LEARNING WHEN TO SEARCH: ADAPTIVE TEST-TIME COMPUTATION FOR GENERATIVE MOTION PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In autonomous driving, generative motion planners often use candidate generation and scoring to improve the final trajectory, but running this search process at every planning step introduces additional computation. In this study, we investigate when search is actually needed. Our analysis shows that the benefit of search varies greatly across planning steps and is related to both the current planning information and recent history. We therefore develop a lightweight history-aware model to decide whether search should be performed after the base trajectory is generated. If search is needed, the original search module generates and evaluates additional candidates; otherwise, the base trajectory is used directly. Closed-loop experiments on nuPlan show that our method achieves an average score of 86.07 with search on 56.81% of planning steps, comparable to Full Search at 85.64 with 100% calls. It also performs better than fixed search schedules at similar search rates and reduces mean planning inference time by 32.0%.

1 INTRODUCTION

Motion planning is a core component of autonomous driving. Traditional planners usually generate multiple candidate trajectories and evaluate them according to safety, efficiency, and other driving objectives (Caesar et al., 2021; Dauner et al., 2023). With the development of deep learning, learning-based planners have been widely studied to directly predict future trajectories from scene observations. More recently, diffusion, flow-based, and other generative models have been used to generate multiple possible trajectories and have achieved strong performance in closed-loop driving (Zheng et al., 2025; Tan et al., 2025; Wang et al., 2025; Tang et al., 2026).

In many planning systems, trajectory generation is only one part of the planning process. Before execution, the generated trajectory may be further compared with other candidates or evaluated by additional scoring functions. Some methods generate multiple trajectories and select one according to learned or rule-based scores. Others construct new candidates around the model output or further optimize the trajectory at test time (Dauner et al., 2023; Yang et al., 2024; Li et al., 2025). Although the detailed implementations are different, these methods all use additional computation to improve the final planning result.

In this work, we focus on a common two-stage setting. A base planner first produces a trajectory, and an additional candidate-generation and scoring process is then used to find a better one. In this paper, we simply call this process *search*. Search can improve the final trajectory, but it also requires extra computation. In most implementations, it is performed at every planning step, regardless of whether the base trajectory already performs well.

However, the benefit of search can be very different across planning steps. During simple lane following, the traffic situation may remain stable and the base planner may already provide a good trajectory. In this case, generating and evaluating more candidates may bring little improvement. During turning, merging, or strong interactions with surrounding vehicles, search may instead find a much better trajectory. The quality of the base trajectory can also change as the traffic situation changes. Therefore, search may not be needed at every planning step. This leads to a simple question: *when should we search?*

We first examine how much search can improve the base trajectory at different planning steps. The results show that the benefit of search is highly uneven. On the development set, the top 20% of

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.

planning steps account for 78.3% of the total positive improvement measured by the trajectory scorer. We also vary the search frequency in closed-loop simulation. More search generally improves planning performance, but the additional improvement becomes smaller as the search rate increases, and the relation is not strictly monotonic. These results show that simply reducing the search frequency is not enough. We also need to choose the right planning steps for search.

We then study whether these planning steps can be identified before search is performed. We find that search benefit is related to information already available during planning, including the route ahead and the trajectory produced by the base planner. Search benefit also shows a clear short-term pattern over time. When search is useful at the current step, it is more likely to remain useful in the following steps. Similarly, planning steps with little search benefit often appear continuously for a short period. These results show that both the current planning information and recent history can help decide when to search.

Based on these observations, we develop a lightweight history-aware model to decide whether search should be performed. At each planning step, the base planner first generates a trajectory as usual. Our model then uses the current planning information and a short history to estimate whether search is likely to improve the trajectory. If search is needed, the original search module is used; otherwise, the base trajectory is used directly. The base planner and the search module remain unchanged, and the decision is made before search is performed.

We evaluate the proposed method in closed-loop simulation on nuPlan. Across four evaluation settings, our method achieves an average score of 86.07 with search on 56.81% of planning steps, comparable to Full Search at 85.64 with a 100% search rate. At a similar search rate, it performs better than fixed search schedules. The lower search rate reduces the mean planning inference time from 258.7 to 176.0 ms. We further apply the same method to diffusion-, flow-, and autoregressive planners, showing that the approach can be used with different types of generative planners.

The main contributions of this study are summarized as follows:

1. We study how the benefit of search changes across planning steps. The results show that search benefit varies greatly over time, is related to current planning information, and has a clear short-term temporal pattern.
2. We develop a lightweight history-aware model to decide when to search. The model uses information available before search and does not change the base planner or the search module.
3. We evaluate adaptive search in closed-loop driving. The results show that our method performs better than fixed search schedules at similar search rates, while reducing both search calls and planning time.

2 Related Work

We review three lines of work related to our study: generative motion planning, trajectory search and scoring, and adaptive test-time computation.

2.1 Generative Motion Planning

Generative models have been widely used in motion planning because they can represent multiple possible future trajectories. Diffusion-based planners generate trajectories through a denoising process and can further use guidance to adjust planning objectives (Janner et al., 2022; Zheng et al., 2025). Recent studies have improved their inference speed and planning performance through truncated sampling, better trajectory representations, and interaction modeling (Liao et al., 2025; Tan et al., 2025; Wang et al., 2025).

Flow-based models provide another way to generate diverse trajectories with fewer inference steps (Tan et al., 2025; Wang et al., 2025). Autoregressive methods have also been used to generate driving trajectories or motion tokens step by step (Tang et al., 2026). Although these methods use different generative models, they share the goal of producing high-quality trajectories from scene observations. Our work does not change the trajectory generator itself. Instead, we study the additional computation used after the base trajectory has been generated.

2.2 Search and Scoring in Motion Planning

Search and trajectory evaluation have long been used in motion planning. A common approach is to generate multiple candidate trajectories and evaluate them according to safety, progress, comfort, and other driving objectives. PDM-based planning generates route-conditioned candidates and compares them using several planning metrics (Dauner et al., 2023). Tree-based methods further organize candidate trajectories over multiple stages and use learned or designed costs to evaluate different choices (Huang et al., 2024).

