT}}$ denote the ground-truth trajectory. We define a probability path $p_t(\tau)$ interpolating between prior and data distribution for $t \in [0, 1]$. The flow is generated by a time-dependent vector field $v_t(\tau)$ parameterized by a neural network $v_\theta(\tau, t, \mathcal{W}, f_{\text{up}})$ conditioned on anchors and context.

Conditional Flow Matching Objective. Using a linear interpolation path $\tau_t = (1-t)\tau_0 + t\tau_1$, the conditional vector field is $u_t(\tau|\tau_1) = \tau_1 - \tau_0$. The training objective is:

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t \sim \mathcal{U}[0,1], \tau_0 \sim \mathcal{N}(0,I), \tau_1 \sim p_{\text{data}}} \left[\| v_\theta(\tau_t, t, \mathcal{W}, f_{\text{up}}) - (\tau_1 - \tau_0) \|_2^2 \right] \tag{8}$$

The vector field network is a transformer that embeds the input trajectory with sinusoidal time encoding, applies self-attention and cross-attention to conditioning features $f_{\text{cond}} = [f_{\text{up}}; \text{Enc}_{\mathcal{W}}(\mathcal{W})]$, and outputs velocity predictions via linear projection.

Anchor Conditioning via Guidance. To ensure generated trajectories respect predicted anchors, we use classifier-free guidance. During training, we randomly drop anchor conditioning with probability $p_{\text{uncond}} = 0.1$ using a learned null embedding. At inference, we modify the vector field:

$$\tilde{v}_\theta = v_\theta(\tau_t, t, \mathcal{W}, f_{\text{up}}) + \omega \left(v_\theta(\tau_t, t, \mathcal{W}, f_{\text{up}}) - v_\theta(\tau_t, t, \varnothing, f_{\text{up}}) \right) \tag{9}$$

where $\omega > 0$ is the guidance scale. We additionally apply explicit anchor projection at intermediate steps: $\tau_t' = \tau_t - \lambda_{\text{anchor}} \nabla_{\tau_t} E_{\text{anchor}}(\tau_t, \mathcal{W})$ where $E_{\text{anchor}}(\tau, \mathcal{W}) = \sum_{k=1}^K \| \tau[t_k] - w_k \|_2^2$.

Sampling. At inference, we generate trajectories by solving the ODE $\frac{d\tau_t}{dt} = v_\theta(\tau_t, t, \mathcal{W}, f_{\text{up}})$ from $t = 0$ to $t = 1$ using fixed-step Euler integration: $\tau_{t+\Delta t} = \tau_t + \Delta t \cdot \tilde{v}_\theta$ with N_{steps} integration steps.

Training with Anchor Consistency. In addition to the flow matching objective, we add an anchor consistency loss on the fully denoised trajectory:

$$\mathcal{L}_{\text{consist}} = \mathbb{E}_{\tau_0, \tau_1} \left[\sum_{k=1}^{K} \| \hat{\tau}_1[t_k] - \texttt{sg}(w_k) \|_2^2 \right] \tag{10}$$

where $\hat{\tau}_1$ is obtained by integrating the learned flow and $\texttt{sg}(\cdot)$ denotes stop-gradient on anchors. We also add smoothness regularization $\mathcal{L}_{\text{smooth}} = \sum_{i=2}^{T_f - 1} \| \hat{\tau}_1[i+1] - 2\hat{\tau}_1[i] + \hat{\tau}_1[i-1] \|_2^2$ to penalize jerky trajectories.

A.4 TRAINING PROCEDURE

Training proceeds in three stages with curriculum scheduling.

Algorithm 1 Hierarchical Policy Inference

Scene context $\mathcal{S}_t$, number of samples N_{samples}, guidance scale ω, integration steps N_{steps}
Trajectory candidates $\{\tau^{(n)}\}_{n=1}^{N_{\text{samples}}}$

