ONLINE POLICY OPTIMIZATION FOR AUTONOMOUS DRIVING WITH A HIERARCHICAL WORLD MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Policies obtained via imitation learning are known to struggle with out-of-distribution cases, i.e., when the environment or the agent deviates from the expert distribution at test time. Test-time planning offers a potential solution by learning a more generic world model and reward model from demonstrations and searching for corrective actions at inference. However, this approach has not been verified in driving setups due to the computational cost of searching over long planning horizons. We introduce HiPlan, a hierarchical planning framework that mirrors the temporal structure of the hierarchical policy it supports, with an upper level operating at the subgoal timescale and a lower level at full resolution. This enables two-stage search, which reduces search dimensionality by an order of magnitude, and whose corrections are distilled back into the policy. We find that the two levels develop qualitatively different representations, with the upper model encoding strategic features and the lower encoding tactical features, a double dissociation confirmed by linear probing and horizon-scaling analysis. HiPlan yields state-of-the-art planning results on nuPlan and CARLA. Our experiments show that the hierarchy enables more efficient search, that representations factorize by timescale, and that structural alignment between the policy and the world model is necessary for both.

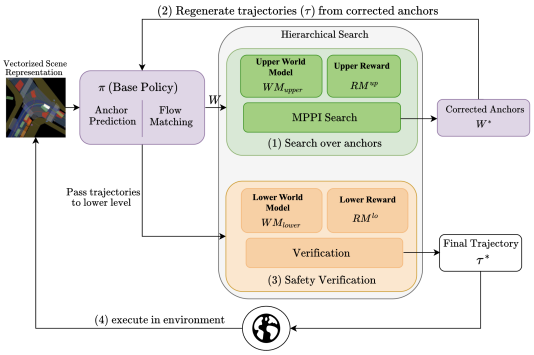
1 INTRODUCTION

Motion planning for autonomous driving is commonly framed as trajectory generation: given a scene, produce a sequence of ego states that the controller executes. The dominant approach is imitation learning, where a policy is trained on expert demonstrations to generate plausible trajectories (39; 30; 3; 26). Imitation-based planners perform well when test conditions match training, but provide no mechanism for recovery once the ego-vehicle deviates from the expert state distribution, which is a central source of brittleness in deployed systems (4; 27; 7).

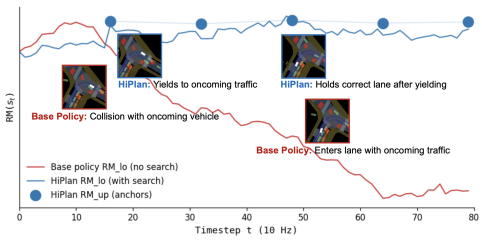
A promising extension is to augment the resulting base policy with test-time planning. Following (14), the idea is to learn two auxiliary components alongside the base policy: a world model that predicts future states given candidate actions in a learned latent space, and a reward model that scores states by how expert-like they are. At inference, a model-predictive path integral (MPPI) procedure searches for residual corrections to the base policy's trajectory by rolling out candidates in the world model and selecting the highest scoring one. We use "planning" in this sense throughout the paper, as test-time trajectory search over a learned latent model, rather than end-to-end policy learning or goal-conditioned planning studied in zero-shot world model literature (35; 40). The key advantage over pure imitation is qualitative. A feed-forward policy can only reproduce learned behavior, but a world model can *evaluate* candidate actions by simulating their consequences, providing a recovery mechanism in states the policy has never encountered.

This framework has been demonstrated for short-horizon robotic manipulation tasks, but not for autonomous driving. Driving requires planning over horizons of several seconds at 10Hz (tens of sequential timesteps), where two problems emerge. First, the search dimensionality: MPC over dense action sequences explores a high-dimensional space where uniform sampling becomes grossly inefficient. Second, prediction compounding: long rollouts through a learned dynamics model accumulate error, degrading the signal that search optimizes against. Both problems become more

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) *HiPlan* inference architecture and search stages.



(b) Reward model scores on an unprotected left turn. Base policy (red) collides with oncoming traffic; *HiPlan* (blue) yields and proceeds safely.

Figure 1: HiPlan augments a hierarchical base policy with two-stage test-time search. An upper world model searches for corrected anchor waypoints via MPPI; a lower world model verifies dense trajectory safety. The two levels are coupled through cross-level conditioning and state reset at anchor boundaries. (b) shows a scenario where anchor-level search recovers from a base policy failure.

severe with increasing temporal horizons. Hence, adopting test-time planning to long horizons must exploit the temporal structure of the planning problem itself.

Driving decisions decompose naturally across timescales. Decisions like which lane to occupy, whether to yield or proceed, or which gap to accept, unfold over several seconds. Conversely, decisions like speed modulation, jerk control, and pausing distance operate at a sub-second scale. Hierarchical imitation policies exploit this decomposition by separating subgoal prediction from dense trajectory generation, achieving strong results on driving benchmarks (17; 23). The world model used for test-time planning should exploit the same structure, for the same reasons: long-horizon reasoning is at a subgoal scale, where the search space is small and predictions need only be coarse; fine-grained reasoning is at a trajectory scale, where precision matters but horizons are short.

We introduce *HiPlan*, a hierarchical framework for test-time planning in autonomous driving. The temporal structure is fixed by design – anchor waypoints are placed at uniformly-spaced time indices, and the upper level operates at the anchor timescale while the lower level operates at full resolution within each anchor segment. This decomposition is imposed as a prior rather than learned, matching the natural separation between strategic and execution-level driving decisions. Within this fixed structure, the world models at both levels, the reward models, and the residual search corrections are all learned. The lower level is reset at each anchor boundary to preserve hierarchy, enabling a two-stage search: a coarse search over anchor perturbations in a low-dimensional subgoal space, followed by dense trajectory generation and safety verification at full resolution but short horizon — an order of magnitude reduction in search dimensionality.

The more consequential finding is what the hierarchy learns. We find that the two levels develop qualitatively different representations. The upper model linearly exposes high-level features like lane identity, and maneuver class. On the other hand, the lower model linearly exposes execution-level features like instantaneous speed, jerk, agent proximity. Linear probes reveal a double dissociation between levels, and a horizon-scaling analysis shows that the upper model accumulates prediction error more slowly than the lower model over long horizons. This temporal factorization makes two-stage search effective: each level is trained to encode, in its latent state, the decisions natural to its timescale. We validate this causally through ablations showing that neither hierarchical world model nor hierarchical search alone recovers the performance of the aligned system.

Contributions. (1) We present the first hierarchical test-time planning framework for autonomous driving, combining a hierarchical world model with two-stage MPPI search to address the dimensionality and compounding problems that make traditional planning intractable at driving horizons, and distilling the resulting corrections back into the base policy. (2) We show that hierarchical world models develop temporally factorized representations, validated through double dissociation in linear probes and horizon-scaling analysis. (3) We achieve state-of-the-art results on established planning benchmarks, supported by analytical experiments that characterize why hierarchy helps.

2 Related Works

Imitation learning for autonomous driving Behavior cloning (BC) learns driving policies by maximizing expert action likelihood, with more recent works using diffusion models (39; 23) and flow matching (30) achieving strong benchmark performance. However, BC provides no recovery mechanism under distribution shift (27). Approaches that augment BC with rule-based refinement (3; 4), reinforcement learning (13; 36), or test-time parameter adaptation (15) either require extensive online interaction, or risk destabilizing learned behavior. Our approach searches for corrective actions at inference using learned world and reward models and distills them back into the base policy.

Hierarchical planning with learned world models Planning benefits from reasoning at multiple timescales (29; 25; 20; 31; 32), and recent driving policies exploit this by separating subgoal prediction from trajectory generation (17; 23). However, the hierarchy in these approaches is confined to the policy, leaving the world models used for planning, where they exist at all, flat. Test-time planning with learned world models has been studied separately, with (14) introducing learning-to-search for short-horizon manipulation with a flat world model, and (35) composing pretrained models at multiple temporal resolutions for goal-conditioned navigation. Neither addresses driving-length horizons with a coupled hierarchical world model trained jointly with a hierarchical policy. We do, and show that the structural alignment between policy and world model hierarchy is necessary – hierarchy in either alone underperforms the aligned configuration (Section 4.5).

We provide a more detailed comparison of these approaches, including their relationship to our method, and a summary across key design axes, in Appendix A.

3 Latent Planning with Hierarchical World Models

3.1 Problem Setup

HiPlan extends a hierarchical imitation policy with hierarchical world and reward models, and a two-stage search procedure that shares the policy's temporal abstraction. Figure 1a shows the complete architecture and data flow. At inference, the base policy generates a candidate trajectory decomposed into sparse subgoals and a dense execution; a learned world model rolls out alternative subgoal sequences in a coarse latent space; a reward model scores them; the best is realized as a dense trajectory and verified for safety before execution. Training proceeds in four phases that progressively integrate the base policy with search components and distill corrections back into the policy.

We consider the standard motion planning setup: given scene context $\mathcal{S}_t = \{E_t, A_t, M\}$ comprising ego history, neighboring agent trajectories, and a vectorized scene map, produce a dense trajectory $\tau = \{e_{t+1}, ..., e_{t+T_f}\}$ over a planning horizon of T_f steps at 10Hz. We describe the base policy, world model, reward models, and search in turn, with training details in the next sections, and all hyperparameters in the Appendix.

3.2 Hierarchical Base Policy

The base policy decomposes trajectory generation into sparse subgoal prediction followed by dense trajectory synthesis, producing both abstractions that the world model will share. A shared encoder processes $\mathcal{S}_t$ into a scene representation f_{up}. An anchor predictor maps f_{up} to K sparse probabilistic waypoints $W = \{w_1, ..., w_K\}$ at uniformly spaced time indices. A conditional flow matching generator (22) maps (W, f_{up}) to a dense trajectory τ by transporting samples from a Gaussian prior along a learned vector field, with classifier-free guidance ensuring that generated trajectories respect the predicted anchors.

The anchor predictor is instantiated as a Mixture Density Network (1) with M Gaussian components per anchor, capturing multi-modal intent such as lane-change versus lane-keep decisions. The flow matching generator is a Transformer conditioned on $[f_{up}; \text{Enc}_W(W)]$ and integrated at inference with fixed-step Euler scheme. Training proceeds in three curriculum stages (anchor predictor alone, flow generator with ground-truth anchors, joint fine-tuning with sampled anchors), following the standard recipe for hierarchical diffusion-style planners (17; 23). Full loss formulations and training schedules are in Appendix B.

3.3 HIERARCHICAL WORLD MODEL

The world model mirrors the two-level structure of the policy: an upper-level model predicts the transitions at the subgoal timescale, with waypoints as anchors, and a lower-level model predicts transitions at full resolution, conditioned on upper-level context. Both levels are Recurrent State-Space Models (9) that maintain a deterministic recurrent state h and a stochastic latent state z, and differ in their temporal resolution and action space. We provide further details in Appendix C.

Upper-level world model At the anchor timescale $\Delta_{\text{anchor}} = \lfloor T_f/K \rfloor$, the upper-level RSSM transitions as

$$h_k^{\text{up}} = \text{GRU}(h_{k-1}^{\text{up}}, [z_{k-1}^{\text{up}}, w_{k-1}]), \quad z_{k-1}^{\text{up}} \sim p(z \mid h_k^{\text{up}}) \tag{1}$$

where the anchor waypoints w_{k-1} serve as the "actions" driving the coarse dynamics. This is a deliberate design choice: the action space of the upper world model matches the anchor output space of the base policy, so the two interact at the same level of abstraction during search. The stochastic latent z^{up} is discrete, with 32 categorical variables of 32 classes each, following (10); the categorical structure supports multi-modal predictions over qualitatively different anchor outcomes (e.g., overtake vs. yield). The posterior is trained to reconstruct a coarse scene summary at each anchor step, regularized by a KL term toward the prior.

Lower-level world model At full resolution, the lower-level RSSM operates on primitive actions a_t, and is conditioned on the upper-level state corresponding to the current anchor segment:

$$h_t^{lo} = \text{GRU}(h_{t-1}^{lo}, [z_{t-1}^{lo}, a_t, z_{k(t)}^{up}]), \quad z_t^{lo} \sim p(z|h_t^{lo}) \tag{2}$$

At each anchor boundary t_k, the lower recurrent state is reinitialized from the upper state via $h_{t_k}^{lo} = \text{MLP}_{\text{init}}(s_k^{up})$. This reset is what enforces the hierarchy. Without it, the lower model would maintain its own long-term memory and the upper level would become redundant. By resetting, the upper level is solely responsible for carrying information across anchor segments while the lower level handles dense dynamics within each segment.

Joint Training The two levels are first pretrained separately, with the upper level on expert trajectories subsampled at anchor indices and the lower level on full-resolution data conditioned on ground-truth upper latents obtained by encoding the true observations at anchor timesteps. They are then jointly fine-tuned where the upper level imagines the latent sequences, the lower level reconstructs fill observations on those imagined states, and the reconstruction loss backpropagates through both RSSM levels. The shared encoder is held fixed via its EMA copy during world model training, ensuring the policy's scene representation remains stable while the world model adapts its latent dynamics to be consistent with that representation. This joint signal forces the upper latent space to encode information the lower level depends on. Both phases use hybrid data $\mathcal{D}_{expert} \cup \mathcal{D}_{policy}$ so the model is accurate on both expert states and states visited by the current policy.

3.4 REWARD MODELS AND CRITIC

The reward models evaluate expert-likeliness at both abstraction levels, supporting cheap coarse search and fine-grained safety verification through single pair of discriminators. The upper reward $\text{RM}_{up} : \mathcal{Z}^{up} \to \mathbb{R}$ scores subgoal sequences, and the lower reward $\text{RM}_{lo} : \mathcal{Z}^{lo} \to \mathbb{R}$ scores the dense trajectory states. Both are trained as Wasserstein discriminators with gradient penalty (6).

$$\mathcal{L}_{RM} = \mathbb{E}_{z\sim\mathcal{D}_{policy}}[\text{RM}(z)] - \mathbb{E}_{z\sim\mathcal{D}_{expert}}[\text{RM}(z)] + \lambda_{GP} \cdot \text{GP} \tag{3}$$

and updated online throughout training so they remain discriminative as the policy improves the gap between expert and learned behavior narrows.

The two reward models play complementary roles in search. RM_{up} is the search driver. It enables cheaper evaluation on K coarse states and sensitivity to high-level intent errors. RM_{lo} is the safety verifier. It is more expensive but catches fine-grained failures that a coarse score cannot detect.

Critic An upper-level critic V_{up} estimates returns beyond the K-step planning window, trained on λ-returns computed at the anchor timescale with effective per-anchor discount $\gamma^{\Delta_{anchor}}$. We use an ensemble of 5 networks and penalize by standard deviation for conservative planning under uncertainty. No lower-level critic is needed as the lower model operates over the full planning horizon, leaving no tail to estimate.

3.5 Two-Stage Search

Test-time search exploits the hierarchy by first optimizing subgoals in a low-dimensional space, then generating and verifying the resulting dense trajectory. The procedure consists of four stages (Algorithm 1): coarse MPPI over anchor perturbations in the upper world model, dense trajectory generation via flow matching from the corrected anchors, safety verification in the lower world model, and final selection by combined score. We provide further details in Appendix D.