Recent generative planners also combine learned trajectory generation with candidate evaluation. Some methods use generative models to produce or modify candidate trajectories and then select better candidates through explicit scoring or optimization (Zheng et al., 2025; Wang et al., 2025; Yang et al., 2024). Other studies improve the scoring model itself so that different trajectory candidates can be compared more accurately (Li et al., 2024; 2025).

Most of these studies focus on how to generate better candidates, how to evaluate them, or how to improve the final searched trajectory. In contrast, we study when the search process should be used. The search module itself remains unchanged.

2.3 Adaptive Test-Time Computation

Adaptive computation aims to use different amounts of computation for different inputs or different stages of inference. In diffusion models, existing methods reduce computation by adapting network depth, width, or token computation according to the current input or denoising state (Tang et al., 2025; Zhao et al., 2025). Similar ideas have also been studied in test-time reasoning, where more computation is assigned to inputs that are expected to benefit more from additional inference (Snell et al., 2025; Damani et al., 2025).

Adaptive computation has also been studied in sequential decision making. Some methods decide when an existing plan should be regenerated as the environment changes (Zhou et al., 2023), while others adjust the amount of computation used inside prediction or planning modules (Zhou et al., 2024; Chen et al., 2025; Tian et al., 2025). Among these works, AdaDrive (Zhang et al., 2025) is closely related to ours, as it learns when to activate an expensive LLM branch in a slow-fast driving system.

In contrast, our base planner runs at every planning step, and we learn when to invoke candidate search and scoring after the base trajectory has been generated. Our work therefore focuses on adaptive search rather than changing the internal computation of the base planner.

3 Analysis of Test-Time Search

Before introducing our method, we first examine how search affects planning. We use the 1,118-scenario Val14 development set for this analysis and keep Test14 for the final evaluation in Section 5.

3.1 Benefit of Search

At each planning step, the base planner first generates a trajectory. The search module then generates and scores additional candidates. We compare the score of the base trajectory with the highest score found by search, and define their difference as the local search gain:

$$g_t = S_t^{\text{search}} - S_t^{\text{base}}, \tag{1}$$

where S_t^{base} is the score of the base trajectory and S_t^{search} is the highest score among the searched candidates. A larger g_t indicates a better candidate found by search; a small g_t indicates that the base trajectory is already close to the best searched candidate.

As shown in Fig. 1, the benefit of search is not evenly distributed across planning steps. After ranking all planning steps by their local search gain, the top 20% account for 78.3% of the total positive gain. Identifying more useful steps may therefore retain much of the search benefit with fewer calls.

3.2 Search Frequency

We next examine closed-loop performance under less frequent search. We use fixed search schedules with different search rates, while keeping the base planner, candidate generation, and scoring process unchanged.

As shown in Fig. 2, increasing the search rate generally improves closed-loop performance. The score increases from 89.79 without search to 93.61 at a search rate of about 60%. However, using more search gives smaller improvements at higher search rates. The 80% schedule obtains a score of 93.53, while full search reaches 94.17. The measured relation is therefore not strictly monotonic. Search rate alone therefore cannot fully explain planning performance: a fixed schedule ignores the current planning state and may miss steps where additional search is useful.

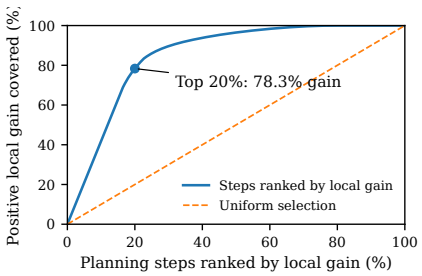


Figure 1: Local search benefit across planning steps. The steps are ranked by local search gain, and the curve shows the cumulative share of positive gain.

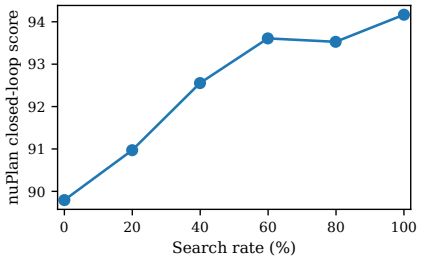


Figure 2: Closed-loop performance under different fixed search rates. Search generally improves performance, while the improvement becomes smaller at higher search rates.

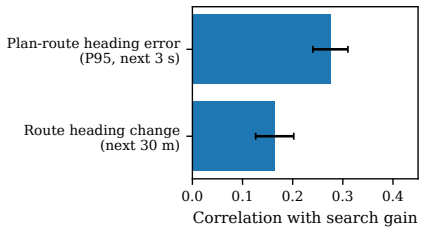
3.3 Planning Context and History

We next examine whether search benefit is related to pre-search information about the route, surrounding traffic, and generated trajectory.

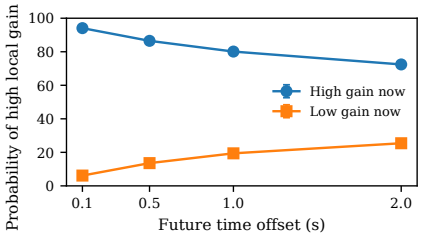
The results show that search benefit changes with several planning variables. Fig. 3(a) gives two examples. Larger route heading changes are associated with higher local search gain. A larger heading difference between the base trajectory and the route shows an even stronger relation. Their Spearman correlations with local search gain are 0.164 and 0.277, respectively. These relations provide useful clues about search benefit.

Search benefit also shows a clear pattern over time. We define a high-gain step as $g_t > 0.05$ and examine whether high-gain steps continue in the following planning steps. When the current step has high gain, the probability of another high-gain step after 0.1 s is 94.1%. In contrast, this probability is only 6.1% when the current step has low gain. The difference becomes smaller over longer time intervals, but remains visible up to 2 s, as shown in Fig. 3(b). This temporal pattern shows that recent planning history can provide useful information for deciding whether search is needed at the current step.

These observations motivate using the current planning state and a short history to decide whether search should be performed.



(a) Search gain and planning context.



(b) Temporal pattern of search gain.

Figure 3: Planning context and recent history both contain information related to search benefit.

4 METHOD

4.1 FRAMEWORK

We consider a two-stage planning system with a base planner and an additional search module. At each planning step, the base planner first generates a trajectory. Our model then uses the current planning information and recent history to decide whether additional search is needed.