$f_{\text{up}} \leftarrow \text{ENCODER}(\mathcal{S}_t)$
$\{\boldsymbol{\pi}_k, \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k\}_{k=1}^{K} \leftarrow \text{ANCHORPREDICTOR}(f_{\text{up}})$ $n = 1$ to N_{samples} $k = 1$ to K
$m^* \sim \text{Categorical}(\boldsymbol{\pi}_k)$
$w_k^{(n)} \sim \mathcal{N}(\mu_{k,m^*}, \Sigma_{k,m^*})$
$\mathcal{W}^{(n)} \leftarrow \{w_1^{(n)}, \ldots, w_K^{(n)}\}$
$\tau_0^{(n)} \sim \mathcal{N}(0, I)$ $s = 0$ to $N_{\text{steps}} - 1$
$t \leftarrow s/N_{\text{steps}}$
$v_{\text{cond}} \leftarrow v_\theta(\tau_t^{(n)}, t, \mathcal{W}^{(n)}, f_{\text{up}})$
$v_{\text{uncond}} \leftarrow v_\theta(\tau_t^{(n)}, t, \varnothing, f_{\text{up}})$
$\tilde{v} \leftarrow v_{\text{cond}} + \omega(v_{\text{cond}} - v_{\text{uncond}})$
$\tau_{t+1/N_{\text{steps}}}^{(n)} \leftarrow \tau_t^{(n)} + \tilde{v}/N_{\text{steps}}$
$\tau^{(n)} \leftarrow \tau_1^{(n)}$
$\{\tau^{(n)}\}_{n=1}^{N_{\text{samples}}}$

Stage 1: Anchor Predictor Training. We extract ground-truth anchors from expert trajectories at the specified time indices. The anchor predictor is trained to minimize negative log-likelihood:

$$\mathcal{L}_{\text{anchor}} = -\sum_{k=1}^{K} \log p(w_k^{\text{GT}} | f_{\text{up}}^{\text{EMA}}) \tag{11}$$

We add a reconstruction loss $\mathcal{L}_{\text{recon}} = \sum_{k=1}^{K} \min_m \|\mu_{k,m} - w_k^{\text{GT}}\|_2^2$ to encourage mixture means to be close to ground truth, and a diversity regularization $\mathcal{L}_{\text{div}} = -\sum_{k=1}^{K} \sum_{m \neq m'} \|\mu_{k,m} - \mu_{k,m'}\|_2$ to prevent mode collapse.

Stage 2: Flow Matching Pretraining. We pretrain the lower-level trajectory generator using ground-truth anchors as conditioning. This allows the flow network to learn trajectory dynamics before being exposed to noise from sampled anchors. We use teacher forcing, feeding ground-truth previous states rather than model predictions.

Stage 3: Joint Fine-Tuning. We jointly train all components end-to-end. The total loss combines all objectives:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CFM}} + \lambda_{\text{anchor}}\mathcal{L}_{\text{anchor}} + \lambda_{\text{consist}}\mathcal{L}_{\text{consist}} + \lambda_{\text{smooth}}\mathcal{L}_{\text{smooth}} + \lambda_{\text{recon}}\mathcal{L}_{\text{recon}} + \lambda_{\text{div}}\mathcal{L}_{\text{div}} \tag{12}$$

During joint training, we transition from ground-truth anchors to sampled anchors for conditioning the flow network.

Curriculum Training Schedule. We employ curriculum learning to stabilize training. For horizon curriculum, we start with short prediction horizons and gradually extend: $T_f^{(\text{epoch})} = \min(T_f^{\max}, T_f^{\min} + \text{epoch} \cdot \Delta T)$. For anchor conditioning, we progress through three phases: (1) condition on ground-truth anchors, (2) mix ground-truth and sampled anchors with decreasing ground-truth probability, (3) condition on sampled anchors only.

A.5 Inference Procedure

At inference, we generate N_{samples} diverse trajectory candidates as described in Algorithm 1. Diversity arises from three sources: MDN mixture component sampling, Gaussian noise in anchor sampling, and prior noise in flow integration. For deterministic inference, we use the mixture component with highest weight, the mean of the selected Gaussian, and zero prior noise.

A.6 HYPERPARAMETERS AND IMPLEMENTATION DETAILS

Table 3 summarizes the key hyperparameters.

Table 3: Hyperparameters for the hierarchical base policy generator.