Stage 1: Upper-level MPPI Given base anchors W^{base} from the policy, we search for residual corrections ΔW by sampling N_{up} candidates, rolling them out through the upper world model, and scoring trajectories as $Q = \sum_k \gamma^k \text{RM}_{up}(z_k^{up}) + \gamma^k V_{up}(z_k^{up})$. The proposal distribution is refined over J_{up} iterations via softmax-weighted averaging. The search operates in a $2K$-dimensional anchor space rather than the $2T_f$-dimensional dense action space. For the nuPlan setup ($K = 5, T_f = 80$), this is a $16\times$ reduction in search dimensionality.

Stages 2-3: Generation and Verification The corrected anchors $W^* = W^{base} + \Delta W^*$ are passed through the flow matching generator to produce N_τ dense trajectory candidates. This step is deterministic given W^* and the prior noise, so no search is required. Each candidate is rolled out in the lower world model, with the lower state reinitialized from upper states at anchor boundaries. RM_{lo} scores each dense state and trajectories whose scores fall below a safety threshold (calibrated at the 10th percentile of RM_{lo} on expert data) are rejected.

Stage 4: Selection Among the surviving candidates, we select the trajectory maximizing $R_{total} = \sum_k \text{RM}_{up}(z_k^{up}) + \alpha \sum_t \gamma^t \text{RM}_{lo}(z_t^{lo})$, balancing intent quality against execution safety. The first action is executed and the procedure replans at the next timestep. The residual formulation is self-regulating when the reward model is informative: if the base policy is already competent, MPPI converges to near-zero corrections and the system executes the base plan directly.

3.6 Training Pipeline

The full system is trained in four phases that progressively integrate the components and recycle search corrections back into the policy. (1) The base policy is trained on expert demonstrations alone via the three-stage curriculum of Section 3.2. (2) The base policy is deployed to collect rollouts $\mathcal{D}_{policy}$, the world models, reward models, and critics are trained on $D_{expert} \cup \mathcal{D}_{policy}$, with upper and lower levels pretrained separately before joint fine-tuning. (3) The full two-stage search is deployed, collecting new rollouts that include recovery data, and all learned components continue training on the growing buffer. (4) Search corrections are distilled into the base policy.

Distillation The two-level structure enables targeted distillation at each abstraction. The anchor predictor is fine-tuned on (f_{up}, W^*) pairs from the search output, and the flow generator is fine-tuned on (W^*, f_{up}, τ^*) triplets. Only cases with non-trivial corrections ($||\Delta W|| > \epsilon$) are distilled, so the policy is not trained on redundant signal where the base plan was already good. In Phases 2 and 3, the base policy absorbs the easier corrections, the reward models see harder negatives (since the policy-expert gap narrows), and the world model deals with the harder cases that still require online search. The system converges when marginal improvement from distillation falls below a threshold.

4 Experiments

We evaluate HiPlan on traditional planning benchmarks used in autonomous driving. We show results on nuPlan (18) and CARLA Longest6 v2 (5). On nuPlan, we report closed-loop score (CLS) on Val14 (1118 scenarios), Test14 (200+ randomly selected scenarios), and Test14-hard (272 scenarios filtered

Algorithm 1 Inference Procedure

Require: observation o_t, base anchors $\mathcal{W}_{\text{base}}$ from policy
Ensure: action a_t to execute

$z_0^{\text{up}} \leftarrow \textsc{EncodeUpper}(o_t)$
$\mu, \sigma \leftarrow \mathbf{0}$, σ_{init}
for $j = 1$ **to** J_{up} **do** ▷ Stage 1: Upper MPPI
  Sample $\{\Delta\mathcal{W}^n \sim \mathcal{N}(\mu, \sigma)\}$ for $n = 1, \ldots, N_{\text{up}}$
  for each n **do**
    Roll out $z_{1:K}^{\text{up},n}$ in upper WM with $\mathcal{W}_{\text{base}} + \Delta\mathcal{W}^n$
  end for
  Compute $Q^n = \sum_k \gamma^k \cdot \text{RM}_{\text{up}}(z_k^{\text{up},n}) + \gamma^K V_{\text{up}}(z_K^{\text{up},n})$
  Update μ, σ via softmax-weighted average over $\{(\Delta\mathcal{W}^n, Q^n)\}$
end for
$\mathcal{W}^* \leftarrow \mathcal{W}_{\text{base}} + \mu$
$\{\tau^n\} \leftarrow \textsc{FlowMatching}(\mathcal{W}^*,\ f_{\text{up}})$ ▷ Stage 2: Generation
for each τ^n **do** ▷ Stage 3: Verification
  Roll out $z_{1:T_f}^{\text{lo},n}$ in lower WM with τ^n, conditioned on $z^{\text{up}*}$
  if $\min_t \text{RM}_{\text{lo}}(z_t^{\text{lo},n}) < \delta$ **then**
    Reject τ^n
  end if
end for
$\tau^* \leftarrow \arg\max_{\tau^n \in \text{surviving}} R_{\text{total}}(\tau^n)$ ▷ Stage 4: Selection
return $\tau^*[1]$ *// first action*

as worst-performing for rule-based planners), in both non-reactive (NR) and reactive (R) settings. On CARLA, we report Driving Score (DS) and its constituent metrics on 36 routes spanning 5-22 safety-critical events in each. We compare against imitation baselines (Diffusion Planner (39), Flow Planner (30)), hybrid refinement methods (PDM-Hybrid (4), Diffusion Planner with refinement), and hierarchical planners (HiDrive (17)), as well as two internal ablations: our hierarchical base policy alone, and a flat world model paired with our search procedure. For CARLA, we evaluate on a different set of planners (PlanT (26), Roach (36), CaRL (13)) since the code of these planners was compatible with only a particular dataset. We also report additional results and supplemental information in Appendix E.

Our experiments answer five questions. Does HiPlan improve over state-of-the-art driving planners (Section 4.1)? When faced with maneuvers absent from training, does planning with a learned world model recover behavior that simple imitation cannot (Section 4.2)? Does hierarchical search reach strong performance with less compute than flat search (Section 4.3)? Do the two levels of the hierarchical world model encode qualitatively different information (Section 4.4)? And, is hierarchy in the world model alone sufficient, or must the search procedure also be hierarchical (Section 4.5)?

4.1 Main Results

Table 1 reports nuPlan closed-loop scores. The full system achieves the highest scores on Test14-Hard (83.77 NR, 86.90 R), surpassing HiDrive (17) by +2.3 and +3.7 in each split respectively. These are scenarios specially filtered for dense traffic and complex intersections where base policies fail. Table 2 reports CARLA Longest6 v2 scores, where 36 routes span up to 2km with 5-22 safety-critical events each. HiPlan achieves 70 DS, outperforming CaRL (13) (64) and PlanT (26) (62). Further results for both datasets with metric decomposition are provided in Appendix E.1.

Three rows in Table 1 separate the contributions of the base policy, the world model and the hierarchy. The base policy alone, trained on expert demonstrations before any distillation, trails the strongest hierarchical planners and serves as the reference point for the planning components. Adding search with a flat world model gives moderate gains on Test14-hard (+4.7 NR, +2.9 R) and on CARLA (+6 DS), so planning with a learned world model helps even without hierarchy. Replacing the flat world model with the hierarchical one adds a further +11.4 Test14-hard R and +7 DS on CARLA, making the hierarchy the largest single contribution in the system. On CARLA, this gap grows with the

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

Method	Type	Val14		Test14-hard		Test14	
		NR	R	NR	R	NR	R
Log Replay	Human	93.53	80.32	85.96	68.80	94.03	75.86
Flow Planner (30)	IL	90.43	83.31	76.47	70.42	89.88	82.93
Diff. Plan. w/refine (39)	Hybrid	94.26	92.90	78.87	82.00	94.80	91.75
HiDrive (17)	Hier.	93.62	**93.15**	81.41	83.18	93.71	92.31
HiPlan (base policy only)	–	91.29 $\pm$1.9	84.93 $\pm$2.4	77.19 $\pm$2.2	72.62 $\pm$3.8	90.13 $\pm$1.2	84.39 $\pm$2.1
HiPlan (flat WM search)	–	**95.81** $\pm$1.7	90.44 $\pm$2.5	81.89 $\pm$1.9	75.47 $\pm$1.3	**95.61** $\pm$0.9	90.16 $\pm$2.2
HiPlan (hier. WM search)	–	**95.62** $\pm$1.5	**93.09** $\pm$1.8	**83.77** $\pm$2.1	**86.90** $\pm$2.3	95.19 $\pm$1.1	**93.15** $\pm$2.0

planning horizon, from 7 DS at 8 s to 22 DS at 24 s (Appendix E.7). We also provide full component ablations (E.7), correction decomposition analysis (E.8), and distillation convergence results (E.9).

4.2 Recovery Under Distribution Shift

A policy trained via behavioral cloning has no mechanism to evaluate whether its output is reasonable at unfamiliar states. A world model, by contrast, enables search: the system samples candidate corrections, simulates their consequences, and selects the one scoring highest against the learned reward. This distinction should matter most when the agent encounters maneuvers absent from training, so we test it directly.

Table 2: Driving Score (DS) and Route Completion (RC) on CARLA Longest6 v2. **Bold** indicates best, underline indicates second best.

Planner	DS ↑	RC ↑
PlanT (26)	62	**96**
Think2Drive (21)	7	39
Roach (36)	22	77
CaRL (13)	64	82
HiPlan (base policy)	57 $\pm$4	73 $\pm$4
HiPlan (flat WM)	63 $\pm$4	82 $\pm$8
HiPlan (hier. WM)	**70** $\pm$2	90 $\pm$4

Using scenario labels from nuPlan, we progressively remove all scenarios tagged as a given maneuver type from the training set and evaluate on held-out scenarios of that type. We apply this to two structurally different maneuvers – U-turns (a geometric challenge) and unprotected left turns (a reactive, gap-selection challenge) – at 100%, 50%, 25%, and 0% retention, evaluating the base policy, flat WM search, and hierarchical WM search at each level. All three systems are trained on identical expert data, with the held-out maneuver removed from every stage of training, so the differences come from the world model and search, both at inference and through the corrections distilled during training.

Figure 5 shows CLS on held-out U-turns as a function of retention. All three systems degrade with reduced coverage, but at sharply different rates: the base policy collapses at 0% retention, the flat WM degrades gracefully as search discovers partial corrections, and the hierarchical WM degrades slowest as anchor-level search in a low-dimensional subgoal space discovers novel trajectory shapes more efficiently than dense-action search. Collision rates on unprotected lefts (Table 3) confirm the same pattern: at 0% retention, the base policy collides in the majority of scenarios while the hierarchical WM maintains substantially lower rates by discovering the yield decision through search rather than memorized demonstrations.

Table 3: Collision rate (%) ($\downarrow$) on unprotected left turns under distribution shift. Collision rate is computed from the nuPlan no-collision (NC) metric as the fraction of scenarios with $\mathrm{NC} < 1$ (NC = 1: no collision, 0.5: partial fault, 0: full fault).

Data retained	100%	50%	25%	0%
Base Policy	12	38	46	72
Flat WM	10	24	31	45
Hierarchical WM	**9**	**16**	**22**	**30**

4.3 Search Efficiency

We sweep the total MPPI evaluation budget from 64 to 5000 and compare hierarchical search (anchor-level MPPI in the upper world model followed by lower-level verification) against flat search (dense-action MPPI in a matched-capacity flat world model). For each configuration, we optimize samples-to-iterations split on validation data so neither method is handicapped by a poor hyperparameter choice. Figure 2 plots Test14-hard reactive scores against total MPPI evaluations

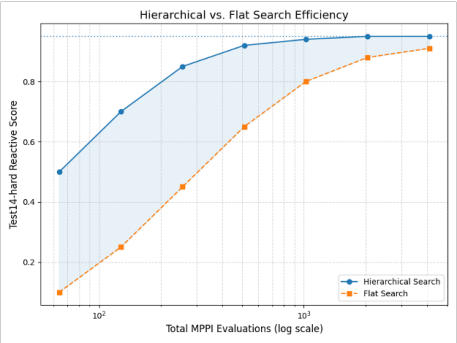


Figure 2: Search efficiency for hierarchical vs. flat MPPI as a function of CLS score. Hierarchical search outperforms flat search at every budget and plateaus by ∼1000 WM calls.

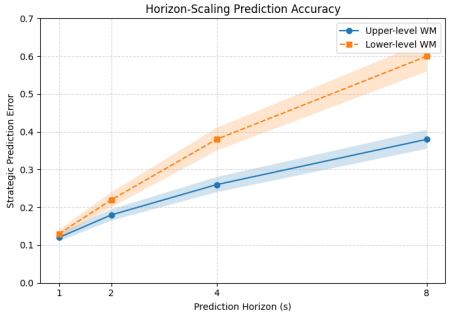


Figure 3: Prediction error on high-level features versus rollout horizon for the upper and lower world models. The upper WM's advantage grows with horizon.

on a log-scale axis. **Our results show that hierarchical search outperforms flat search at every compute budget, and plateaus with an order of magnitude less compute.**

The hierarchical curve strictly outperforms the flat baseline at every computational budget. At the lowest setting (64 evaluations), hierarchical search already reaches 0.50 reactive score while flat search reaches only 0.10. This is because a flat MPPI with 64 samples in a 160-dimensional action space is essentially random sampling, whereas the same budget in 10-dimensional anchor space provides meaningful coverage. The hierarchical curve plateaus by $\sim$ 1000 WM calls while flat search continues improving but never closes the gap, even with $3\times$ more compute. We additionally verify these results also translate to wall-clock time (Appendix E.3). A full hierarchical replan takes 69 ms on a single GPU, against 220 ms for flat MPPI at its best achievable performance.

4.4 Representation Analysis

We analyze the representations learned by our hierarchical world model, asking whether and how the two levels encode different information. We approach this in three complementary ways below.

Linear probing with double dissociation We train linear probes on frozen latent states from the upper WM, the lower WM, and a flat WM baseline, predicting semantic, high-level features (current lane ID, maneuver, lateral offset from lane center, distance to goal) and kinematic, low-level features (distance to nearest agent, ego speed, longitudinal jerk, lateral acceleration). All features were derived from ground-truth annotations provided by nuPlan. Figure 4 reports probe R^2 for each level-feature pair, showing a clear double dissociation where upper latents predict semantic features well ($R^2 \geq 0.78$) and kinematic features poorly ($R^2 \leq 0.22$), while lower latents show the opposite pattern. The flat WM predicts both with intermediate accuracy showing no systematic difference between feature classes. Under nonlinear probing (Appendix E.5, Fig. 6), the dissociation narrows substantially, confirming that both levels encode both feature types but only linearly expose those aligned with their timescale. The flat WM shows no dissociation under either probe, with comparable scores on both feature classes, indicating that it has not learned to factorize information by timescale as the hierarchical WM does.