If search is not needed, the base trajectory is used directly. Otherwise, the search module generates additional candidates, scores them, and returns the selected trajectory for execution. The final trajectory at planning step t is

$$\tau_t = \begin{cases} \tau_t^{\text{search}}, & a_t = 1, \\ \tau_t^{\text{base}}, & a_t = 0, \end{cases} \tag{2}$$

where a_t denotes the search decision.

Both the base planner and the search module remain unchanged. Our model only decides whether the additional search process should be used. The decision is made before search is performed, so it only uses information already available from the base planning process. After the selected trajectory is executed, the new planning state is added to the recent history and used for the next decision. Fig. 4 shows the online process.

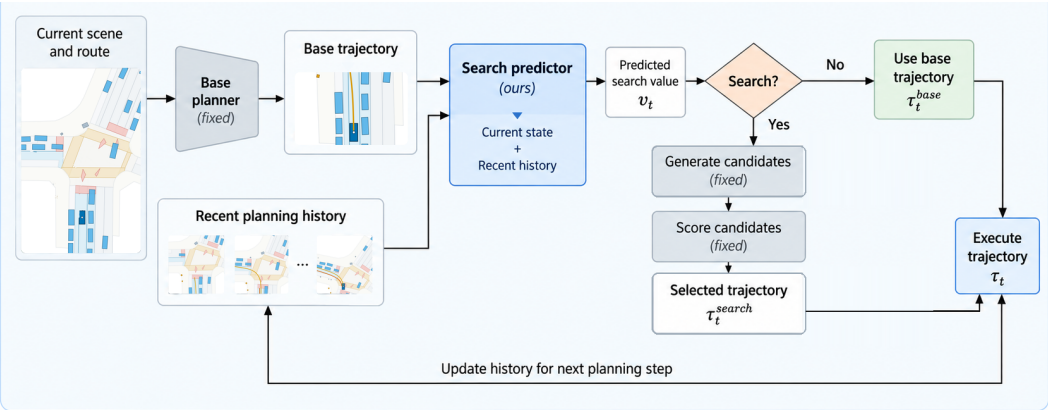


Figure 4: Online inference of adaptive search. The base planner runs at every planning step, while the predictor decides whether the additional search module should be used.

4.2 SEARCH PREDICTION MODEL

Section 3 shows that search benefit is related to both the current planning state and recent history. We therefore use these two types of information to estimate whether search is useful.

At planning step t, we construct a feature vector x_t from information available before search. The features describe the surrounding traffic, the route ahead, and the trajectory generated by the base planner. We use the most recent W planning steps,

$$X_t = [x_{t-W+1}, \ldots, x_t], \tag{3}$$

as the input to the search predictor.

A lightweight temporal encoder first processes the recent history. The encoded feature is then passed to a small prediction network. Instead of directly predicting a binary search decision, the model estimates the score of the base trajectory and how the score may change after search. These predictions are combined using the same scoring rule as the search module to obtain a search value v_t. A larger v_t means that additional search is expected to provide a larger improvement.

In our implementation, each step provides 25 features, and $W = 10$ covers the current step and the preceding 0.9 s. We use MiniRocket (Dempster et al., 2021) followed by a two-hidden-layer MLP. The feature list and network settings are given in Appendix A.

4.3 TRAINING

We train the search predictor using paired results collected offline. For each recorded planning state, we first run the base planner and obtain the base trajectory. We then run the complete search module from the same state and obtain the searched trajectory. Both trajectories are evaluated using the same scoring function. This gives paired base and searched results for training, as shown in Fig. 5.

The score difference between the two results tells us whether search improves the current plan. However, using only the final score provides limited information about how the trajectory changes. We therefore also use the individual score components as training targets. These components describe different planning objectives, including collision avoidance, time to collision, drivable-area compliance, driving direction, progress, comfort, and speed-limit compliance. The predictor estimates the base component scores and their changes after search.

We train score and magnitude predictions with Smooth-L1 losses and improvement and degradation predictions with class-weighted asymmetric binary cross-entropy. Appendix A.2.4–A.2.5 gives the complete targets, losses, and optimization settings.

During inference, the predicted base scores and score changes are combined using the same fixed PDM scoring rule as the search module. The resulting search value is

$$v_t = A(\hat{p}_t + \hat{\Delta}_t) - A(\hat{p}_t) - 1.5\hat{d}_t, \tag{4}$$

where $A(\cdot)$ is the fixed score aggregation, $\hat{p}_t$ denotes the predicted base component scores, $\hat{\Delta}_t$ their predicted changes after search, and $\hat{d}_t$ the predicted probability of non-progress degradation. Search is requested when $v_t > \delta$. We set $\delta = 0.0933$ on the development set and keep it fixed during testing.

During the first nine steps, search runs only at steps 4 and 8. After warm-up, a call-control rule reduces repeated search during long runs of positive requests. It only removes requested calls. Appendix A.2.5 gives the exact rule, and Section 5.5 evaluates its effect.

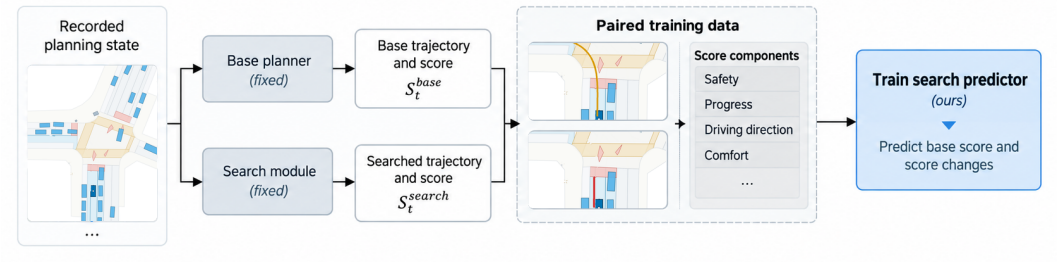


Figure 5: Offline training of the search predictor. Paired base and searched results from the same planning states provide component-score supervision for the prediction model.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate the proposed method in nuPlan closed-loop simulation (Caesar et al., 2021). We use Test14-random with 261 scenarios and Test14-hard with 272 scenarios, each with non-reactive (NR) and reactive (R) agents. These four settings cover different levels of planning difficulty and interaction.