Hyperparameter	**Symbol**	**Value**
Architecture		
Model dimension	d	256
Number of attention heads	H	8
Number of transformer layers (flow network)	L	6
Ego history length	T_h	20 (2s at 10Hz)
Prediction horizon	T_f	80 (8s at 10Hz)
Number of anchors	K	5
MDN mixture components	M	6
Maximum agents	$N_{\max}$	32
Flow Matching		
Integration steps (inference)	N_{steps}	20
Integration steps (training consistency)	$N_{\text{steps}}^{\text{train}}$	10
Guidance scale	ω	2.0
Anchor guidance step size	$\lambda_{\text{anchor}}^{\text{guide}}$	0.1
Unconditional dropout (anchor)	$p_{\text{uncond}}^{\text{anchor}}$	0.1
Position normalization scale	s_{pos}	50 m
Velocity normalization scale	s_{vel}	20 m/s
Training		
EMA momentum	α	0.999
Stop distance threshold	δ_{stop}	2.0 m
Stop velocity threshold	v_{thresh}	0.5 m/s
Safe distance	d_{safe}	3.0 m
Consistency computation frequency	N_{consist}	100 steps
Loss Weights		
Reconstruction loss weight	λ_{recon}	0.1
Diversity loss weight	λ_{div}	0.01
Consistency loss weight	λ_{consist}	1.0
Smoothness loss weight	λ_{smooth}	0.1
Velocity regularization weight	λ_{vel}	0.01
Anchor loss weight	λ_{anchor}	1.0
Policy loss weight	λ_{policy}	0.5
Optimization		
Learning rate	–	1×10^{-4}
Batch size	–	64
Optimizer	–	AdamW
Weight decay	–	0.01
Gradient clip norm	–	1.0

We use sinusoidal positional encodings: $\text{PE}(t, 2i) = \sin(t/10000^{2i/d})$ and $\text{PE}(t, 2i+1) = \cos(t/10000^{2i/d})$. All spatial coordinates are normalized to ego-centric frame and scaled by normalization constants. For variable-length sequences, we apply attention masking to prevent attending to padded positions. For MDN training stability, we clamp log-variance to $[-5, 5]$, use log-sum-exp with max subtraction for NLL computation, and add $\epsilon = 10^{-6}$ to variance terms. Gradient norms are clipped to 1.0 throughout training.

A.7 FUTURE WORK: CONTINUOUS POLICY MODULE

While the MDN-based anchor predictor provides multi-modality through mixture sampling, one can further enrich the space of behaviors by introducing a continuous policy module. Instead of relying

solely on discrete mixture components, this module outputs a continuous latent code $z_{\text{policy}} \in \mathbb{R}^{d_z}$ that conditions the trajectory generator:

$$z_{\text{policy}} = \text{MLP}_{\text{policy}}(f_{\text{up}}, \text{Enc}_{\mathcal{W}}(\mathcal{W})) \tag{13}$$

The latent code is concatenated with the anchor and context embeddings when conditioning the flow network. During training, the policy module can be regularized with a KL divergence term toward a standard Gaussian prior, encouraging a smooth and well-structured latent space. At inference, sampling different latent codes produces trajectories with varied characteristics like aggressiveness, comfort, or caution, that are not easily captured by discrete mixture sampling alone. This continuous formulation allows the model to represent a richer spectrum of driving behaviors while remaining compatible with the downstream MPC search, which can explore the latent space to find trajectories that maximize the learned reward. However, while adding a continuous policy module shows improvement in planning capabilities over edge cases, the module does significantly increase training complexity and requires dataset-specific tuning. The results in this work do not include the continuous policy module, but we leave this here as a foundation for future work in this direction.

B Test-Time Planning via Learning to Search

This section provides additional details on the learning-to-search framework summarized in the main text.

B.1 Latent World Model

We adopt the Recurrent State-Space Model (RSSM) architecture from Dreamer (Hafner et al., 2022). The world model consists of three components:

Encoder. The encoder $\text{enc} : \mathcal{Z} \times \mathcal{O} \times \mathcal{A} \to \mathcal{Z}$ maps observations to latent states. Given observation o_t, previous latent state z_{t-1}, and previous action a_{t-1}:

$$z_t = \text{enc}(z_{t-1}, a_{t-1}, o_t) \tag{14}$$

The encoder processes camera images and vehicle state through separate branches before fusing them into a compact latent representation.