Horizon-scaling prediction accuracy We roll out both world models from shared initial states for horizons of 1s, 2s, 4s, and 8s and measure prediction error against ground truth (Figure 3). Lower-level prediction error grows rapidly with horizon; upper-level error grows more slowly, and the relative advantage of the upper WM on high-level features increases with horizon. At 1s, the two levels predict these features comparably; by 4s and 8s the gap widens substantially. This horizon-dependent divergence is consistent with the upper level operating at a coarser temporal scale: by predicting only at anchor timesteps, it accumulates fewer

Table 4: Effect of the upper-level latent source on search. All rows share the full system's base policy and the same two-stage search. Only the Stage 1 latent source differs. Test14-hard R.

Upper-level source	CLS (↑)
Base policy, no search	82.8
Hierarchical upper WM	**86.9**
Flat WM (subsampled)	82.8
Flat WM (coarse rollout)	80.1

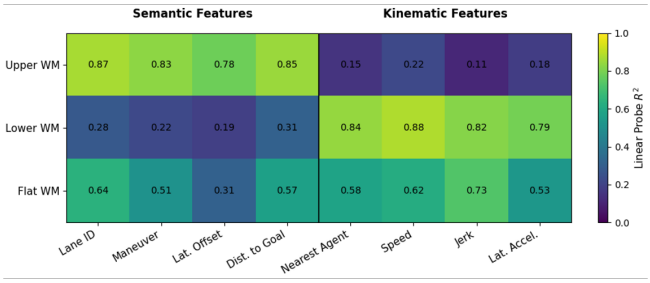


Figure 4: Linear probe R^2 for upper, lower, and flat WM latents on high-level and low-level features. Upper and lower WM latents show a double dissociation; the flat WM does not.

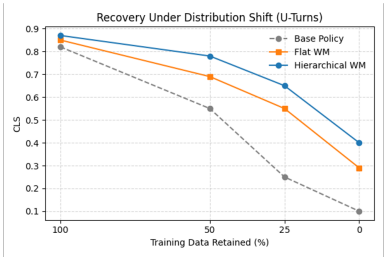


Figure 5: CLS (↑) on held-out U-turns versus fraction of U-turn scenarios retained in training. All systems share identical training data at each retention level.

compounding errors over a fixed horizon than the lower level does at full resolution. The advantage grows with horizon, as expected when prediction error compounds multiplicatively with the number of transition steps.

Hierarchical training makes latents useful for search A natural concern is that any model's latents, read out at anchor timesteps, would support search equally well. We test this by replacing only the source of the Stage 1 latents, keeping the distilled base policy, verification, and search procedure fixed, and training a reward model on each flat source with the same objective as RM^{up}. From a base policy at 82.8 Test14-hard R without search, the hierarchical upper WM raises the score to 86.9, while subsampled flat latents leave it unchanged and coarse-rollout flat latents lower it (Table 4). The gain from search therefore comes from the hierarchically trained representation.

Table 5: Alignment ablation on Test14-hard (NR / R). Matched parameters and compute across all configurations.

	Flat Search	Hier. Search
Flat WM	77.2 ±3 / 72.5 ±4	80.4 ±2 / 74.1 ±3
Hier. WM	79.1 ±3 / 75.8 ±5	**83.8** ±2 / **86.9** ±2

4.5 Alignment Ablation

We run the full 2×2 matrix crossing world model structure (flat vs hierarchical) with search structure (flat MPPI over dense actions vs two-stage MPPI with anchor-level coarse search). All four configurations are trained with matched parameter counts and compute budgets (Table 5).

The aligned hierarchical configuration (hierarchical WM + hierarchical search) outperforms the other three alternatives. Both mismatched configurations (hierarchical WM with flat search and flat WM with hierarchical search) perform substantially worse than the aligned version. This shows that **the gain from hierarchy requires matched hierarchy across both the world model and the search procedure**, and neither alone is sufficient.

5 Conclusion

We introduced HiPlan, which aligns a hierarchical world model with a hierarchical driving policy to enable two-stage search, and showed that the two levels learn timescale-factorized representations that improve both the corrections distilled during training and the search at inference. A natural extension is to learn the hierarchy itself. HiPlan uses uniformly spaced anchors over a fixed horizon, which works well for the horizons studied here, but other domains or more complex settings may benefit from additional levels, non-uniform spacing, or learned placement, and discovering the right temporal abstraction from data rather than imposing it remains an open question. The world model is also trained on the same demonstrations as the base policy, which leaves unused a key advantage of world-model planning, namely learning from large-scale logs with unrecorded actions. Finally, HiPlan operates on processed scene representations rather than raw sensor input, which matches standard practice in nuPlan and CARLA but means the framework does not address the sensor-level world modeling required for fully end-to-end driving.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods. We have not used generative AI tools for developing theoretical models or conceptual frameworks, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, cleaning and reformatting datasets, or interpreting results, and generating synthetic datasets, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, assisting with translation, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for editing the paper to improve readability. We have reviewed all AI-assisted work. All AI-assisted code was reviewed and validated by the authors. All AI-edited text was checked against the underlying experimental results by the authors. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies world models for planning tasks and does not involve human subjects or user studies. All benchmarks used for evaluation are established public research datasets, used for their intended research purpose. At this stage, the work remains an early research contribution, and any practical societal effects will depend on substantial future progress in model design, scaling, validation, and system integration.

REPRODUCIBILITY STATEMENT

We have made every effort to ensure that our results can be reproduced. Section 3 describes the full HiPlan framework, and the appendix provides the remaining details needed to reimplement it. Appendices B and C specify the architectures of the base policy, the hierarchical world model, the reward models and the critic, including all training objectives. Appendix D describes the four-phase training pipeline, buffer management and distillation procedure, and Algorithms 1–3 give the complete two-stage search. All hyperparameters for the world model, search and distillation are listed in Tables 8 and 12, and the compute resources and training cost are reported in Appendix E.3. All experiments use the public nuPlan, interPlan and CARLA Longest6 v2 benchmarks with their standard splits and closed-loop evaluation protocols. HiPlan results are reported as the mean and standard deviation over three seeds, and baseline numbers are taken from the original publications on the same splits and simulation modes.

REFERENCES

[1] Christopher M Bishop. Mixture density networks. 1994.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] Christian Gumbsch, Noor Sajid, Georg Martius, and Martin V Butz. Learning hierarchical world models with adaptive temporal abstractions from discrete latent dynamics. In *The Twelfth International Conference on Learning Representations*, 2023.

[8] Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. Deep hierarchical planning from pixels. *Advances in Neural Information Processing Systems*, 35:26091–26104, 2022.

[9] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[10] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

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

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

[13] 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.

[14] 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. *arXiv preprint arXiv:2506.05294*, 2025.

[15] 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*, pages 7953–7963, 2023.

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

[17] 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.

[18] Napat Karnchanachari, Dimitris Geromichalos, Kok Seang Tan, Nanxiang Li, Christopher Eriksen, Shakiba Yaghoubi, Noushin Mehdipour, Gianmarco Bernasconi, Whye Kit Fong, Yiluan Guo, et al. Towards learning-based planning: The nuplan benchmark for real-world autonomous driving. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 629–636. IEEE, 2024.

[19] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

[20] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948*, 2017.

[21] 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*, pages 142–158. Springer, 2024.

[22] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

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

[24] Han Lu, Xiaosong Jia, Yichen Xie, Wenlong Liao, Xiaokang Yang, and Junchi Yan. Activead: Planning-oriented active learning for end-to-end autonomous driving. *arXiv preprint arXiv:2403.02877*, 2024.

[25] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[26] 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.

[27] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[28] Robin Schiewer, Anand Subramoney, and Laurenz Wiskott. Exploring the limits of hierarchical world models in reinforcement learning. *Scientific reports*, 14(1):26856, 2024.

[29] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[30] 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.

[31] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International conference on machine learning*, pages 3540–3549. PMLR, 2017.

[32] Yi Wan and Richard S Sutton. Toward discovering options that achieve faster planning. *arXiv preprint arXiv:2205.12515*, 2022.

[33] Xiaofeng Wang, Zheng Zhu, Guan Huang, Xinze Chen, Jiagang Zhu, and Jiwen Lu. Drivedreamer: Towards real-world-drive world models for autonomous driving. In *European conference on computer vision*, pages 55–72. Springer, 2024.

[34] Xinshuo Weng, Boris Ivanovic, Yan Wang, Yue Wang, and Marco Pavone. Para-drive: Parallelized architecture for real-time autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15449–15458, 2024.

[35] Wancong Zhang, Basile Terver, Artem Zholus, Soham Chitnis, Harsh Sutaria, Mido Assran, Randall Balestriero, Amir Bar, Adrien Bardes, Yann LeCun, et al. Hierarchical planning with latent world models. *arXiv preprint arXiv:2604.03208*, 2026.

[36] 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*, pages 15222–15232, 2021.

[37] Guosheng Zhao, Yaozeng Wang, Xiaofeng Wang, Zheng Zhu, Tingdong Yu, Guan Huang, Yongchen Zai, Ji Jiao, Changliang Xue, Xiaole Wang, et al. Unidrivedreamer: A single-stage multimodal world model for autonomous driving. *arXiv preprint arXiv:2602.02002*, 2026.

[38] Wenzhao Zheng, Ruiqi Song, Xianda Guo, Chenming Zhang, and Long Chen. Genad: Generative end-to-end autonomous driving. In *European Conference on Computer Vision*, pages 87–104. Springer, 2024.

[39] 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.

[40] Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. *arXiv preprint arXiv:2411.04983*, 2024.

A Related Work

Table 6 compares HiPlan to related methods along key design axes.

Table 6: Comparison of related methods. **Hier. Policy**: hierarchical policy structure. **Hier. WM**: hierarchical world model. **Test-time Search**: search-based correction at inference. **Domain**: primary evaluation domain.

Method	Hier. Policy	Hier. WM	Test-time Search	Learning Signal	Domain
Imitation learning for driving					
Diffusion Planner (39)	✗	✗	✗	Expert demos	Driving
Flow Planner (30)	✗	✗	✗	Expert demos	Driving
PLUTO w/refine (3)	✗	✗	Rule-based	Expert demos + rules	Driving
HiDrive (17)	✓	✗	✗	Expert demos + POMDP	Driving
BridgeDrive (23)	✓	✗	✗	Expert demos	Driving
RL with world models					
Dreamer (v1–v3) (9)	✗	✗	✗	Reward	DMC / Atari
CaRL (13)	✗	✗	✗	Reward (RC)	Driving
Director (8)	✓	✗	✗	Reward	DMC / Atari
THICK (7)	✗	✓	✗	Reward	DMC
Schiewer et al. (28)	✗	✓	✗	Reward	DMC
Test-time planning with learned models					
SAILOR (14)	✗	✗	✓(flat)	Expert demos + rollouts	Manipulation
HWM (35)	✗	✗	✓(hier.)	Goal-conditioned	Manip. / Nav.
HiPlan (Ours)	✓	✓	✓(hier.)	Expert demos + rollouts	Driving

HiPlan is the only method that combines a hierarchical policy, a coupled hierarchical world model, and hierarchical test-time search in the imitation learning setting. Existing hierarchical driving policies (HiDrive, BridgeDrive) decompose the policy but have no world model for test-time correction. THICK learns multi-scale dynamics through sparse latent state changes but relies on reward signals and does not couple its levels through cross-level conditioning or joint training. HWM trains independent world models at different temporal resolutions for hierarchical MPC, but operates in a goal-conditioned setting rather than learning from demonstrations. SAILOR introduced test-time planning from expert and policy rollout data but uses a flat world model and was demonstrated only on short-horizon manipulation. HiPlan bridges these lines: the upper and lower world models are structurally coupled through cross-level conditioning, anchor-boundary resets, and joint fine-tuning, enabling efficient two-stage search at driving-length horizons.

A separate line of work targets end-to-end driving by jointly learning perception and planning from raw sensor input, including UniAD (12), VAD (16), GenAD (38), and related approaches (24; 34; 2). These methods predict trajectories directly from camera or LiDAR streams, bundling scene understanding, prediction, and planning into a single network. We do not compare against this line because HiPlan operates at the planning level on processed scene representations (vectorized maps, tracked agents, ego state), as is standard in nuPlan and CARLA evaluation. Extending the hierarchical world-model framework to sensor-level prediction would require world models capable of predicting at the pixel or point-cloud level, which is a substantial scaling challenge that we leave to future work.

Recent work has also explored RL with learned dynamics for autonomous planning (33; 37), but these methods have not been evaluated on standard closed-loop planning benchmarks. We compare against RL-based planners that do report on these benchmarks in our main results.

B Hierarchical Base Policy

This appendix provides full architectural and training details for the hierarchical base policy summarized in Section 3.2.

B.1 Overview

The hierarchical policy decomposes trajectory generation into two stages: sparse anchor prediction followed by dense trajectory generation via conditional flow matching. The architecture consists of three modules: (1) an upper-level encoder that extracts scene context, (2) an anchor predictor that generates sparse probabilistic waypoints, and (3) a flow matching trajectory generator that produces dense trajectories conditioned on anchors. Full hyperparameters are in Table 7.

B.2 Problem Formulation

Let $\mathcal{S}_t = \{E_t, A_t, M\}$ 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; $A_t = \{A_t^{(i)}\}_{i=1}^{N_t}$ is the set of neighboring agent histories; and $M = \{\ell_1, \ldots, \ell_L\}$ is the local map represented as lane polylines. The 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.

B.3 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. This representation is shared between the base policy and the world model (Section 3.3).

Ego encoder. The ego history is encoded using a temporal transformer:

$$h_{\text{ego}} = \text{TemporalTransformer}\left(\{W_e \cdot e_\tau + \text{PE}(\tau)\}_{\tau=t-T_h+1}^{t}\right) \tag{4}$$

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 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:

$$f_{\text{agent}}^{(i)} = \text{MeanPool}\left(\text{TemporalTransformer}\left(\{W_a \cdot \tilde{a}_\tau^{(i)} + \text{PE}(\tau)\}\right)\right) \tag{5}$$

$$f_{\text{agents}} = \text{MultiHeadSelfAttn}\left(\{f_{\text{agent}}^{(i)}\}_{i=1}^{N_t}\right) \tag{6}$$

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:

$$f_{\text{map}} = \text{MultiHeadSelfAttn}\left(\{\text{MeanPool}(h_{\text{lane}}^{(j)})\}_{j=1}^{L}\right) \tag{7}$$

Cross-modal fusion. The final context representation is obtained via cross-attention:

$$f_{\text{up}} = \text{CrossAttn}(Q = f_{\text{ego}},\ K = V = [f_{\text{agents}};\ f_{\text{map}}]) \tag{8}$$

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.

B.4 Anchor Predictor

The anchor predictor generates K sparse probabilistic waypoints $W = \{w_1, \ldots, w_K\}$ at uniformly spaced time indices $\{t_k\}_{k=1}^{K}$ where $t_k = t + k \cdot \lfloor T_f/K \rfloor$. Each anchor is modeled using a Mixture

Density Network (MDN) (1) with M Gaussian components:

$$p(w_k|f_{\text{up}}^{\text{EMA}}) = \sum_{m=1}^{M} \pi_{k,m} \cdot \mathcal{N}(w_k;\ \mu_{k,m},\ \Sigma_{k,m}) \tag{9}$$

The MDN head outputs mixture weights $\pi_k \in \Delta_{M-1}$ 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 $m^* \sim \text{Categorical}(\pi_k)$, 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.