We compare our method with three main search settings. **No Search** directly uses the trajectory generated by the base planner. **Full Search** runs the complete search process at every planning step. **Periodic Search** runs search at fixed rates of 20%, 40%, 60%, and 80%, without using the current planning state. All settings use Diffusion Planner (Zheng et al., 2025) as the base planner and the same PDM-based search module (Dauner et al., 2023).

We report the official nuPlan closed-loop score and the search rate. The search rate is the percentage of planning steps in which the search module is executed after the history input becomes available. All model parameters and decision thresholds are selected on the development set and kept fixed during testing.

5.2 Planning Performance

Table 1 presents the main closed-loop results. Full Search improves the average score of Diffusion Planner from 79.04 to 85.64, showing that additional search provides a clear benefit over the base planner. Our method achieves an average score of 86.07 with search on 56.81% of planning steps, comparable to Full Search while using substantially fewer calls. It also performs better than fixed schedules at similar search budgets.

We further compare our method with fixed schedules in Fig. 6. In the non-reactive settings, it scores 94.59 on Random-NR and 79.89 on Hard-NR, above the tested schedules at similar rates. In the reactive settings, it scores 90.46 on Random-R at 54.31% calls and 79.36 on Hard-R at 59.49%; the latter exceeds Periodic-60 by 2.81 points.

Across all four settings, our method improves on Periodic-60 by 1.76 points while using 3.13 percentage points fewer calls. This comparison shows that choosing the planning steps for search is more effective than applying a fixed schedule at a similar average rate.

5.3 Search Allocation Analysis

We next examine where the search computation is used. We score base and searched candidates offline at every state visited by the final policy, including skipped steps, without changing the recorded decisions. Across the four settings, 53.13%–60.30% of steps capture 66.06%–70.77% of positive local gain. Uniform selection has expected coverage equal to its search rate, and mean gain is higher on searched than skipped steps in all four settings (Fig. 7(a)).

Fig. 7(b) directly groups the same states by offline-computed local search gain. Across the four settings, search is triggered on 69.1%–76.4% of states in the highest-gain quintile, compared with 54.1%–60.2% in the lowest-gain quintile. The highest-gain states therefore receive substantially more search than the lowest-gain states.

We also group Test14 states into context quintiles, from Q1 (lowest 20%) to Q5 (highest 20%). Search rates increase by 15.8–26.7 percentage points across route-heading-change quintiles and by 34.9–45.8 points across plan–route-heading-difference quintiles. The final decisions thus follow the contextual trends in Section 3.3 (Fig. 7(c,d)).

Table 1: Closed-loop planning results with Diffusion Planner. All scores are on a 0–100 scale. ↑: higher is better; ↓: lower is better.

Search policy	Random-NR	Random-R	Hard-NR	Hard-R	Avg. ↑	Calls (%) ↓
No search	89.19	83.19	75.39	68.38	79.04	0.00
Periodic-20	90.00	86.87	77.66	72.02	81.64	19.98
Periodic-40	92.37	88.18	77.75	74.69	83.25	39.96
Periodic-60	92.96	90.57	77.20	76.54	84.32	59.94
Periodic-80	92.64	91.11	77.41	78.80	84.99	79.91
Full search	92.95	92.11	77.16	80.33	85.64	100.00
Ours	**94.59**	90.46	**79.89**	79.36	**86.07**	56.81

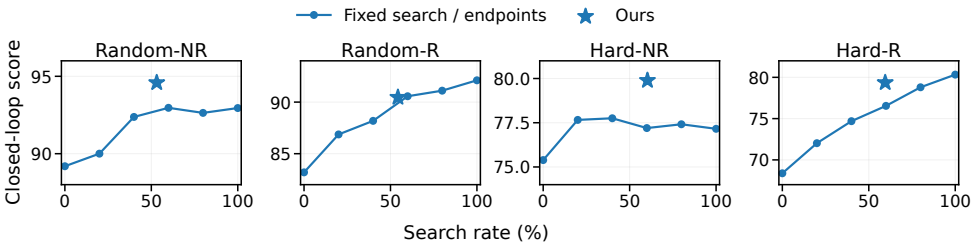


Figure 6: Closed-loop performance under different search rates in the four Test14 settings. Connected points show fixed search schedules together with the no-search and full-search endpoints, while stars show our adaptive policy.

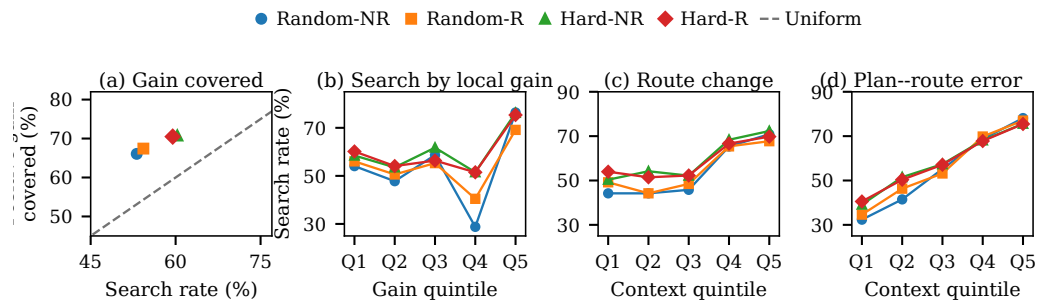


Figure 7: Search allocation in the final closed-loop policy. (a) Positive local gain covered at the realized search rate. (b) Actual search rate across local-gain quintiles. (c,d) Search rate across context quintiles. Q1 and Q5 denote the lowest and highest 20%, respectively.

We further examine a driving sequence to show how search decisions change with the driving situation. In Fig. 8(A), the vehicle is driving straight without a lead vehicle, and the model uses the base trajectory directly. As the vehicle starts and continues a left turn through traffic, the model calls search at the steps shown in Fig. 8(B,C). The gain curve, computed offline, also shows larger score improvements from search at these two turning steps.