Dynamics Model. The dynamics model $f : \mathcal{Z} \times \mathcal{A} \to \Delta(\mathcal{Z})$ predicts the distribution over next latent states given current state and action:

$$\hat{z}_{t+1} \sim f(z_t, a_t) \tag{15}$$

This enables multi-step rollouts in latent space without requiring expensive observation reconstruction at each step.

Decoder. The decoder $\text{dec} : \mathcal{Z} \to \mathcal{O}$ reconstructs observations from latent states and is used only during training to provide a reconstruction loss that shapes the latent space.

The world model is trained on hybrid data from both $\mathcal{D}_{\text{expert}}$ and $\mathcal{D}_{\text{policy}}$ using a combination of reconstruction loss, dynamics prediction loss, and KL regularization:

$$\mathcal{L}_{\text{WM}} = \mathbb{E}\left[\|\text{dec}(z_t) - o_t\|^2 + \text{KL}(z_t \| \hat{z}_t)\right] \tag{16}$$

Training on hybrid data ensures the world model is accurate on both expert states and states visited by the base policy, which is critical for planning recovery trajectories.

B.2 Reward Model

The reward model $\text{RM} : \mathcal{Z} \to \mathbb{R}$ scores latent states by how expert-like they are. We train it as a discriminator using a moment-matching objective:

$$\mathcal{L}_{\text{RM}} = \mathbb{E}_{(z,a,o)\sim\mathcal{D}_{\text{policy}}}[\text{RM}(\text{enc}(z,a,o))] - \mathbb{E}_{(z,a,o)\sim\mathcal{D}_{\text{expert}}}[\text{RM}(\text{enc}(z,a,o))] \tag{17}$$

Minimizing this objective encourages the reward model to assign high scores to expert-like states and low scores to states characteristic of policy failures. We add a gradient penalty (Gulrajani et al., 2017) to enforce Lipschitz continuity and stabilize training:

$$\mathcal{L}_{\text{GP}} = \lambda_{\text{GP}} \mathbb{E}_{\hat{z}\sim p(\hat{z})}\left[(\|\nabla_{\hat{z}} \text{RM}(\hat{z})\|_2 - 1)^2\right] \tag{18}$$

where $\hat{z}$ is sampled along straight lines between expert and policy latent states. The reward model is updated online throughout training to ensure it can detect failures of the current policy iteration.

B.3 CRITIC

The critic $V : \mathcal{Z} \to \mathbb{R}$ estimates long-horizon returns beyond the planning window k. We train it to predict bootstrapped λ-returns:

$$v_t^\lambda = \text{RM}(z_t) + \gamma \left((1-\lambda) V(z_t) + \lambda v_{t+1}^\lambda \right), \quad v_{t+k}^\lambda = V(z_{t+k}) \tag{19}$$

where γ is the discount factor and λ controls the bias-variance tradeoff. The critic is trained to minimize:

$$\mathcal{L}_\text{V} = \mathbb{E}\left[\| V(z_t) - v_t^\lambda \|^2 \right] \tag{20}$$

We train an ensemble of 5 critic networks to estimate uncertainty. At inference, we compute the mean of 2 randomly sampled critics and subtract a penalty proportional to the ensemble standard deviation, encouraging conservative planning under uncertainty.

B.4 MODEL PREDICTIVE CONTROL

At inference, we solve the following optimization problem to find residual corrections:

$$\Delta_{t:t+k}^\star = \arg\max_{\Delta_{t:t+k}} \sum_{h=0}^{k-1} \gamma^h \text{RM}(z_{t+h}) + \gamma^k V(z_{t+k}) \tag{21}$$

where z_{t+h} is obtained by rolling out corrected actions $a_{t:t+k}^\text{base} + \Delta_{t:t+k}$ in the learned world model.