B.5 Flow Matching Trajectory Generator

The lower-level generator produces dense trajectories $\tau = \{e_{t+1}, \ldots, e_{t+T_f}\}$ conditioned on sampled anchors 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. Using a linear interpolation path $\tau_t = (1-t)\tau_0 + t\tau_1$ for $t \in [0, 1]$, the conditional vector field is $u_t(\tau|\tau_1) = \tau_1 - \tau_0$. The training objective is:

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t\sim\mathcal{U}[0,1],\ \tau_0\sim\mathcal{N}(0,I),\ \tau_1\sim p_{\text{data}}} \left[\|v_\theta(\tau_t, t, W, f_{\text{up}}) - (\tau_1 - \tau_0)\|_2^2\right] \tag{10}$$

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 $f_{\text{cond}} = [f_{\text{up}};\ \text{Enc}_W(W)]$, 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:

$$\tilde{v}_\theta = v_\theta(\tau_t, t, W, f_{\text{up}}) + \omega\,(v_\theta(\tau_t, t, W, f_{\text{up}}) - v_\theta(\tau_t, t, \varnothing, f_{\text{up}})) \tag{11}$$

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

Sampling. At inference, we generate trajectories by solving the ODE $\frac{d\tau_t}{dt} = v_\theta(\tau_t, t, W, f_{\text{up}})$ 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.

B.6 Loss Functions

Anchor predictor loss. The anchor predictor is trained to minimize negative log-likelihood:

$$\mathcal{L}_{\text{anchor}} = -\sum_{k=1}^{K} \log p(w_k^{\text{GT}}|f_{\text{up}}^{\text{EMA}}) \tag{12}$$

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.

Anchor consistency loss. Applied to the fully denoised trajectory:

$$\mathcal{L}_{\text{consist}} = \mathbb{E}_{\tau_0,\tau_1} \left[\sum_{k=1}^{K} \|\hat{\tau}_1[t_k] - \text{sg}(w_k)\|_2^2\right] \tag{13}$$

where $\hat{\tau}_1$ is obtained by integrating the learned flow and $\text{sg}(\cdot)$ denotes stop-gradient on anchors.

Smoothness regularization.

$$\mathcal{L}_{\text{smooth}} = \sum_{i=2}^{T_f-1} \|\hat{\tau}_1[i+1] - 2\hat{\tau}_1[i] + \hat{\tau}_1[i-1]\|_2^2 \tag{14}$$

Total loss. During joint fine-tuning (Stage 3), the total loss combines all objectives:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CFM}} + \lambda_{\text{anchor}}\mathcal{L}_{\text{anchor}} + \lambda_{\text{consist}}\mathcal{L}_{\text{consist}} + \lambda_{\text{smooth}}\mathcal{L}_{\text{smooth}} + \lambda_{\text{recon}}\mathcal{L}_{\text{recon}} + \lambda_{\text{div}}\mathcal{L}_{\text{div}} \quad (15)$$

B.7 Training Procedure

Training proceeds in three stages with curriculum scheduling.

Stage 1: Anchor predictor training. Ground-truth anchors are extracted from expert trajectories at the specified time indices. The anchor predictor is trained to minimize $\mathcal{L}_{\text{anchor}} + \lambda_{\text{recon}}\mathcal{L}_{\text{recon}} + \lambda_{\text{div}}\mathcal{L}_{\text{div}}$.

Stage 2: Flow matching pretraining. The flow network is pretrained using ground-truth anchors as conditioning, allowing it to learn trajectory dynamics before being exposed to noise from sampled anchors.

Stage 3: Joint fine-tuning. All components are trained end-to-end with $\mathcal{L}_{\text{total}}$. Anchor conditioning transitions from ground-truth to sampled anchors for the flow network.

Curriculum 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.

B.8 Inference Procedure

At inference, we generate N_{samples} diverse trajectory candidates. Diversity arises from three sources: MDN mixture component sampling, Gaussian noise in anchor sampling, and prior noise in flow integration. The full sampling procedure is given in Algorithm 2.

Algorithm 2 Hierarchical Base Policy Inference

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

$f_{\text{up}} \leftarrow \text{Encoder}(\mathcal{S}_t)$
$\{\pi_k, \mu_k, \sigma_k\}_{k=1}^{K} \leftarrow \text{AnchorPredictor}(f_{\text{up}})$
for $n = 1$ to N_{samples} **do**
  for $k = 1$ to K **do**
    $m^* \sim \text{Categorical}(\pi_k)$
    $w_k^{(n)} \sim \mathcal{N}(\mu_{k,m^*}, \Sigma_{k,m^*})$
  end for
  $W^{(n)} \leftarrow \{w_1^{(n)}, \ldots, w_K^{(n)}\}$
  $\tau_0^{(n)} \sim \mathcal{N}(0, I)$
  for $s = 0$ to $N_{\text{steps}} - 1$ **do**
    $t \leftarrow s/N_{\text{steps}}$
    $v_{\text{cond}} \leftarrow v_\theta(\tau_t^{(n)}, t, W^{(n)}, f_{\text{up}})$
    $v_{\text{uncond}} \leftarrow v_\theta(\tau_t^{(n)}, t, \varnothing, f_{\text{up}})$
    $\tilde{v} \leftarrow v_{\text{cond}} + \omega(v_{\text{cond}} - v_{\text{uncond}})$
    $\tau_{t+1/N_{\text{steps}}}^{(n)} \leftarrow \tau_t^{(n)} + \tilde{v}/N_{\text{steps}}$
  end for
  $\tau^{(n)} \leftarrow \tau_1^{(n)}$
end for
return $\{\tau^{(n)}\}_{n=1}^{N_{\text{samples}}}$

B.9 HYPERPARAMETERS

Table 7 lists out the hyperparameters we used for the hierarchical base policy.

Table 7: Hyperparameters for the hierarchical base policy.

Hyperparameter	**Symbol**	**Value**
Architecture		
Model dimension	d	256
Number of attention heads	H	8
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)	$N_{\text{steps}}^{\text{train}}$	10
Guidance scale	ω	2.0
Anchor guidance step size	$\lambda_{\text{anchor}}^{\text{guide}}$	0.1
Unconditional dropout	p_{uncond}	0.1
Loss weights		
Reconstruction loss	λ_{recon}	0.1
Diversity loss	λ_{div}	0.01
Consistency loss	λ_{consist}	1.0
Smoothness loss	λ_{smooth}	0.1
Anchor loss	λ_{anchor}	1.0
Training		
EMA momentum	α	0.999
Learning rate	–	1×10^{-4}
Batch size	–	64
Optimizer	–	AdamW
Weight decay	–	0.01
Gradient clip norm	–	1.0

C HIERARCHICAL WORLD MODEL AND REWARD MODELS

This appendix provides full architectural and training details for the hierarchical world model, reward models, and critic summarized in Section 3.3 and Section 3.4.

C.1 OVERVIEW

The hierarchical world model consists of two coupled Recurrent State-Space Models (RSSMs) (9) operating at different temporal resolutions. The upper-level RSSM predicts transitions at the anchor timescale ($\Delta_{\text{anchor}} = \lfloor T_f/K \rfloor = 16$ steps), treating anchor waypoints as actions. The lower-level RSSM predicts transitions at full 10Hz resolution, conditioned on upper-level context and reset at anchor boundaries. Both levels are trained on hybrid data $\mathcal{D}_{\text{expert}} \cup \mathcal{D}_{\text{policy}}$.

C.2 UPPER-LEVEL WORLD MODEL

C.2.1 STATE REPRESENTATION

The upper-level RSSM maintains a deterministic-stochastic state pair at the anchor timescale:

$$s_k^{\text{up}} = (h_k^{\text{up}}, z_k^{\text{up}}) \tag{16}$$

where $h_k^{\text{up}} \in \mathbb{R}^{d_h}$ is the deterministic recurrent state and z_k^{up} is the stochastic latent. We use discrete stochastic latents following DreamerV3 (10): z_k^{up} consists of N_z categorical variables each with C_z classes, yielding a total stochastic dimensionality of $N_z \times C_z$.

C.2.2 TRANSITION MODEL

The upper-level transitions are driven by anchor waypoints w_k as actions:

$$h_k^{\text{up}} = \text{GRU}(h_{k-1}^{\text{up}}, [z_{k-1}^{\text{up}}, w_{k-1}]) \tag{17}$$

$$z_k^{\text{up}} \sim q(z \mid h_k^{\text{up}}, o_{t_k}) \quad \text{(posterior, training only)} \tag{18}$$

$$\hat{z}_k^{\text{up}} \sim p(z \mid h_k^{\text{up}}) \quad \text{(prior, for imagination)} \tag{19}$$

The posterior is computed by encoding the observation at anchor timestep t_k through the shared encoder:

$$f_{\text{up}}^{t_k} = \text{Encoder}(\mathcal{S}_{t_k}) \tag{20}$$

$$\mu_q, \sigma_q = \text{MLP}_{\text{posterior}}([h_k^{\text{up}};\ f_{\text{up}}^{t_k}]) \tag{21}$$

$$z_k^{\text{up}} \sim \text{Categorical}(\text{softmax}(\mu_q)) \tag{22}$$

The encoder backbone is shared with the base policy (Section 3), grounding the world model's latent space in the same scene representation the policy uses for anchor generation. We use the EMA copy of the encoder for stable conditioning during training.

C.2.3 DECODER

Rather than reconstructing full observations, the upper-level decoder reconstructs a compact, planning-relevant state summary at each anchor timestep:

$$\hat{o}_k^{\text{coarse}} = \text{MLP}_{\text{decoder}}(s_k^{\text{up}}) \to (x_k, y_k, \theta_k, v_k, d_{\text{agents}}, d_{\text{lane}}) \tag{23}$$

consisting of ego position and heading, ego velocity, distances to nearest agents, and lane-relative lateral offset. This keeps the upper latent space focused on strategic, planning-relevant features rather than pixel-level reconstruction.

C.2.4 TRAINING LOSS

$$\mathcal{L}_{\text{upper}} = \sum_k \left[\|\hat{o}_k^{\text{coarse}} - o_k^{\text{coarse}}\|^2 + \beta \cdot \text{KL}\left(q(z_k^{\text{up}} \mid h_k^{\text{up}}, o_{t_k}) \parallel p(z_k^{\text{up}} \mid h_k^{\text{up}})\right)\right] \tag{24}$$

We use β-VAE style annealing, starting $\beta = 0.1$ and linearly increasing to $\beta = 1.0$ over training. We apply free bits (19): the KL is clamped to at least 1 nat per latent dimension to prevent posterior collapse.

C.3 LOWER-LEVEL WORLD MODEL

C.3.1 STATE REPRESENTATION

The lower-level RSSM maintains a deterministic-stochastic state pair at full 10Hz resolution:

$$s_t^{\text{lo}} = (h_t^{\text{lo}}, z_t^{\text{lo}}) \tag{25}$$

with $h_t^{\text{lo}} \in \mathbb{R}^{d_h}$ and z_t^{lo} using the same discrete latent specification as the upper level.

C.3.2 TRANSITION MODEL

The lower-level transitions are conditioned on primitive actions a_t and the upper-level latent state for the current anchor segment:

$$h_t^{\text{lo}} = \text{GRU}(h_{t-1}^{\text{lo}}, [z_{t-1}^{\text{lo}}, a_t, z_{k(t)}^{\text{up}}]) \tag{26}$$

$$z_t^{\text{lo}} \sim q(z \mid h_t^{\text{lo}}, o_t) \quad \text{(posterior)} \tag{27}$$

$$\hat{z}_t^{\text{lo}} \sim p(z \mid h_t^{\text{lo}}) \quad \text{(prior)} \tag{28}$$

where $z_{k(t)}^{\text{up}}$ is the upper-level stochastic state for whichever anchor segment timestep t falls within. This conditioning tells the lower model the current plan context — which anchor the system is executing toward and what the upper level believes about the scene at that scale.

C.3.3 Anchor Boundary Reset

At each anchor timestep t_k, the lower-level recurrent state is reinitialized from the upper-level state:

$$h^{\text{lo}}_{t_k} = \text{MLP}_{\text{init}}(s^{\text{up}}_k) \tag{29}$$

This reset is the mechanism that enforces hierarchy. Without it, the lower-level model would maintain its own long-term memory across anchor segments, making the upper level redundant. With the reset, the upper level is solely responsible for carrying information across anchor boundaries, while the lower level handles dense dynamics within each segment. The reset also bounds prediction error: compounding occurs only within each 16-step segment rather than over the full 80-step horizon.

C.3.4 Decoder

The lower-level decoder reconstructs full observations from the lower latent state:

$$\hat{o}_t = \text{Decoder}_{\text{lo}}(s^{\text{lo}}_t) \tag{30}$$

including BEV features, agent positions, and ego state at full resolution.

C.3.5 Training Loss

$$\mathcal{L}_{\text{lower}} = \sum_t \left[\|\hat{o}_t - o_t\|^2 + \beta \cdot \text{KL}\left(q(z^{\text{lo}}_t \mid h^{\text{lo}}_t, o_t) \parallel p(z^{\text{lo}}_t \mid h^{\text{lo}}_t) \right) \right] \tag{31}$$

The same β-annealing and free bits schedule is used as for the upper level.

C.4 Joint Training Procedure

Training proceeds in two stages.

Stage A: Separate pretraining ($\sim$50% of WM training budget).

- **Upper level:** Expert trajectories are subsampled at anchor indices (every $\Delta_{\text{anchor}} = 16$ steps). The upper RSSM is trained on these coarse sequences with $\mathcal{L}_{\text{upper}}$.
- **Lower level:** Trained on full-resolution data conditioned on ground-truth upper states. Ground-truth upper states are obtained by encoding the real observations at anchor timesteps through the upper encoder posterior: $z^{\text{up}}_k \sim q(z \mid h^{\text{up}}_k, o_{t_k})$. The lower-level reset mechanism uses these ground-truth upper states.

Separate pretraining ensures each level learns dynamics at its own temporal resolution before being coupled.

Stage B: Joint fine-tuning ($\sim$50% of WM training budget). The upper-level RSSM generates imagined latent sequences using its prior $p(z \mid h^{\text{up}})$ rather than the posterior. The lower-level RSSM is conditioned on these imagined upper states and must reconstruct full-resolution observations. The reconstruction error backpropagates through both levels:

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{upper}} + \lambda_{\text{lower}} \cdot \mathcal{L}_{\text{lower}} \tag{32}$$

with $\lambda_{\text{lower}} = 1.0$. This joint signal is critical: if the upper-level latents are uninformative, the lower level cannot reconstruct accurately, and the joint loss increases. This forces the upper latent space to encode information the lower level depends on.