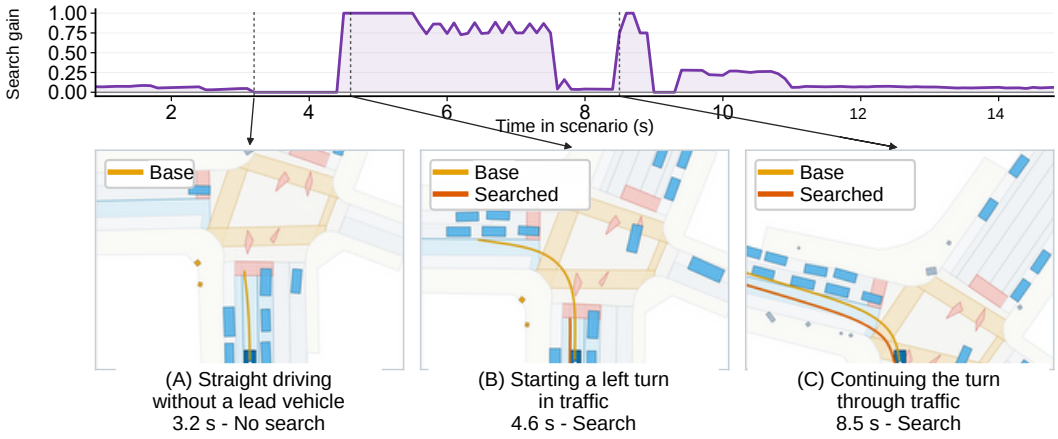


Figure 8: Search decisions in one driving sequence. The gain trace is computed offline. Search is skipped during straight driving (A) and used during a left turn (B,C).

5.4 Generalization to Other Planners

We further apply the same framework to Flow Planner (Tan et al., 2025), FlowDrive (Wang et al., 2025), and Plan-R1 (Tang et al., 2026), which use different trajectory generators. At about 56%–60% calls, Flow Planner scores 83.89 versus 85.65 for Full Search, FlowDrive scores 85.13 versus 85.35, and Plan-R1 improves from 84.27 without search to 86.35, versus 87.12 for Full Search (Table 2).

These results show that adaptive search is not limited to one trajectory generator. The same idea can be used with diffusion-based, flow-based, and autoregressive planners when they share a similar candidate search and scoring process.

Table 2: Average closed-loop score and search rate across different base planners. ↑: higher is better; ↓: lower is better.

Base planner	No search	Full search	Ours ↑	Calls (%) ↓
Diffusion Planner	79.04	85.64	86.07	56.81
Flow Planner	77.25	85.65	83.89	56.46
FlowDrive	82.31	85.35	85.13	59.01
Plan-R1	84.27	87.12	86.35	60.28

5.5 Ablation Study

Table 3: Closed-loop ablation of structured supervision and temporal history. Overall-only uses aggregate PDM-score supervision; Current-only repeats the current feature vector over the temporal input window. ↑: higher is better; ↓: lower is better.

Variant	Random-NR	Random-R	Hard-NR	Hard-R	Avg. ↑	Calls (%) ↓
Overall-only + W10	93.96	90.42	78.02	79.23	85.41	62.37
Component + Current-only	92.92	89.88	77.61	78.16	84.64	58.66
Ours: Component + W10	**94.59**	**90.46**	**79.89**	**79.36**	**86.07**	**56.81**

Planning history. Using recent history improves the average closed-loop score from 84.64 to 86.07, while slightly reducing the search rate from 58.66% to 56.81%. This result is consistent with the temporal pattern observed in Section 3.3 and shows that recent planning history is useful for the final search decision.

Training supervision. Using component-level supervision improves the average score from 85.41 to 86.07 and reduces the search rate from 62.37% to 56.81%.

Repeated-call control. Keeping the predictor and decision threshold unchanged, executing the first eight consecutive positive requests and then four of every five reduces the average search rate from 65.55% to 56.81%. The average score changes from 86.51 to 86.07, showing that repeated calls can be reduced with only a small change in planning performance.

5.6 Runtime Analysis

We measure planning time from preprocessed input to the returned trajectory on a single NVIDIA RTX 4090 GPU. This includes base trajectory generation, the adaptive search decision, and any executed search, but excludes preprocessing and logging.

As shown in Table 4, No Search takes 60.13 ms per step and Full Search takes 258.73 ms. Our method takes 176.02 ms, including 4.53 ms for prediction and decision, reducing planning time by 32.0%. The prediction cost is therefore much smaller than the computation saved by skipping search.

Table 4: Planning inference time. Times are milliseconds per post-warm-up planning step; preprocessing and logging are excluded. ↓: lower is better.

Policy	Search rate (%)	Predictor (ms)	Planning time (ms) ↓	Speedup
No search	0.00	0.00	60.13	–
Full search	100.00	0.00	258.73	1.00×
Ours	56.81	4.53	176.02	1.47×

6 Conclusion

In this study, we develop an adaptive search method for generative motion planning. A lightweight history-aware model decides whether additional search should be performed after the base trajectory is generated. When search is skipped, the base trajectory is used directly; otherwise, the original search module generates and evaluates additional candidates. Closed-loop experiments show that our method reduces search calls and planning time while maintaining strong planning performance. It also performs better than fixed search schedules at similar search rates and can be applied to different generative planners.

In the future, we plan to extend this work in two directions. First, we will study more flexible search control. The current method decides whether search should be used, while future work can further adjust the number of candidates or the search budget according to the planning situation. Second, we will evaluate the method with more planning systems and more complex closed-loop driving scenarios, especially highly interactive and long-tail situations, to further examine its generalization and robustness.

AI USE DISCLOSURE

In this work, we used generative AI tools for engineering code implementation. We did not use generative AI tools to develop the research ideas, design or provide feedback on the experimental methodology, or interpret the experimental results; other required-disclosure tasks were not applicable to this work. Additionally, we used generative AI tools for language polishing to improve readability. We reviewed all AI-assisted work, and all AI-assisted code was checked and validated by the authors. We take responsibility for the final content of this work, including the method, experimental evidence, text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

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

Yuan Chen, Zi-han Ding, Ziqin Wang, Yan Wang, Lijun Zhang, and Si Liu. Asynchronous large language model enhanced planner for autonomous driving. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 22–38, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-72764-1. doi: 10.1007/978-3-031-72764-1_2.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 102783–102802, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ff414825df833edb8b1839e3d5d495e9-Paper-Conference.pdf.