We solve this using a MPPI, a derivative algorithm of MPC (Kouvaritakis & Cannon, 2016). MPPI maintains a Gaussian distribution over residuals $\mathcal{N}(\mu, \sigma)$ and iteratively refines it:

1. Sample N candidate residuals: $\Delta_{t:t+k}^n \sim \mathcal{N}(\mu^j, \sigma^j)$ for $n = 1, \ldots, N$
2. Roll out each candidate in the world model: $z_{t:t+k}^n \sim \text{WM}(o_t, a_{t:t+k}^\text{base} + \Delta_{t:t+k}^n)$
3. Compute trajectory scores: $\hat{Q}^n = \sum_{h=0}^{k-1} \gamma^h \text{RM}(z_{t+h}^n) + \gamma^k V(z_{t+k}^n)$
4. Update distribution via softmax-weighted average:

$$w^n = \frac{\exp(\hat{Q}^n / \tau)}{\sum_{m=1}^N \exp(\hat{Q}^m / \tau)}, \quad \mu^{j+1} = \sum_{n=1}^N w^n \Delta_{t:t+k}^n \tag{22}$$

After J iterations, we execute the first corrected action $a_t^\star = a_t^\text{base} + \mu_t^J$ and replan at the next timestep in receding-horizon fashion.

B.5 TRAINING PHASES

Following Jain et al. (2025), we also schedule our training in three phases:

Phase 1: We collect rollouts using the base hierarchical policy to populate the replay buffer $\mathcal{D}_\text{policy}$. We then co-train the world model, reward model, and critic on hybrid data from $\mathcal{D}_\text{expert}$ and $\mathcal{D}_\text{policy}$. This phase uses approximately 20% of the total interaction budget and ensures the learned components are accurate on the base policy's state distribution before planning begins.

Phase 2: We deploy the full planning stack and collect rollouts with test-time planning active. We continue hybrid training of all components on the growing buffer, allowing the world model to improve on states visited during recovery and the reward model to distinguish the improved policy's behavior from expert behavior.

Phase 3: We periodically distill the planner's outputs back into the base policy. Specifically, we relabel observations in $\mathcal{D}_\text{policy}$ with actions generated by the full planning stack and fine-tune the hierarchical policy via behavioral cloning:

$$\pi^\text{base} \leftarrow \arg\min_\pi \mathcal{L}_\text{BC}(\{(o_t, a_t^\star) \mid o_t \in \mathcal{D}_\text{policy}\}, \pi) \tag{23}$$

This recycles test-time compute into an improved base policy, reducing the need for planning when similar situations arise in the future. Phases 2 and 3 repeat iteratively throughout training.

Table 4: Learning-to-search hyperparameters.

Hyperparameter	**Symbol**	**Value**
Planning horizon	k	8
Discount factor	γ	0.99
λ-return coefficient	–	0.95
MPC samples	N	256
MPC iterations	J	6
MPC temperature	τ	1.0
Critic ensemble size	–	5
Gradient penalty coefficient	λ_{GP}	10.0
Warm-start budget fraction	–	0.2
Expert iteration frequency	–	Every 5 epochs

Table 5: Performance comparison across Val14, Test14-hard, and Test14 benchmarks. Best results are in **bold**, second best are underlined.

Category	**Method**	**Val14**		**Test14-hard**		**Test14**	
		NR	R	NR	R	NR	R
Expert	Log-Replay	93.53	80.32	85.96	68.80	94.03	75.86
Learning	PLUTO w/o refine	88.89	78.11	70.03	59.74	89.90	78.62
	Diffusion Planner	89.87	82.80	75.99	69.22	89.19	82.93
	Flow Planner	90.43	83.31	76.47	70.42	89.88	82.93
Hybrid	PDM-Hybrid	92.77	92.11	65.99	76.07	90.10	91.28
	PLUTO w/refine	89.87	82.80	75.99	69.22	89.19	82.93
	Diff. Planner w/refine	94.26	92.90	78.87	82.00	94.80	91.75
	Flow Planner w/refine	94.31	92.38	78.64	80.25	94.79	92.40
Hierarchical	HiDrive	93.62	**93.15**	81.41	83.18	93.71	92.31
	HiDrive-Dynamic	93.51	**93.15**	81.88	83.17	93.72	**92.56**
	HiDrive-QCNet	93.42	92.84	**81.91**	82.69	93.37	91.80
	HiDrive-Uniform	92.60	92.80	79.98	82.56	92.08	91.62
Ours	Ours (Diffusion)	89.93	83.47	77.63	71.38	89.26	82.81
	Ours (FM)	91.29	84.93	78.19	72.62	90.13	84.39
	Ours (FM) w/L2S	**95.81**	90.44	81.89	**85.47**	**95.61**	90.16

B.6 Hyperparameters

Table 4 summarizes the key hyperparameters used in our experiments.