Both stages use hybrid data $\mathcal{D}_{\text{expert}} \cup \mathcal{D}_{\text{policy}}$. Training on policy rollout data ensures the world model is accurate on both expert states and the off-distribution states the base policy visits when it fails, which is essential for planning recovery trajectories.

C.5 Upper Reward Model

The upper reward model $\text{RM}_{\text{up}} : \mathcal{Z}^{\text{up}} \to \mathbb{R}$ scores upper-level latent states by how expert-like they are. It is trained as a Wasserstein discriminator:

$$\mathcal{L}_{\text{RM}_{\text{up}}} = \mathbb{E}_{s^{\text{up}} \sim \mathcal{D}_{\text{policy}}}[\text{RM}_{\text{up}}(s^{\text{up}})] - \mathbb{E}_{s^{\text{up}} \sim \mathcal{D}_{\text{expert}}}[\text{RM}_{\text{up}}(s^{\text{up}})] + \lambda_{\text{GP}} \cdot \mathcal{L}_{\text{GP}} \tag{33}$$

where the gradient penalty enforces Lipschitz continuity:

$$\mathcal{L}_{\text{GP}} = \mathbb{E}_{\hat{s}\sim p(\hat{s})}\left[(\|\nabla_{\hat{s}}\text{RM}_{\text{up}}(\hat{s})\|_2 - 1)^2\right] \tag{34}$$

with $\hat{s}$ sampled along straight lines between expert and policy upper latent states, and $\lambda_{\text{GP}} = 10.0$.

Architecture. MLP with 3 hidden layers of 256 units each, LayerNorm, and SiLU activations. Input is $[h_k^{\text{up}};\ z_k^{\text{up}}]$.

Role in search. RM_{up} is the primary search driver in Stage 1 of the two-stage search (Section 3.5). It is cheap to evaluate — only $K = 5$ coarse states per rollout — and is sensitive to high-level intent errors such as wrong lane choice, aggressive gap selection, or poorly chosen yield decisions.

C.6 Lower Reward Model

The lower reward model $\text{RM}_{\text{lo}} : \mathcal{Z}^{\text{lo}} \to \mathbb{R}$ scores lower-level latent states for fine-grained trajectory safety. It is trained with the same Wasserstein objective as RM_{up}:

$$\mathcal{L}_{\text{RM}_{\text{lo}}} = \mathbb{E}_{s^{\text{lo}}\sim\mathcal{D}_{\text{policy}}}[\text{RM}_{\text{lo}}(s^{\text{lo}})] - \mathbb{E}_{s^{\text{lo}}\sim\mathcal{D}_{\text{expert}}}[\text{RM}_{\text{lo}}(s^{\text{lo}})] + \lambda_{\text{GP}} \cdot \mathcal{L}_{\text{GP}} \tag{35}$$

Architecture. Same MLP structure as RM_{up}, but operating on $[h_t^{\text{lo}};\ z_t^{\text{lo}};\ z_{k(t)}^{\text{up}}]$ — conditioning on the upper context so the lower reward knows what behavior the agent is attempting. This conditioning is important: the same dense trajectory may be appropriate in one strategic context (e.g., accelerating to merge) and inappropriate in another (e.g., accelerating toward a red light).

Role in search. RM_{lo} serves as the safety verifier in Stage 3 of the two-stage search. It catches tactical failures that the coarse upper reward cannot detect: unsafe passing distances, excessive jerk, near-miss collisions. It is more expensive to evaluate (80 dense states per trajectory) but is only applied to the N_τ candidate trajectories that survive upper-level MPPI, not to the full N_{up} sample set.

Online updating. Both reward models are updated online throughout Phases 1–3 of the training pipeline (Section 3.6). As the base policy improves through distillation, the gap between expert and policy behavior narrows. Online updating ensures the discriminators remain sensitive to the increasingly subtle deviations of the improving policy.

C.7 Upper-Level Critic

The critic $V_{\text{up}} : \mathcal{Z}^{\text{up}} \to \mathbb{R}$ estimates returns beyond the K-step anchor planning horizon. It is trained on λ-returns computed at the anchor timescale:

$$v_k^\lambda = \text{RM}_{\text{up}}(s_k^{\text{up}}) + \gamma^{\Delta_{\text{anchor}}}\left[(1-\lambda)V_{\text{up}}(s_{k+1}^{\text{up}}) + \lambda v_{k+1}^\lambda\right], \quad v_K^\lambda = V_{\text{up}}(s_K^{\text{up}}) \tag{36}$$

Note the effective per-anchor discount $\gamma^{\Delta_{\text{anchor}}} = 0.99^{16} \approx 0.851$, which correctly accounts for the temporal abstraction — each anchor step spans 16 real timesteps, so the discount must compound accordingly.

The critic is trained to minimize:

$$\mathcal{L}_V = \mathbb{E}\left[\|V_{\text{up}}(s_k^{\text{up}}) - \text{sg}(v_k^\lambda)\|^2\right] \tag{37}$$

where $\text{sg}(\cdot)$ denotes stop-gradient on the target.

Ensemble and uncertainty. We train an ensemble of 5 critic networks. At inference, we randomly sample 2 from the ensemble and compute:

$$V_{\text{up}}^{\text{inf}}(s) = \text{mean}(V_{\text{up}}^i(s), V_{\text{up}}^j(s)) - \lambda_{\text{unc}} \cdot \text{std}(V_{\text{up}}^i(s), V_{\text{up}}^j(s)) \tag{38}$$

with $\lambda_{\text{unc}} = 1.0$. The uncertainty penalty encourages conservative planning: the system avoids states where critic estimates disagree, which tend to be states far from the training distribution where value estimates are unreliable. Each critic network is an MLP: $[h_k^{\text{up}};\ z_k^{\text{up}}] \to [256, 256, 256] \to 1$, with LayerNorm and SiLU activations.

No lower-level critic. We do not train a lower-level critic. The lower-level verification (Stage 3 of search) evaluates RM_{lo} over the full dense trajectory within the planning horizon, so there is no tail beyond the planning window that requires value estimation. The upper-level critic handles long-horizon value estimation at the appropriate abstraction level; adding a lower critic would increase complexity without a clear payoff.

C.8 HYPERPARAMETERS

Table 8 lists out the hyperparameters we used for our world model.

Table 8: Hierarchical world model, reward model, and critic hyperparameters.

Hyperparameter	**Symbol**	**Value**
Upper-level world model		
Deterministic state dim	d_h	256
Stochastic latent (categorical)	$N_z \times C_z$	32×32
Anchor timescale	Δ_{anchor}	16 steps
KL weight (initial $\rightarrow$ final)	β	$0.1 \rightarrow 1.0$
Free bits	–	1.0 nat
Lower-level world model		
Deterministic state dim	d_h	256
Stochastic latent (categorical)	$N_z \times C_z$	32×32
Lower loss weight	λ_{lower}	1.0
Joint training		
Stage A fraction	–	50% of WM budget
Stage B fraction	–	50% of WM budget
Reward models		
Hidden layers	–	$[256, 256, 256]$
Activation	–	SiLU
Normalization	–	LayerNorm
Gradient penalty coefficient	λ_{GP}	10.0
Upper-level critic		
Ensemble size	–	5
Ensemble sample size (inference)	–	2
Uncertainty penalty	λ_{unc}	1.0
λ-return coefficient	λ	0.95
Discount factor	γ	0.99
Effective per-anchor discount	$\gamma^{\Delta_{\text{anchor}}}$	0.851
Hidden layers	–	$[256, 256, 256]$
Training		
Learning rate (all components)	–	1×10^{-4}
Batch size	–	64
Optimizer	–	AdamW
Weight decay	–	0.01
Gradient clip norm	–	1.0

D TWO-STAGE SEARCH AND TRAINING PIPELINE

This appendix provides the full search procedure, safety threshold calibration, training pipeline, and hierarchical distillation details summarized in Section 3.5 and Section 3.6.

D.1 FULL SEARCH PSEUDOCODE

Algorithm 3 provides the complete two-stage search procedure with all hyperparameters. At each planning step, the system encodes the current observation, runs upper-level MPPI to find corrected anchors, generates dense trajectory candidates via flow matching, verifies them in the lower world model, and selects the best surviving candidate.

Fallback behavior. If all N_τ candidates are rejected by the safety filter, the system selects the trajectory with the lowest maximum speed. This conservative fallback ensures the system always produces a safe action even when the world model cannot find a confidently safe plan. In practice, this fallback is triggered in fewer than 2% of planning steps.

Residual self-regulation. The MPPI search operates over residual perturbations ΔW to the base policy's anchors W^{base}. When the base policy is already performing well, the reward landscape around W^{base} is approximately flat, and MPPI converges to $\Delta W \approx 0$. The system then executes the base plan with negligible computational overhead beyond the initial scoring pass. This relies on the reward model ranking the neighborhood of the base plan correctly. When the upper-level latents come from a flat world model rolled out at the anchor timescale, search lowers the score of the same base policy from 82.8 to 80.1 (Table 4), so the property holds for the hierarchical reward model but not in general.

D.2 Safety Threshold Calibration

The rejection threshold $R_{\text{threshold}}$ in Stage 3 determines which dense trajectories are filtered out as unsafe. We calibrate it on the validation set as follows:

1. Roll out the base policy on all validation scenarios and encode the resulting trajectories in the lower world model.
2. Compute per-timestep RM_{lo} scores for each trajectory.
3. Compute the minimum per-timestep RM_{lo} score for each expert trajectory (from $\mathcal{D}_{\text{expert}}$) on the same scenarios.
4. Set $R_{\text{threshold}}$ at the 10th percentile of the expert minimum scores.

This calibration means we reject trajectories whose worst-case safety score falls below the worst-case score of 90% of expert trajectories. The threshold is computed once after Phase 1 training and held fixed throughout Phases 2 and 3. Table 9 shows that performance is stable across a range of threshold choices.

Table 9: Sensitivity to the safety threshold percentile on Test14-hard.

Percentile	5th	10th	20th	30th
Test14-hard R	86.4	86.9	86.5	85.8
Rejection rate	3.1%	5.7%	11.2%	18.4%

Learned filter versus geometric checks. Both nuPlan CLS and CARLA DS penalize collisions and drivable-area violations with rule-based checks, so all reported gains are measured against explicit constraints rather than against RM^{lo} itself. To test the learned filter directly, we add a hard geometric filter before it, which rejects candidates whose bounding box overlaps a predicted agent box, that leave the drivable area, or that exceed the speed limit. Adding the geometric filter changes the score by 0.2 CLS and the collision rate by 0.4 points (Table 10), so RM^{lo} rarely accepts geometrically unsafe plans. The two filters disagree in an informative direction (Table 11). RM^{lo} rejects 4.9% of candidates that pass the geometric check, which are graded failures such as unsafe passing distance or excessive jerk that a binary check cannot express. All other results are reported without the geometric filter.

Table 10: Adding a hard geometric filter before RM^{lo} on Test14-hard.

Configuration	Test14-hard R	Collision rate	Geometric-only rejections
RM^{lo} only	86.9	2.3%	–
Geometric filter + RM^{lo}	87.1	1.9%	1.8%

Table 11: Agreement between RM^{lo} and the geometric filter over all Stage 3 candidates on Test14-hard.

	Geometric accepts	Geometric rejects
RM^{lo} accepts	91.4%	1.8%
RM^{lo} rejects	4.9%	1.9%

D.3 Training Pipeline

The full system is trained in four phases. We describe each in detail, including data management, interaction budgets, and component update schedules.

Phase 0: Base policy training (offline). The hierarchical base policy is trained on expert demonstrations $\mathcal{D}_{\text{expert}}$ alone, following the three-stage curriculum described in Appendix B. No environment interaction is required. The output is a trained base policy π^0_{base} that performs well on-distribution but has no recovery mechanism.

Phase 1: World model and reward pretraining ($\sim$20% of interaction budget). We deploy π^0_{base} in the environment to collect rollout data $\mathcal{D}_{\text{policy}}$. This data includes both successful trajectories and failures; the failures are particularly important as they define the off-distribution states the planning module must handle. All learned components are then trained on hybrid data $\mathcal{D}_{\text{expert}} \cup \mathcal{D}_{\text{policy}}$:

- Upper and lower world models: separate pretraining (Stage A), then joint fine-tuning (Stage B), as described in Appendix C.
- RM_{up} and RM_{lo}: trained as Wasserstein discriminators on expert vs. policy latent states.
- V_{up}: trained on λ-returns from upper-level imagined rollouts.
- Safety threshold $R_{\text{threshold}}$: calibrated on validation data.

Phase 2: Planning deployment ($\sim$60% of interaction budget). The full two-stage search (Algorithm 3) is deployed. The system collects rollouts where the planner actively corrects the base policy's mistakes. These corrected rollouts are added to $\mathcal{D}_{\text{policy}}$, growing the buffer. All learned components continue training on the growing buffer:

- The world model sees new states — the states visited during recovery differ from both expert states and the base policy's failure states, and the model must be accurate on these intermediate states to plan further recoveries.
- The reward models see harder negatives — as the policy improves through planning, the gap between expert and current policy narrows, forcing the discriminators to become more sensitive to subtle deviations.
- The critic's value estimates improve as the reward model refines, creating a virtuous cycle.

Phase 3: Distillation ($\sim$20% of interaction budget). Search corrections are distilled back into the base policy, as described below. Phases 2 and 3 repeat iteratively.

Buffer management. We maintain a fixed-size replay buffer for $\mathcal{D}_{\text{policy}}$ with first-in-first-out replacement. $\mathcal{D}_{\text{expert}}$ is fixed throughout training. When sampling mini-batches for world model and reward model training, we sample 50% from $\mathcal{D}_{\text{expert}}$ and 50% from $\mathcal{D}_{\text{policy}}$ to prevent the expert signal from being diluted as the policy buffer grows.

D.4 Hierarchical Distillation

Distillation converts test-time compute into permanent policy improvements. Without distillation, the system relies on the planning module for every correction at every timestep, incurring computational cost and depending on world model accuracy. Distillation teaches the base policy to produce corrected behavior directly, so that search is only needed for genuinely novel situations. Distillation

also improves the deployed system itself, since search starting from a better base policy reaches better plans (Appendix E.9).

Hierarchical distillation targets. The two-level policy structure enables targeted distillation at each abstraction level:

- **Anchor predictor:** Fine-tuned via behavioral cloning on (f_{up}, W^*) pairs, where W^* are the corrected anchors from Stage 1 of the search. This teaches the anchor predictor to produce corrected subgoals directly.
- **Flow matching generator:** Fine-tuned on $(W^*, f_{\text{up}}, \tau^*)$ triples, where τ^* is the selected dense trajectory from Stage 4. This teaches the flow generator to produce corrected trajectories given corrected anchors.

This separation is cleaner than distilling into a flat policy. The anchor predictor learns strategic corrections (different lane choice, different gap selection); the flow generator learns tactical corrections (adjusted speed profile, smoother trajectory through corrected waypoints). Each component receives supervision at the abstraction level it operates at.