Daniel Dauner, Marcel Hallgarten, Andreas Geiger, and Kashyap Chitta. Parting with misconceptions about learning-based vehicle motion planning. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1268–1281. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/dauner23a.html.

Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. MiniRocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 248–257, New York, 2021. ACM. doi: 10.1145/3447548.3467231.

Zhiyu Huang, Peter Karkus, Boris Ivanovic, Yuxiao Chen, Marco Pavone, and Chen Lv. DTPP: Differentiable joint conditional prediction and cost evaluation for tree policy planning in autonomous driving. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6806–6812, 2024. doi: 10.1109/ICRA57147.2024.10610550.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9902–9915. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/janner22a.html.

Zhenxin Li, Kailin Li, Shihao Wang, Shiyi Lan, Zhiding Yu, Yishen Ji, Zhiqi Li, Ziyue Zhu, Jan Kautz, Zuxuan Wu, Yu-Gang Jiang, and Jose M. Alvarez. Hydra-MDP: End-to-end multimodal planning with multi-target Hydra-Distillation. arXiv preprint arXiv:2406.06978, 2024. URL https://arxiv.org/abs/2406.06978.

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

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b623663fd9b874366f3ce019fdfdd44-Paper-Conference.pdf.

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. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 38310–38335. Curran Associates, Inc., 2025. doi: 10.52202/085713-1281. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/36d1e8aa9ceec3b781682bf5e63c31bf-Paper-Conference.pdf.

Shengkun Tang, Yaqing Wang, Caiwen Ding, Yi Liang, Yao Li, and Dongkuan Xu. AdaDiff: Accelerating diffusion models through step-wise adaptive computation. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 73–90, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-72986-7. doi: 10.1007/978-3-031-72986-7_5.

Xiaolong Tang, Meina Kan, Shiguang Shan, and Xilin CHEN. Plan-R1: Safe and feasible trajectory planning as language modeling. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 23715–23735, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/28a80012e6f564c0cc8e7661e1db83fe-Paper-Conference.pdf.

Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Yang Wang, Zhiyong Zhao, Kun Zhan, Peng Jia, XianPeng Lang, and Hang Zhao. DriveVLM: The convergence of autonomous driving and large vision-language models. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4698–4726. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/tian25c.html.

Lingguang Wang, Ömer Şahin Taş, Marlon Steiner, and Christoph Stiller. FlowDrive: moderated flow matching with data balancing for trajectory planning. *arXiv preprint arXiv:2509.21961*, 2025. URL https://arxiv.org/abs/2509.21961.

Brian Yang, Huangyuan Su, Nikolaos Gkanatsios, Tsung-Wei Ke, Ayush Jain, Jeff Schneider, and Katerina Fragkiadaki. Diffusion-ES: Gradient-free planning with diffusion for autonomous and instruction-guided driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15342–15353, June 2024.

Ruifei Zhang, Junlin Xie, Wei Zhang, Weikai Chen, Xiao Tan, Xiang Wan, and Guanbin Li. AdaDrive: Self-adaptive slow-fast system for language-grounded autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5112–5121, October 2025.

Wangbo Zhao, Yizeng Han, Jiasheng Tang, Kai Wang, Yibing Song, Gao Huang, Fan Wang, and Yang You. Dynamic diffusion transformer. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 65520–65552, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a44a70acd5d0abc1a252ada9719dd06d-Paper-Conference.pdf.

Yinan Zheng, Ruiming Liang, Kexin ZHENG, Jinliang Zheng, Liyuan Mao, Jianxiong Li, Weihao Gu, Rui Ai, Shengbo Li, Xianyuan Zhan, and Jingjing Liu. Diffusion-based planning for autonomous driving with flexible guidance. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 37207–37227, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5c0cd43063f087eb32e572d040a19cd2-Paper-Conference.pdf.

Siyuan Zhou, Yilun Du, Shun Zhang, Mengdi Xu, Yikang Shen, Wei Xiao, Dit-Yan Yeung, and Chuang Gan. Adaptive online replanning with diffusion models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 44000–44016. Curran Associates, Inc., 2023. doi: 10.52202/075280-1909. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/893a5db6100028ec814cfd99fe92c31b-Paper-Conference.pdf.

Yang Zhou, Hao Shao, Letian Wang, Steven L. Waslander, Hongsheng Li, and Yu Liu. SmartRefine: A scenario-adaptive refinement framework for efficient motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15281–15290, June 2024.

A Additional Implementation Details

A.1 Input Features

The search predictor uses 25 scalar factors at each planning step. Here P95 denotes the 95th percentile, and dynamics percentiles other than speed are computed from absolute values. Table 5 groups them by their source and role.

Table 5: Input factors used by the search predictor. Percentiles are computed from the base-plan trajectory over the stated horizon.

Category	**Factors**
Environment	valid neighbor count; static-object count; lane count; route-lane count; nearest-agent distance; agents within 10/20/40 m; red-light flag; yellow-light flag; traffic-light count
Route	branch count; route-ahead length; heading change over 30 m; mean/max absolute curvature over 30 m
Base plan / route	3 s plan–route lateral P95/max; 3 s plan–route heading P95
Base-plan dynamics	4 s mean speed; max/P95 curvature; lateral-acceleration P95; steering P95; curvature-rate P95

A.2 Search Prediction Model

The search predictor estimates the quality of the base trajectory and the component-level changes that search may produce. These estimates are combined using the search module's scoring rule to obtain a search value. This section describes the temporal representation, the meaning of each output, the search-value calculation, and the supervision used to train the model.

A.2.1 Temporal Representation and Network Architecture

At planning step t, the base planner first generates its trajectory. The available environment, route, and base-plan information then provide the feature vector $x_t \in \mathbb{R}^{25}$ described in Table 5. The predictor uses the ten most recent feature vectors, ordered from oldest to newest:

$$X_t = [x_{t-9}, \ldots, x_t] \in \mathbb{R}^{25\times 10}, \qquad z_t = \Phi(\widetilde{X}_t) \in \mathbb{R}^{2436}. \tag{5}$$

Here $\widetilde{X}_t$ replaces non-finite entries with stored channel-specific fill values, and Φ is the frozen MiniRocket transform. Both the transform and the fill values remain fixed during deployment. At a planning frequency of 10 Hz, the window spans the current observation and the preceding 0.9 s. It includes steps with and without search, but contains neither future observations nor the current step's search result.