C Results

Table 5 details performance across the Val14 and Test14 benchmarks. Our complete method, **Ours (FM) w/L2S**, sets a new state-of-the-art on the most challenging Test14-hard benchmark (NR: 81.89, R: **85.47**) and achieves top scores on Val14 NR (**95.81**). We observe that HiDrive, a recent competitive baseline, also performs well; since both approaches leverage hierarchical structures, this shared success strongly validates that decoupling high-level behavior from low-level control is a key design choice for robust planning. Regarding the generative backbone, **Ours (FM)**, the model with flow matching, consistently outperforms **Ours (Diffusion)** (the model with denoising diffusion instead of FM) across all metrics (e.g., +5.88 on Val14 NR). We hypothesize that Flow Matching is superior here because it enforces straighter probability flow trajectories during training, simplifying the learning objective and resulting in higher-fidelity generation than standard diffusion. Finally, adding the search module to the FM base yields a massive performance boost (e.g., +12.8 points on Test14-

hard Reactive), confirming that test-time optimization is essential to recover from base policy errors in complex scenarios.

Table 6: Ablation study on the source of candidate trajectories for the world model planner. Performance is measured on the Test14-NR benchmark.

Method	Performance (Test14-NR)
Ours	**95.62**
Diffusion Policy	79.15
Random Policy	32.16

How much does the WM depend on the base policy? To understand the relationship between our world model planner and the underlying base policy, we evaluated the planner's performance when initialized with candidate trajectories from different sources (Table 6). While our full method achieves a score of 95.61, replacing the base policy with a standard Diffusion Policy (Ren et al., 2024) drops performance to 79.15, and using a Random Policy results in a collapse to 32.16. This indicates that our world model is not a global planner capable of solving the driving task from scratch. Instead, it functions as a highly effective *local refinement module*. It relies on the base policy to provide a "warm start", a distribution of trajectories reasonably close to the expert solution, which the planner then optimizes. Without a high-quality structural prior from the base policy, the search space is too vast for the world model to navigate effectively in real-time.

Ablating Design Choices We also ablate certain architectural and training design choices for both our hierarchical base policy generator and the L2S module. Table 7 shows performative comparisons of CLS Score on nuPlan Val14-NR. We see that training that ablating every design choice results in a drop in overall performance.

Table 7: Ablation study on the source of candidate trajectories for the world model planner. Performance is measured on the Test14-NR benchmark.

Method	Performance (Val14-NR)
Only Hierarchical Base Policy	**91.29**
w/o Anchor Consistency	82.75
w/o Curriculum Training Schedule	63.40
Base Policy + L2S	**95.81**
w/o Critic	84.19
w/o Buffer Initialization	86.02
w/o Joint Training	70.37

Qualitative Evaluation We provide qualitative evidence of the ego vehicle entering "unsafe" states when the base policy fails to adapt to dynamic environmental changes. Our reward model identifies these failures by assigning low scores to the base trajectories (blue). In response, the L2S module leverages the world model to simulate potential outcomes and generates a residual plan to alter the original base policy during inference (red), ensuring a safe and optimal path is chosen during execution. Figures 1b, 1c, 2, and 3 show different ways the residual plan updates the base policy. This also helps interpreting the model, as we are able to observe different trajectories and the effects of residuals generated by the world model.

D DISCUSSION

Test-Time Planning as a Complementary Paradigm. Our work suggests that test-time planning offers a complementary approach to parameter-based adaptation methods like test-time training and test-time adaptation. Where those methods modify the model to better fit the test distribution, we instead modify the decisions while keeping the model fixed. This distinction has practical implications for autonomous driving: planning corrections are local to the current decision and do not