Selective distillation. We only distill cases where the planner's correction was substantial:

$$\|\Delta W\| = \|W^* - W^{\text{base}}\| > \epsilon_{\text{distill}} \tag{39}$$

with $\epsilon_{\text{distill}} = 0.5$m. Distilling near-zero corrections would reinforce behavior the base policy already produces correctly, wasting training signal and risking overfitting to the planner's idiosyncrasies on easy scenarios.

Iteration dynamics. Phases 2 and 3 repeat for $N_{\text{iter}} = 5$ iterations. Each iteration produces a better base policy that requires less test-time search:

- **Iteration 1:** The base policy π^0_{base} has broad failure modes (e.g., struggles with lane changes in dense traffic, misjudges gaps at unprotected left turns). Phase 2 corrects these; Phase 3 distills corrections into π^1_{base}.
- **Iteration 2:** π^1_{base} handles dense traffic lane changes natively but may still struggle with unprotected left turns and may have developed new edge-case failures from distillation. The planner corrects a different distribution of failures. The reward models see harder negatives as the policy-expert gap narrows.
- **Iterations 3+:** Each iteration narrows the set of situations requiring test-time correction. Common failures are absorbed into the policy; rare, complex failures persist as the domain of online search.

Two mechanisms stabilize the iteration loop. First, $\mathcal{D}_{\text{expert}}$ is fixed and always included in training, preventing both the reward models and the policy from drifting away from expert behavior. Second, the reward model's discriminative formulation adapts automatically to the policy's current level: it scores relative to the expert, not against an absolute standard, so it remains informative even as the policy improves.

Convergence criterion. We run $N_{\text{iter}} = 5$ iterations and report the final checkpoint. The largest improvements occur in iterations 1–2, and the marginal improvement after iteration 4 falls below 0.5 CLS on the validation set, which is within the variation across repeated training runs (Appendix E.9).

D.5 Search Hyperparameters

Table 12 lists out hyperparameters for inference-time search.

Computational resources Training was conducted on 8 NVIDIA H200 GPUs with data-parallel distributed training. Phase 0 (base policy) required approximately 80 GPU-hours; Phase 1 (world model and reward pretraining) approximately 120 GPU-hours; Phases 2–3 iterated five times for

Table 12: Two-stage search hyperparameters.

Hyperparameter	Symbol	Value
Stage 1: Upper-level MPPI		
Number of samples	N_{up}	128
Number of iterations	J_{up}	4
Temperature	τ_{up}	0.5
Initial perturbation std	σ_{init}	2.0m
Stages 2–3: Generation and verification		
Trajectory candidates	N_τ	8
Safety threshold percentile	–	10th
Score combination weight	α	0.5
Distillation		
Correction threshold	$\epsilon_{\text{distill}}$	0.5m
Number of iterations	N_{iter}	5
Convergence threshold	–	0.5 CLS
General		
Discount factor	γ	0.99
Replanning frequency	–	Every timestep (10Hz)

approximately 400 GPU-hours total, with environment rollout collection dominating ($\sim$60% of per-iteration cost) and consuming approximately 20M simulated environment steps in aggregate. The full pipeline thus required approximately 600 GPU-hours, or roughly 3 days on 8 H200s. Inference latency was measured on a single NVIDIA RTX 3090 (Appendix E.3), on which a full hierarchical replan completes in 69 ± 3 ms. The full research project, including ablations and preliminary experiments not reported here, required approximately $3\times$ the final reported compute.

E Results

E.1 Quantitative Analysis

nuPlan The nuPlan CLS aggregates several driving metrics into a single score in $[0, 100]$, computed by closing the loop on a learned planner against either logged (non-reactive, NR) or interactive (reactive, R) traffic. The score is a multiplicative combination of safety multipliers (no at-fault collisions, drivable-area compliance, driving-direction compliance) and a weighted average of comfort and progress metrics (time-to-collision, ego progress along route, speed-limit compliance, longitudinal/lateral comfort). Any safety violation zeroes the multiplier, so a high CLS requires both safe and efficient driving. We use the official nuPlan implementation (18) without modification.

Table 13 reports the metric decomposition. The Log Replay rows correspond to controllers tracking the human-driven expert trajectory and serve as a reference upper bound rather than a competing planner. Our full configuration achieves the best score among learned planners, with the largest gains on collision avoidance (99.02), TTC (95.61), and comfort (99.13). The lower Speed score (94.19 vs. 97–98 for baselines) reflects conservative speed selection from the safety-oriented reward model, traded for these gains on collision and comfort metrics.

CARLA The CARLA Longest6 v2 Driving Score is the product of route completion (RC, fraction of the planned route traversed) and an infraction multiplier that penalizes safety violations: $\text{DS} = \text{RC} \times \prod_i p_i^{n_i}$, where $p_i \in (0, 1]$ is the per-event penalty for infraction type i (collisions with pedestrians, vehicles, or static objects; running red lights or stop signs; off-road driving) and n_i is the number of such infractions. A planner that completes the full route without infractions scores 100; collisions and rule violations multiplicatively reduce the score. We report DS together with RC and per-infraction rates, following the CARLA Leaderboard 2.0 protocol (5).

interPlan We additionally evaluate on the interPlan benchmark (11), which contains 335 specially augmented out-of-distribution scenarios with modified traffic agent density and behavior, designed to test interactive driving in complex and unexpected situations. We use the full-scale test set in *reactive*

Table 13: Detailed nuPlan results with metric decomposition. Score is the overall closed-loop score (CLS). Multiplier metrics (Collisions, Drivability, Direction, Progress) act as multiplicative penalties; weighted metrics (TTC, RC, Speed, Comfort) are averaged into the score. **Bold** indicates best within each column. Values measured on Val14 NR.

Planner	Type	Score	Multiplier				Weighted			
			Collisions	Drivability	Direction	Progress	TTC	RC	Speed	Comfort
IDM	Rule	77.33	89.22	93.02	99.24	96.33	81.57	85.20	97.20	93.11
PLUTO	IL	88.74	95.84	98.48	99.42	98.57	92.49	89.43	98.00	96.00
Diffusion Planner	IL	89.62	95.89	98.30	99.60	99.91	90.52	94.19	97.27	94.63
CARL (General)	RL	91.25	97.36	99.28	99.02	99.37	92.04	94.44	**98.74**	89.09
CARL (nuPlan Tuned)	RL	93.87	97.23	**100.00**	**99.78**	99.55	93.47	96.14	98.41	97.14
Ours (base policy only)	Ours	91.29	97.82	99.56	98.92	99.55	92.04	93.23	88.92	96.00
Ours (full configuration)	Ours	**95.62**	**99.02**	99.39	98.79	**100.00**	**95.61**	**97.56**	94.19	**99.13**
Log Replay (LQR)	Human	93.53	98.79	97.94	99.06	100.00	94.28	99.00	96.50	99.28
Log Replay (iLQR)	Human	96.24	99.55	99.55	99.24	100.00	96.51	99.32	96.46	98.93
Log Replay (Perf.)	Human	96.39	99.78	99.46	99.37	100.00	96.78	99.28	95.86	99.20

Table 14: Performance on CARLA Longest6 v2. **Bold** indicates best.

Planner	Type	DS ↑	RC ↑	Ped ↓	Veh ↓	NS ↓
PlanT (26)	IL	62	**96**	0.07	0.43	3.18
Think2Drive (21)	RL	7	39	0.97	2.55	18.05
Roach (36)	RL	22	77	0.45	2.42	7.12
CaRL (13)	RL	64	82	**0.01**	0.36	1.71
HiPlan (base policy only)	–	57	73	0.38	1.95	4.07
HiPlan (flat WM search)	–	63	82	0.06	0.28	1.24
HiPlan (hier. WM search)	–	**70**	90	0.11	**0.19**	**1.24**

mode. Table 15 reports overall closed-loop score along with three challenging scenario subsets: *Nudge Around* (lateral interaction with adjacent agents), *High Traffic Density* (dense multi-agent scenarios), and *Jaywalk* (unexpected pedestrian crossings). HiPlan achieves the highest overall score (68.97), outperforming the strongest baseline (Flow Planner, 61.82) by $+7.2$ CLS, with the largest margins on High Traffic Density ($+13.4$) and Jaywalk ($+15.7$) – the scenarios that most stress reactive interaction with other agents. This is consistent with the thesis that hierarchical search becomes increasingly valuable as scenarios deviate from typical driving behavior.

Table 15: interPlan benchmark results (closed-loop score, reactive mode). HiPlan outperforms all baselines on the overall score and on every scenario subset. **Bold** indicates best.

Method	Overall	Nudge Around	High Traffic Density	Jaywalk
PlanTF (26)	47.70	49.40	58.85	33.94
PLUTO (w/o refinement) (3)	58.47	71.56	67.25	25.48
Diffusion Planner (39)	52.90	60.48	49.71	26.20
Flow Planner (30)	61.82	72.96	67.21	43.57
HiPlan (full)	**68.97**	**78.27**	**80.59**	**59.31**

E.2 Recovery Under Distribution Shift

Section 4.2 showed results on the effect of progressively removing U-turn occurrences during training to benchmark the effectiveness of the hierarchy evaluating plans over simply learning behavioral patterns. Table 16 extends this analysis to the full 2×2 matrix of WM and search structures, evaluated on unprotected left turns. We report three nuPlan metrics: collision rate (Coll., fraction of scenarios with at-fault collisions, $\downarrow$), time-to-collision score (TTC, the nuPlan TTC subscore in $[0, 1]$ measuring

Table 16: Robustness to distribution shift on unprotected left turns. We progressively remove unprotected left turn scenarios from training and evaluate on held-out left turn scenarios. Coll. (%) is the collision rate (fraction of scenarios with NC<1, $\downarrow$), TTC is the mean time-to-collision score ($\uparrow$), and EP is the mean ego progress ratio ($\uparrow$).

	Hier WM + Hier Search			**Flat WM + Hier Search**			**Hier WM + Flat Search**			**Flat WM + Flat Search**		
Training Data	Coll.↓	TTC↑	EP↑	Coll.↓	TTC↑	EP↑	Coll.↓	TTC↑	EP↑	Coll.↓	TTC↑	EP↑
100% left turns	**9**	**0.94**	**0.91**	10	0.92	0.89	11	0.91	0.88	12	0.89	0.86
50% left turns	**16**	**0.86**	**0.84**	24	0.78	0.79	26	0.76	0.77	33	0.69	0.71
25% left turns	**22**	**0.79**	**0.78**	31	0.70	0.72	33	0.68	0.70	40	0.61	0.64
0% left turns	**30**	**0.69**	**0.69**	45	0.56	0.59	42	0.57	0.60	53	0.47	0.51

Table 17: Per-component inference latency (ms) for the hierarchical pipeline on RTX 3090.

Component	Latency
Scene encoder	4 ± 1 ms
Anchor predictor	< 1 ms
Stage 1: Upper MPPI ($J_{\text{up}} = 4$, $N_{\text{up}} = 128$)	17 ± 2 ms
Stage 2: Flow generation ($N_\tau = 8$, $N_{\text{steps}} = 20$)	33 ± 5 ms
Stage 3: Lower verification ($N_\tau = 8$, $T_f = 80$)	15 ± 3 ms
Stage 4: Selection	< 1 ms
Total (hierarchical)	$\mathbf{69 \pm 3}$ **ms**
Flat MPPI (best achievable)	220 ± 20 ms

minimum predicted time-to-collision against surrounding agents, $\uparrow$), and ego progress ratio (EP, fraction of expert progress achieved before scenario end, $\uparrow$). Together these capture safety, near-miss margin, and task completion. The aligned hierarchical configuration degrades least across all three metrics as left-turn retention drops, and the gap between aligned and mismatched configurations widens at lower retention – consistent with the Section 4.2 finding that the advantage of the hierarchy is largest exactly when the base policy fails most. Both mismatched configurations underperform aligned-hierarchy, reinforcing the alignment ablation result (Section 4.5): hierarchy in either the world model or the search alone does not recover the performance of jointly hierarchical inference.

E.3 Inference Latency Breakdown

Section 4.3 reports search efficiency in MPPI evaluation count. Here we verify that this translates to wall-clock latency (Table 17). All measurements use a single NVIDIA RTX 3090 with batch size 1, timed via `torch.cuda.synchronize()` barriers. We report mean latency over 500 replans on Test14-hard after a 50-call warmup.

Stage 1 scores all N_{up} anchor candidates per iteration in a single forward pass, and Stage 3 batches across the N_τ trajectory candidates (sequential in time due to the recurrence). The hierarchical pipeline completes a full replan well within the 100 ms budget required for 10 Hz closed-loop planning. Flat MPPI at matched performance exceeds this budget.

E.4 World Model Dependence on Base Policy

The two-stage search procedure searches over residual corrections to the base policy's anchor predictions. This makes the system's performance dependent not only on the world model's accuracy but also on the quality of the base policy that anchors the search. To quantify this dependence, we replace the hierarchical base policy with two alternatives while keeping the world model, reward models, and search procedure unchanged: (i) a standard non-hierarchical diffusion policy (39) matched in parameter count, and (ii) a random policy that samples anchor sequences uniformly from the kinematically feasible space. Table 18 reports Test14 NR performance for each.

Replacing the hierarchical base policy with a flat diffusion policy drops performance by 16.0 CLS, and using a random policy drops it by 63.0 CLS. Two factors contribute to this dependence. First, search corrects *around* the base policy's anchors via residual perturbations; if those anchors are far from a good plan, the residual MPPI distribution must cover a much larger neighborhood to find one,

Table 18: World model dependence on base policy. Test14 NR CLS scores when the hierarchical base policy is replaced, holding the world model, reward models, and search fixed.

Base policy	Test14 NR
Hierarchical (ours)	**95.19**
Diffusion policy	79.15
Random policy	32.16

which is sample-inefficient at fixed compute. Second, the world and reward models are trained on data collected partly by the base policy. A weaker policy produces a narrower and less informative coverage of states, leaving the world model less accurate exactly where search needs it. The random policy result confirms that test-time search alone cannot compensate for an absent base policy: it requires meaningful proposals to refine.

This result also has practical implications. Even when the world model is held fixed, deploying with a stronger base policy reduces the search burden at inference since the planner has to correct less and can converge in fewer MPPI iterations.

E.5 Why Hierarchy Matters: Representation Quality

Subsampled flat WM ablation. Table 4 compares three sources for the upper-level representation, all evaluated with the identical two-stage search procedure of Algorithm 3:

- **Config A: Hierarchical upper WM.** The upper-level RSSM trained at the anchor timescale, with RM_{up} trained on its latents. This is the configuration used in the full system.
- **Config B: Flat WM (subsampled).** A flat WM trained at full resolution and rolled out at full resolution; latent states are read out at anchor timesteps and used as the upper-level representation. A matched reward model is trained on these subsampled latents using the same Wasserstein objective as RM_{up}.
- **Config C: Flat WM (coarse rollout).** A flat WM trained at full resolution but rolled out with skip-step transitions (one transition per anchor interval, predicting $t + \Delta_{\text{anchor}}$ directly from t). The same reward model architecture as Config B is trained on these coarse-rollout latents.