The temporal representation is passed through a shared two-layer multilayer perceptron and a linear output layer:

$$\begin{aligned} h_t^{(1)} &= \mathrm{SiLU}(\mathrm{LN}_{\gamma,\beta}(W_1 z_t + b_1)), & h_t^{(1)} &\in \mathbb{R}^{256}, \\ h_t^{(2)} &= \mathrm{SiLU}(W_2 h_t^{(1)} + b_2), & h_t^{(2)} &\in \mathbb{R}^{128}, \\ o_t &= W_o h_t^{(2)} + b_o, & o_t &\in \mathbb{R}^{36}. \end{aligned} \tag{6}$$

Layer normalization computes the mean and population variance over the 256 hidden coordinates, with affine parameters γ, β and numerical stabilizer $\varepsilon_{\mathrm{LN}} = 10^{-5}$. The activation is $\mathrm{SiLU}(u) = u\sigma(u)$, where σ denotes the sigmoid function. Including the affine normalization parameters, the trainable parameter count is

$$\begin{aligned} &(2436 \cdot 256 + 256) + 2 \cdot 256 \\ &\quad + (256 \cdot 128 + 128) + (128 \cdot 36 + 36) = 661{,}924. \end{aligned} \tag{7}$$

The frozen temporal transform is excluded from this count.

A.2.2 PREDICTION OUTPUTS

The prediction targets use seven PDM score components in the order

$$p = (p_{\mathrm{NC}}, p_{\mathrm{TTC}}, p_{\mathrm{DAC}}, p_{\mathrm{DDC}}, p_{\mathrm{P}}, p_{\mathrm{C}}, p_{\mathrm{SL}}), \tag{8}$$

corresponding to no collision, time to collision, drivable-area compliance, driving direction, progress, comfort, and speed-limit compliance. We denote the six non-progress components by $\mathcal{S} = \{\mathrm{NC}, \mathrm{TTC}, \mathrm{DAC}, \mathrm{DDC}, \mathrm{C}, \mathrm{SL}\}$.

The first 35 coordinates of o_t form the task prediction vector

$$q_t = (u_t^p, u_t^\Delta, \ell_t^S, m_t^S, \ell_t^d, \ell_t^P, m_t^P, u_t^{\mathrm{ord}}), \tag{9}$$

with respective dimensions $(7, 7, 6, 6, 1, 2, 2, 4)$. Each group has a distinct role:

- u_t^p predicts the seven base-trajectory component scores in standardized coordinates. The auxiliary output u_t^Δ predicts standardized component changes, which can be positive or negative.
- ℓ_t^S gives improvement logits for the six non-progress components, and m_t^S gives the corresponding magnitude outputs. The separate logit ℓ_t^d predicts whether any non-progress component deteriorates.
- ℓ_t^P and m_t^P give two event logits and two magnitude outputs for progress increase and decrease, respectively.
- u_t^{ord} gives four logits for the auxiliary ordinal classification task defined below.

The implementation retains one additional scalar in the 36-dimensional output layer. It is included in the parameter count but receives no task supervision and is not used to compute the deployed search value. The direct component-change and ordinal outputs are also used only for training. Deployment reconstructs component changes from the event and magnitude outputs as described next.

A.2.3 COMPONENT RECONSTRUCTION AND SEARCH VALUE

The base-score output is first mapped back to the original score space using the stored training-set statistics:

$$\hat{p}_{t,j} = \mu_j^p + s_j^p u_{t,j}^p, \qquad s_j^p = \max\{\mathrm{std}_{\mathrm{train}}(p_j), 0.03\}. \tag{10}$$

Here μ_j^p is the training mean of component j. The scale floor prevents components with very small training variance from receiving excessively large normalization factors. Thus $\hat{p}_t$ estimates the quality of the current base trajectory before search.

For each non-progress component, the predictor separately estimates whether search improves the component and the magnitude of that improvement. With $\sigma(u) = (1 + e^{-u})^{-1}$ and $\mathrm{sp}(u) = \log(1 + e^u)$, the reconstructed improvement is

$$\widehat{\Delta}_{t,j} = \sigma(\ell_{t,j}^S)\,\mathrm{sp}(m_{t,j}^S), \qquad j \in \mathcal{S}. \tag{11}$$

The sigmoid term estimates the probability of improvement, while the softplus term gives its positive magnitude conditional on improvement. Their product gives the component's predicted improvement. Progress is modeled in both directions:

$$\widehat{\Delta}_{t,\mathrm{P}} = \sigma(\ell_{t,+}^P)\,\mathrm{sp}(m_{t,+}^P) - \sigma(\ell_{t,-}^P)\,\mathrm{sp}(m_{t,-}^P). \tag{12}$$

The resulting vector $\widehat{\Delta}_t$ is added to $\hat{p}_t$ to form the component estimates used to score search. Non-progress deterioration is represented separately by

$$\widehat{d}_t = \sigma(\ell_t^d), \tag{13}$$

the predicted probability that at least one component in $\mathcal{S}$ deteriorates. Consequently, the decoder does not subtract a separate predicted deterioration magnitude from each non-progress component; it accounts for this risk in the search value below.

The component estimates are aggregated using the same fixed scoring rule as the search module. Let $\mathcal{C}(y)$ clip the progress coordinate to $[-2, 1]$ and each other coordinate to $[0, 1]$. For $c = \mathcal{C}(y)$, define

$$A(y) = c_{\mathrm{NC}} c_{\mathrm{DAC}} c_{\mathrm{DDC}} c_{\mathrm{SL}} \frac{5c_{\mathrm{P}} + 5c_{\mathrm{TTC}} + 10c_{\mathrm{C}}}{20}. \tag{14}$$

The multiplicative components gate the weighted combination of progress, TTC, and comfort. The difference between the aggregated search and base predictions measures the predicted gain from search. Subtracting a non-progress degradation penalty gives

$$v_t = A(\hat{p}_t + \widehat{\Delta}_t) - A(\hat{p}_t) - 1.5\widehat{d}_t. \tag{15}$$

The coefficient 1.5 is the fixed degradation penalty used in deployment. Each evaluation of A clips its input independently: in particular, $\hat{p}_t + \widehat{\Delta}_t$ is formed before clipping. The predictor issues a desired search request when

$$b_t = \mathbf{1}\{v_t > \delta\}, \qquad \delta = 0.09331828355789185. \tag{16}$$

This threshold is the full-precision value corresponding to 0.0933 in Section 4. The distinction between the desired request b_t and the executed action a_t is specified in the deployment details below.