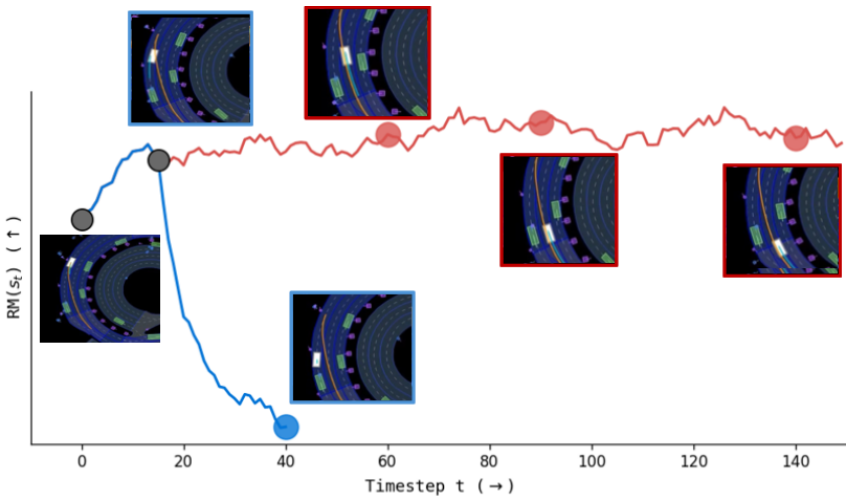

Figure 2: The original trajectory (blue) causes the ego vehicle to stop behind a stationary vehicle on a turn. Similar behaviors have also been observed by Zheng et al. (2025) where the ego vehicle has difficulty executing lateral motion. The residual plan (red) forces the ego vehicle to overcome this and change lanes beforehand in order to avoid getting stuck behind stationary vehicles. The orange line is the ground-truth trajectory the vehicle actually took in the simulation with the LQR controller. Our updated plan also guides the ego vehicle to follow the same trajectory.

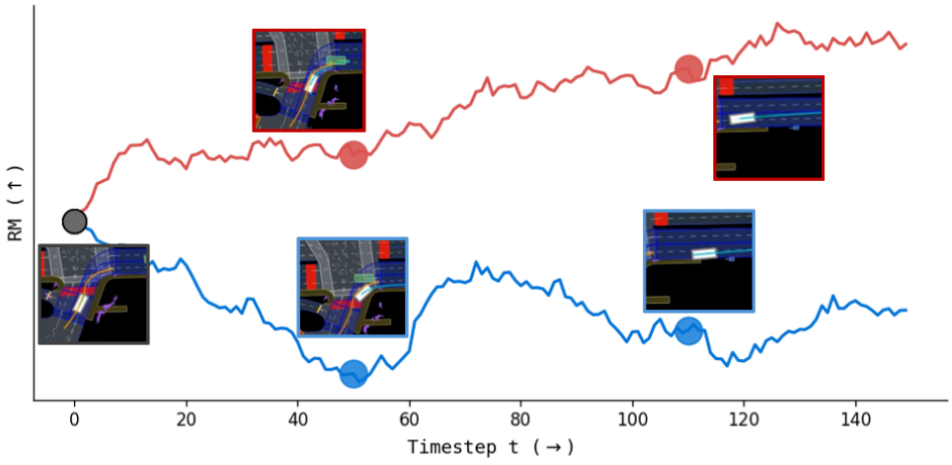

Figure 3: We see examples of two different trajectories rolled out in the WM and scored against the RM. The original trajectory (blue) results in the ego vehicle colliding with an incoming vehicle at ($t = 50$) and later passing a cyclist with an unsafe distance (at $t = 110$). The new trajectory (red) avoids this by initializing the ego vehicle with a slower speed from the very beginning, resulting in an already higher reward, making it easier to stop and wait for incoming traffic. The new trajectory also guides the ego vehicle a safe distance away from the cyclist.

risk destabilizing learned behavior elsewhere in the state space. Furthermore, candidate trajectories can be inspected and filtered against safety constraints before execution, providing a layer of interpretability and control that implicit parameter updates lack.

Limitations. Our approach inherits limitations from its constituent components. The quality of test-time planning depends on the accuracy of the learned world model, which may degrade in scenarios far from both expert and base policy experience. While our hybrid training paradigm somewhat mitigates this issue, truly out-of-distribution scenarios—such as novel road geometries or unprecedented traffic patterns—may exceed the world model's generalization capacity. Additionally, MPC planning introduces computational overhead at inference time. Although we found this overhead acceptable for the planning frequencies typical in autonomous driving, real-time constraints on embedded hardware may require more efficient search algorithms or distillation of the planning process into the base policy. Addressing these limitations through improved world model generalization and reduced inference cost remains an important direction for future work.