All three configurations share the base policy of the full system after five distillation rounds, which reaches 82.8 Test14-hard R without search. Config A improves on it by 4.1, Config B matches it, and Config C falls 2.7 below it, so coarse-rollout latents from a flat model actively mislead the search rather than merely failing to help it.

Latent Space Probing We detail the feature extraction procedure for the linear probing experiments in Section 4.4. All eight probe features are derived from nuPlan's standard scenario interface, requiring no manual annotation. Features are extracted at each timestep along ground-truth trajectories and used as targets for linear (and, for nonlinear probes, nonlinear) regression from the corresponding latent state.

We probe for the following high-level features:

- **Current lane identity.** The ego (x, y) position is queried against nuPlan's map API to retrieve the nearest lane centerline; the corresponding lane ID is used as a discrete classification target.
- **Maneuver class.** Each timestep is labeled with the maneuver type from nuPlan's scenario tags (e.g., lane change, unprotected left turn, U-turn, lane keep). Used as a discrete classification target.
- **Lateral offset from lane center.** The signed perpendicular distance from the ego position to the nearest lane centerline polyline, computed via nearest-point projection. Single continuous scalar.

- **Distance to goal along route.** nuPlan provides the planned route as a sequence of lane segments (`route_roadblock_ids`). The ego position is projected onto the route centerline polyline; the target is the remaining arc length to the route endpoint. Single continuous scalar.

We probe for the following low-level features:

- **Distance to nearest agent.** Euclidean distance from ego (x, y) to the nearest tracked agent, taken over all agents in nuPlan's `TrackedObjects` at the current timestep. Single continuous scalar.
- **Ego speed.** Magnitude of the ego velocity, $\sqrt{v_x^2 + v_y^2}$, taken directly from the ego state. Single continuous scalar.
- **Longitudinal jerk.** Computed by finite-differencing ego speed across three consecutive timesteps: $j_t = (v_{t+1} - 2v_t + v_{t-1})/\Delta t^2$ with $\Delta t = 0.1\,\text{s}$ at 10 Hz. Single continuous scalar.
- **Lateral acceleration.** Computed as $a_{\text{lat}} = v^2\kappa$, where v is ego speed and κ is the local path curvature estimated from three consecutive ego positions. Single continuous scalar.

The dissociation we observe reflects which features each level finds useful for prediction at its own timescale, rather than which feature its decoder reconstructs. The upper-level decoder is supervised on a mix of features that span both abstraction levels. Yet, linear readability tracks abstraction level, not decoder supervision. ego speed and agent distance are both in the decoder's reconstruction target but are linearly unreadable from the upper WM's latent ($R^2 < 0.22$). On the other hand, lane offset (which is also in the decoder target) is linearly readable ($R^2 \geq 0.78$). The upper WM exposes a subset of features useful to the anchor timescale, regardless of what the decoder asks it to reconstruct.

Probe training. For each (latent source, feature) pair, we train a linear probe on frozen latent states from the corresponding world model. Latents are extracted at every timestep for the upper WM (at anchor timesteps) and lower WM / flat WM (at all timesteps). Continuous features use ridge regression with L_2 regularization tuned on a held-out validation split; discrete features use multinomial logistic regression with the same regularization scheme. We report R^2 for continuous targets and accuracy converted to a comparable scale for discrete targets, following standard practice for probe analyses. Each probe is trained on ∼50K examples from the training split and evaluated on the validation split; reported R^2 values in Figure 4 are validation-set scores.

E.6 Alignment Ablation

Section 4.5 reports results for the 2×2 matrix crossing world model structure (flat vs. hierarchical) with search structure (flat vs. two-stage). The two "matched" configurations — flat WM with flat search, and hierarchical WM with two-stage search — follow standard implementations: dense MPPI over primitive actions for the former, and the procedure of Algorithm 3 for the latter. The two "mismatched" configurations require additional design choices, which we document here.

Flat WM with two-stage search. Two-stage search requires anchor-level latents to evaluate candidate anchor sequences during Stage 1 MPPI. Since the flat WM has no native anchor-level representation, we obtain anchor-level latents by rolling out the flat WM at full resolution conditioned on a dense trajectory consistent with the candidate anchors, then reading out the latent state at each anchor timestep. A reward model is trained on these subsampled latents using the same Wasserstein objective as RM_{up}. This uses the same latent readout as the Flat WM (subsampled) condition in Table 4, but unlike that experiment, the full training pipeline, including distillation, is rerun with this configuration, so the base policy is distilled from the flat planner's corrections. Stages 2–4 follow Algorithm 3 unchanged.

Hierarchical WM with flat search. Flat search runs MPPI over dense action sequences over the full planning horizon. To apply this to the hierarchical WM, we run the upper-level WM once on the base policy's anchors W_{base} to produce upper latents $z^{\text{up}}_{1:K}$, then run dense MPPI in the lower-level WM conditioned on these fixed upper latents (with anchor-boundary resets retained). Candidate dense

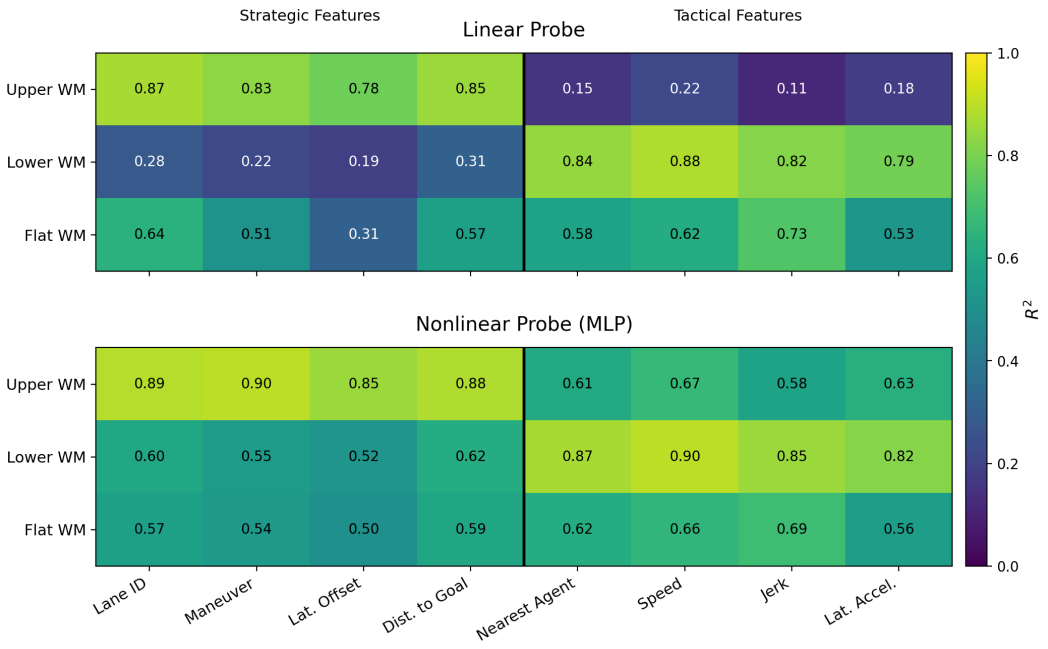


Figure 6: Linear (top) and nonlinear MLP (bottom) probe R^2 for upper WM, lower WM, and flat WM latents on high-level and low-level features. Upper and lower WM linear probes show a double dissociation; the gap narrows under nonlinear probing. Flat WM shows no such structure.

Table 19: Component ablations on Val14 NR and Test14-hard R. Each row removes a single component from the full system; Δ is the change relative to the full system. Ablations are grouped by system area.

	Val14 NR		Test14-hard R	
Configuration	Score	Δ	Score	Δ
Full system	**95.62**	–	**86.90**	–
Base policy				
Hierarchical base policy only	91.29	−4.33	72.62	−14.28
w/o anchor consistency loss	82.75	−12.87	68.29	−18.61
w/o curriculum training schedule	63.40	−32.22	60.11	−26.79
Hierarchical world model				
w/o anchor boundary reset	84.19	−11.43	80.15	−6.40
w/o cross-level conditioning	80.22	−15.40	77.31	−9.59
w/o hierarchical reward models	72.74	−22.88	69.90	−17.00
w/o joint training	70.37	−25.25	64.18	−22.72
Search and critic				
w/o critic	84.19	−11.43	80.48	−6.42
w/o buffer initialization	86.02	−9.60	79.22	−7.68

trajectories are scored using RM_{lo} over the full horizon. This configuration uses the hierarchical WM's full machinery but does not exploit the upper-level latent space for search.

E.7 Ablation Studies

We show that **every component of the system contributes measurably**, with the largest effects from curriculum scheduling, joint world model training, and the distillation iteration loop. Table 19 reports Val14 NR CLS with one component ablated at a time, grouped by system area.

Three ablations produce disproportionately large drops. The curriculum schedule is critical for the base policy: removing it causes the flow matching generator to overfit to ground-truth anchors and fail when conditioned on sampled anchors at inference. The joint world model fine-tuning is essential for the hierarchy: without it, the upper latent space loses the signal that forces it to encode information the lower level depends on, and the lower level effectively ignores it. The Phase 2-3 distillation loop matters substantially when iterated — a single round captures easy corrections but fails to propagate improvements to rarer failures (see Appendix E.9 for the full distillation iteration analysis).

Sensitivity to the number of anchors. The number of anchors K controls the granularity of the hierarchy. We sweep $K \in \{3, 5, 8, 10\}$ holding all other hyperparameters fixed, retraining the full pipeline at each setting. Table 20 reports Test14-hard reactive CLS.

Table 20: Sensitivity to the number of anchors K on Test14-hard R. Default $K = 5$.

K	3	5	8	10
Δ_{anchor} (steps)	26	16	10	8
Test14-hard R	81.4	**86.9**	85.2	83.7

Performance is best at $K = 5$ but remains within $\sim$2 CLS for $K \in \{5, 8\}$, indicating moderate robustness to anchor density. Performance degrades more substantially at the extremes: $K = 3$ provides too few anchors to capture strategic diversity over an 8-second horizon, while $K = 10$ pushes anchors close enough together that the lower-level segments shrink and the upper-level search dimensionality rises (from 10 to 20), reducing the efficiency advantage of two-stage search. The result supports our use of $K = 5$ as a reasonable default while showing that the framework does not require fine-tuned anchor density to function.

Longer planning horizons. Table 20 varies anchor density at a fixed horizon. To test whether the advantage of the hierarchy grows with horizon, we instead fix the anchor interval at $\Delta_{\text{anchor}} = 16$ steps and increase T_f, so that only the number of strategic decisions grows. We run this sweep on CARLA Longest6 v2, where routes run 1 to 2 km continuously, because nuPlan episodes are too short to execute horizons much beyond 8 s. Both systems decline modestly on the longer horizons, but flat search declines faster, and the gap widens from 7 DS at 8 s to 22 DS at 24 s (Table 21).

Table 21: Driving Score on CARLA Longest6 v2 as the planning horizon grows, with the anchor interval fixed at 16 steps.

T_f (steps)	K	HiPlan (hier. WM)	HiPlan (flat WM)
80 (8 s)	5	70	63
160 (16 s)	10	68	54
240 (24 s)	15	65	43

E.8 Correction Analysis

For every scenario on Test14-hard where the base policy fails (collision, off-route, or stalled progress) and the full system recovers, we run four ablated configurations of the search procedure and classify each recovery by which level was necessary. The four configurations are: (i) upper-level search only (anchor MPPI with no dense verification), (ii) lower-level search only (dense MPPI conditioned on the base policy's anchors), (iii) full two-stage search, and (iv) no search (base policy alone). A scenario is classified as an *upper-level catch* if (i) recovers but (ii) does not; *lower-level catch* if the reverse; *joint* if only (iii) recovers; and *indifferent* if both (i) and (ii) recover.

Upper-level catches occur when the base policy commits to a fundamentally incorrect plan and no amount of lower-level refinement suffices without a different anchor sequence (e.g., wrong lane choice at an intersection). Lower-level catches involve correct high-level intent but unsafe execution (e.g., correct decision to yield but unsafe deceleration profile). Joint cases require coordinated changes at both scales. Figure 7 supplements this with a scatter plot of anchor displacement $\|W^* - W_{\text{base}}\|$ against dense residual magnitude $\|\tau^* - \tau_{W^*}\|$, colored by recovery type. Upper-level corrections

Table 22: Distribution of recovery types on Test14-hard scenarios where the base policy fails and the full system recovers (N = 100 scenarios).

Recovery type	Fraction (%)
Upper-level catch (anchor change required)	40
Lower-level catch (refinement sufficient)	22
Joint (both levels required)	24
Indifferent (either level suffices)	14

cluster at high anchor displacement with low dense residual; lower-level corrections cluster at the opposite corner; joint corrections require large changes at both scales. The categories overlap substantially — driving failures are continuous, not categorical — but the overall trend confirms that the two search levels operate on geometrically distinct aspects of the trajectory.

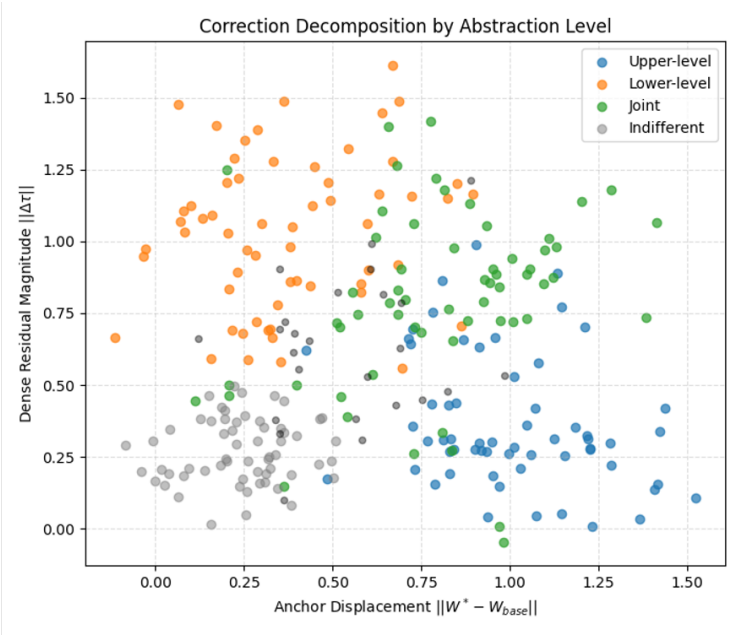


Figure 7: Decomposition of search corrections by abstraction level on Test14-hard. Each point is a recovered scenario, with anchor displacement on the x-axis and dense residual magnitude on the y-axis. Upper-level catches (blue) require large anchor changes with minimal dense residual; lower-level catches (orange) require small anchor changes with large dense residual; joint corrections (green) require both. Indifferent scenarios (gray) cluster near the origin where small changes at either level suffice.

E.9 Distillation and Training Dynamics

The training pipeline (Section 3.6) iterates Phases 2 and 3 for $N_{\text{iter}} = 5$ rounds, with each iteration deploying the planner, collecting corrections, and distilling them back into the base policy. We report two analyses: convergence of the iteration loop, and ablation of the distillation pipeline. The "HiPlan (base policy only)" row in Table 1 reports the iteration-1 base policy (trained on expert data alone, before any distillation) as the pre-distillation reference; below we track how the base policy evolves over subsequent iterations.

Distillation convergence across iterations. We track three metrics across the five Phase 2–3 iterations on Test14-hard R: (i) standalone CLS of the base policy at each iteration (without test-time search), (ii) full system CLS (with two-stage search) at each iteration, and (iii) the fraction of all Test14-hard scenarios in which the planner produces a non-trivial correction ($\|\Delta W\| > \epsilon_{\text{distill}}$, with

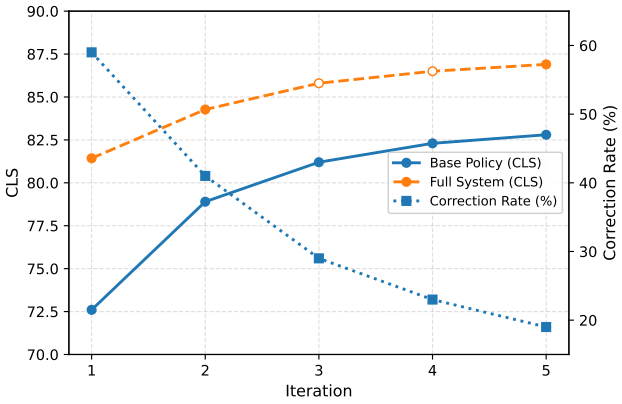


Figure 8: Distillation convergence across Phase 2–3 iterations on Test14-hard R. Base policy CLS (without search) rises from 72.6 to 82.8; full system CLS (with two-stage search) rises from 81.4 to 86.9; the fraction of scenarios in which the planner produces a non-trivial correction ($\|\Delta W\| > \epsilon_{\text{distill}}$) falls from 58% to 19%. The narrowing base-policy/full-system gap and the decreasing correction rate together indicate that distillation transfers planner corrections into the base policy, reducing the test-time search burden at deployment.

Table 23: Distillation ablation. Test14-hard R after training under three pipeline configurations. All configurations use the full two-stage search at inference; only the training pipeline differs.

Configuration	Test14-hard R
Full pipeline ($N_{\text{iter}} = 5$)	**86.90**
No distillation	81.43
Single-round distillation ($N_{\text{iter}} = 1$)	84.27

$\epsilon_{\text{distill}}$ held fixed across iterations). Figure 8 shows all three across iterations. Iteration k denotes the system after $k - 1$ rounds of distillation, so iteration 1 uses the undistilled base policy.

The base policy CLS rises monotonically as distilled corrections accumulate, from 72.6 at iteration 1 to 82.8 at iteration 5. The full system also improves, from 81.4 with the undistilled base policy to 86.9 at iteration 5, but more slowly, so the gap between them narrows from 8.8 CLS to 4.1 CLS. The fraction of scenarios requiring non-trivial corrections falls from 58% to 19% over the same range. Because $\epsilon_{\text{distill}}$ is fixed across iterations, this decrease reflects policy improvement rather than changes in reward-model sensitivity, and it occurs despite the reward model becoming more discriminative as the policy-expert gap narrows, which indicates that policy improvement outpaces reward sharpening.

The iterations therefore matter in two ways. The deployed system improves by 5.5 CLS, and the fraction of scenarios requiring non-trivial planner corrections falls by a factor of three, so in the converged system 81% of scenarios are handled directly by the base policy with negligible MPPI residual while search remains active for the harder 19%. The largest gains occur in iterations 1–2; marginal improvement after iteration 4 is small (below the noise floor across repeated training runs), motivating the choice of $N_{\text{iter}} = 5$.

Distillation ablation. We compare three training configurations to isolate the contributions of distillation and iteration. In all three, the full two-stage search is run at inference; what differs is whether and how planner corrections are distilled back into the base policy during training. (a) The full pipeline (Phase 0→1→2→3 with $N_{\text{iter}} = 5$). (b) No distillation (Phase 0→1→2): the planner is deployed during training to refine the world and reward models, but its corrections are never distilled into the base policy. (c) Single-round distillation (Phase 0→1→2→3 with $N_{\text{iter}} = 1$). Table 23 reports final Test14-hard R performance for each configuration. Configurations (b) and (c) correspond to iterations 1 and 2 of Figure 8.

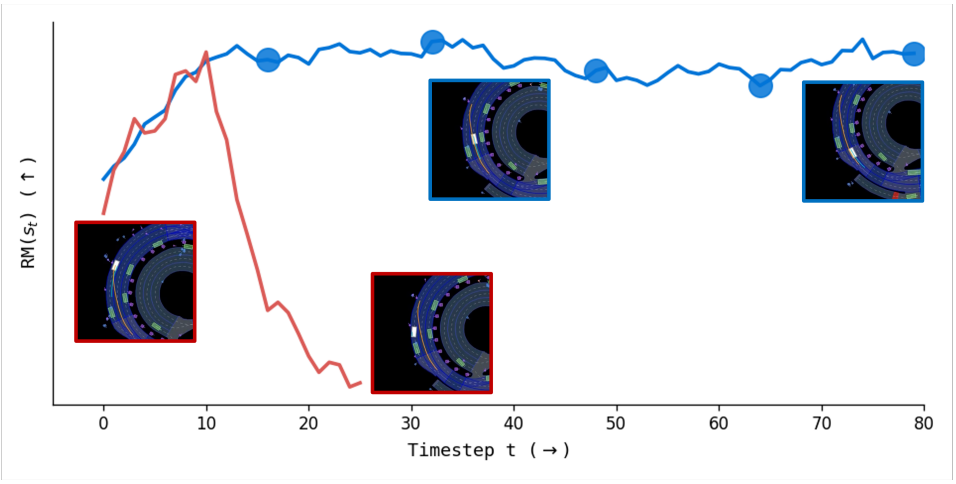


Figure 9: The original trajectory (red) causes the ego vehicle to stop behind a stationary vehicle on a turn. Similar behaviors have also been observed by (39) where the ego vehicle has difficulty executing lateral motion. HiPlan (blue) forces the ego vehicle to overcome this and change lanes beforehand in order to avoid getting stuck behind stationary vehicles. The red 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.

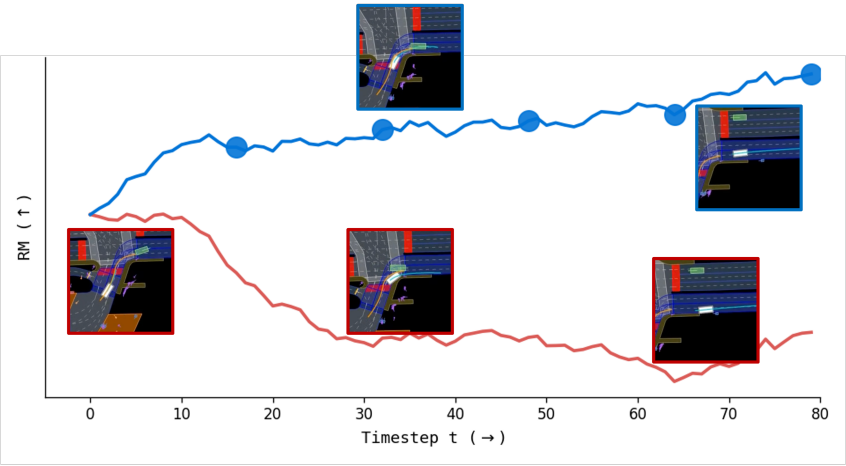


Figure 10: We see examples of two different trajectories rolled out from the same starting point. The base policy (red) results in the ego vehicle colliding with an incoming vehicle at ($t = 35$) and later passing a cyclist with an unsafe distance (at $t = 68$). HiPlan (blue) avoids this by initializing the ego vehicle with a slower speed from the very beginning, resulting in a higher reward, making it easier to stop and wait for incoming traffic, guiding the ego vehicle a safe distance away from the cyclist.

The full pipeline outperforms no distillation by 5.5, CLS, confirming that distilling planner corrections into the base policy improves the deployed system – the planner alone, without an improving base policy, is not sufficient. The full pipeline also outperforms single-round distillation by 2.6, CLS. A single round captures most of the easy corrections, but iterating beyond it provides additional gains, particularly on rarer failure modes that only become reachable after the policy has improved enough that the planner's corrections shift toward them.

E.10 Qualitative Analysis

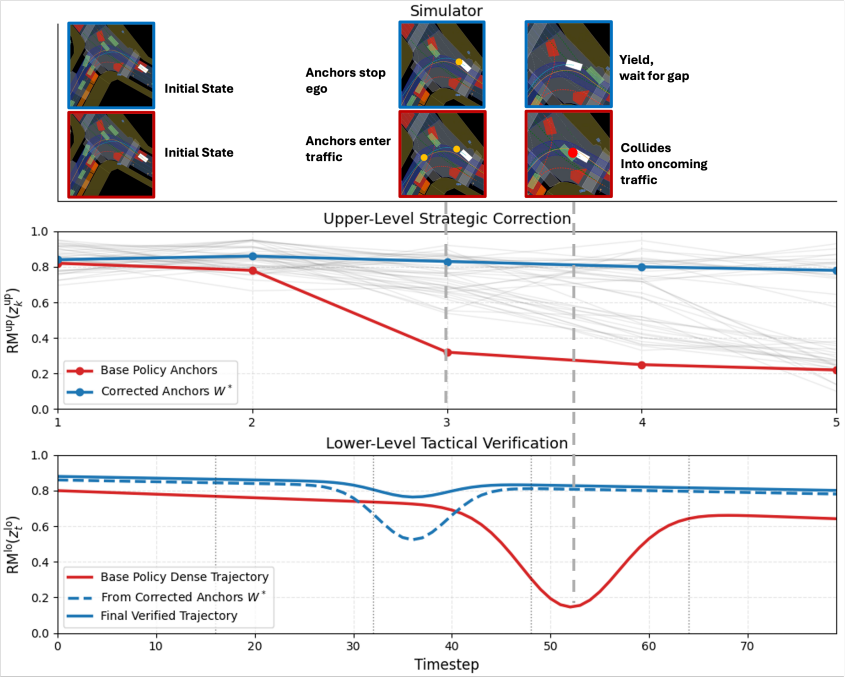


Figure 11: Recovery case study on an unprotected left turn. **Top:** BEV snapshots showing both methods at three moments. Both start from the same scene (left). At anchor 3 (middle), yellow dots show the predicted anchor waypoints: corrected anchors keep the ego stopped before the intersection, while base policy anchors lead into oncoming traffic. Executing those plans (right) produces a safe yield versus a collision. **Middle:** Upper-level reward RM^{up} on anchor latents. Base policy anchors collapse from anchor 3 onward as the strategic error compounds; corrected anchors maintain expert-like scores. Grey lines show other Test14-hard scenarios for context. **Bottom:** Lower-level reward RM^{lo} on dense latents. The base policy's dense trajectory collapses around timestep 50 as the unsafe plan executes (red); the trajectory regenerated from corrected anchors recovers strategically but passes through a tactically suboptimal region around timestep 35 (dashed blue); Stage 3 verification selects an alternative dense trajectory (solid blue) that maintains high scores throughout. Dashed vertical lines connect each BEV column to the corresponding moment in the reward panels.

Algorithm 3 Two-Stage Hierarchical Search

Require: Observation o_t, base policy π^{base}, upper WM, lower WM, RM_{up}, RM_{lo}, V_{up}
Ensure: Action a_t^* to execute

// Encode and generate base plan
$f_{\text{up}} \leftarrow \text{Encoder}(\mathcal{S}_t)$
$W^{\text{base}} \leftarrow \text{AnchorPredictor}(f_{\text{up}})$
$z_0^{\text{up}} \leftarrow \text{encode_upper}(o_t)$

// Stage 1: Upper-level MPPI
$\mu \leftarrow \mathbf{0}$, $\sigma \leftarrow \sigma_{\text{init}}$
for $j = 1$ to J_{up} **do**
  for $n = 1$ to N_{up} **do**
    Sample $\Delta W^n \sim \mathcal{N}(\mu, \sigma)$
    $W^n \leftarrow W^{\text{base}} + \Delta W^n$
    Roll out: $\{z_{1:K}^{\text{up},n}\} \leftarrow \text{upper_WM}(z_0^{\text{up}}, W^n)$
    $Q^n \leftarrow \sum_{k=1}^{K} \gamma^k \cdot \text{RM}_{\text{up}}(z_k^{\text{up},n}) + \gamma^K \cdot V_{\text{up}}^{\text{inf}}(z_K^{\text{up},n})$
  end for
  $w^n \leftarrow \exp(Q^n/\tau_{\text{up}}) \;/\; \sum_m \exp(Q^m/\tau_{\text{up}}) \quad \forall n$
  $\mu \leftarrow \sum_n w^n \Delta W^n$
  $\sigma \leftarrow \sqrt{\sum_n w^n (\Delta W^n - \mu)^2}$ (element-wise)
end for
$W^* \leftarrow W^{\text{base}} + \mu$
$z_{1:K}^{\text{up},*} \leftarrow \text{upper_WM}(z_0^{\text{up}}, W^*)$ (store for lower conditioning)

// Stage 2: Dense trajectory generation
for $n = 1$ to N_τ **do**
  Sample prior noise $\epsilon^n \sim \mathcal{N}(0, I)$
  $\tau^n \leftarrow \text{FlowMatching}(W^*, f_{\text{up}}, \epsilon^n)$
end for

// Stage 3: Lower-level verification
$\mathcal{T}_{\text{valid}} \leftarrow \emptyset$
for $n = 1$ to N_τ **do**
  for $k = 1$ to K **do**
    $h_{t_k}^{\text{lo}} \leftarrow \text{MLP}_{\text{init}}(s_k^{\text{up},*})$ (reset at anchor boundary)
  end for
  Roll out: $\{z_{1:T_f}^{\text{lo},n}\} \leftarrow \text{lower_WM}(\tau^n, z_{1:K}^{\text{up},*})$
  $R_{\text{lo}}^n \leftarrow \sum_t \gamma^t \cdot \text{RM}_{\text{lo}}(z_t^{\text{lo},n})$
  if $\min_t \text{RM}_{\text{lo}}(z_t^{\text{lo},n}) \geq R_{\text{threshold}}$ **then**
    $\mathcal{T}_{\text{valid}} \leftarrow \mathcal{T}_{\text{valid}} \cup \{n\}$
  end if
end for

// Stage 4: Final selection
if $\mathcal{T}_{\text{valid}} = \emptyset$ **then**
  $n^* \leftarrow \arg\min_n \; v_{\max}(\tau^n)$ (fallback: most conservative)
else
  $R_{\text{up}}^n \leftarrow \sum_k \gamma^k \cdot \text{RM}_{\text{up}}(z_k^{\text{up},*}) \quad \forall n \in \mathcal{T}_{\text{valid}}$
  $n^* \leftarrow \arg\max_{n \in \mathcal{T}_{\text{valid}}} \; R_{\text{up}}^n + \alpha \cdot R_{\text{lo}}^n$
end if
$a_t^* \leftarrow \tau^{n^*}[1]$
return a_t^* (replan at next timestep)