A.2.4 Offline Supervision and Training Objective

Training pairs the base and searched trajectories evaluated from the same recorded planning state. For sample i, let their component scores be p_i and s_i, respectively, and define $\Delta_i = s_i - p_i$. These targets describe the local effect of one search invocation on the candidate trajectory. They are not obtained by comparing complete future closed-loop rollouts under always-search and never-search policies.

Event and magnitude targets. We use $\varepsilon_{\mathrm{event}} = 10^{-6}$ to distinguish improvements and degradations from negligible numerical differences. This event tolerance is separate from the layer-normalization stabilizer $\varepsilon_{\mathrm{LN}}$. The binary targets are

$$\begin{aligned} y^S_{ij} &= \mathbf{1}\{\Delta_{ij} > \varepsilon_{\mathrm{event}}\}, && j \in \mathcal{S}, \\ y^d_i &= \mathbf{1}\{\min_{j \in \mathcal{S}} \Delta_{ij} < -\varepsilon_{\mathrm{event}}\}, \\ y^P_{i,+} &= \mathbf{1}\{\Delta_{i,\mathrm{P}} > \varepsilon_{\mathrm{event}}\}, \qquad y^P_{i,-} = \mathbf{1}\{\Delta_{i,\mathrm{P}} < -\varepsilon_{\mathrm{event}}\}. \end{aligned} \tag{17}$$

Writing $[u]_+ = \max(u, 0)$, the magnitude targets are $[\Delta_{ij}]_+$ for non-progress improvements and $[\Delta_{i,\mathrm{P}}]_+$ and $[-\Delta_{i,\mathrm{P}}]_+$ for progress increase and decrease. A magnitude loss is evaluated only where its corresponding event label is positive; an empty positive-event mask contributes zero.

Regression and event losses. Base scores and direct component changes are standardized separately using their per-component training means and standard deviations, with the same 0.03 scale floor as in Eq. (10). The statistics remain fixed on the calibration and test sets. Both regression tasks use mean Smooth-L1 losses. The softplus-transformed magnitude predictions also use Smooth-L1 losses, evaluated in the original component-score space on positive events. All Smooth-L1 terms use transition parameter 1.

For event logit ℓ, binary target y, and positive-class weight w, the asymmetric binary loss is

$$B_{w,\gamma}(\ell, y) = -wy \log \sigma(\ell) - (1 - y)\sigma(\ell)^\gamma \log(1 - \sigma(\ell)). \tag{18}$$

The negative term downweights easy negative examples, while the positive term is weighted to address class imbalance. We use $\gamma = 2$ for non-progress improvement events and $\gamma = 1$ for degradation and progress events. If π denotes the training prevalence of the corresponding event, the positive-class weights are

$$\begin{aligned} w_{S,P} &= \mathrm{clip}\left(\frac{1 - \pi}{\max(\pi, 10^{-5})}, 1, 20\right), \\ w_d &= \min\left(30, \frac{1 - \pi}{\max(\pi, 10^{-5})}\right). \end{aligned} \tag{19}$$

Auxiliary ordinal supervision. The four-class target summarizes component changes according to a fixed priority. Any non-progress degradation assigns class 0. Otherwise, an improvement in NC, DAC, or DDC assigns class 3; if none occurs, an improvement in TTC, comfort, or speed-limit compliance assigns class 2. A progress increase with no non-progress improvement assigns class 1, and all remaining samples receive class 0. Improvements and degradations use the event tolerance defined above. The class-weighted cross-entropy uses weight $\mathrm{clip}(N/[4\max(n_k,1)],0.2,15)$ for class k, where N is the number of training samples and n_k its class count.

The overall training objective is

$$\begin{aligned} L = {} & 0.35L_p + 0.35L_\Delta + L_S + L_d + 0.5L_P \\ & + 0.6L_{mS} + 0.6L_{mP} + 0.35L_{\mathrm{ord}}. \end{aligned} \tag{20}$$

Here L_p and L_Δ are the base-score and direct component-change regression losses. The event losses L_S, L_d, and L_P average Eq. (18) over non-progress improvement, non-progress degradation, and progress events, respectively. The terms L_{mS} and L_{mP} average the corresponding positive-event magnitude losses, and L_{ord} is the auxiliary ordinal cross-entropy. Thus training supervises trajectory quality, improvement events and magnitudes, and degradation risk separately. The deployed search value is computed from these predictions rather than supervised by a binary search label, a recorded total-score regression target, or a scalar ranking loss.

A.2.5 Optimization and Reproducibility Details

Training uses AdamW for 30 epochs with learning rate 2×10^{-4}, weight decay 10^{-4}, and gradient-norm clipping at 5. Each epoch contains $\lceil N/2048\rceil$ updates. Each update independently samples $\min(2048,N)$ examples without replacement within that batch; examples may recur across updates. The deployed model was trained on 606 scenarios with random seed 20260912.

The threshold in Eq. (16) is the 40th percentile of the calibration-set search values, corresponding to a nominal 60% desired-request rate. It remains fixed during testing. This calibration target does not constrain the actual test-time search rate, which also depends on the encountered states and the call-control rule in Section 4.

Deployment details. Each scenario starts with an empty feature history and a reset request counter. Indexing steps from $t=1$, warm-up uses $a_t=\mathbf{1}\{t \bmod 4=0\}$ for $1\le t\le 9$, so search runs only at steps 4 and 8. These calls do not update the consecutive-request counter. From step 10 onward, the predictor runs at every step, including steps at which a call is suppressed. Let r_t count consecutive positive desired requests, initialized by $r_9=0$: a positive b_t increments the counter and a negative b_t resets it to zero. The controller executes all of the first eight requests in a run and suppresses requests numbered $9,14,19,\ldots$ thereafter, equivalently those with $r_t\ge 9$ and $(r_t-9) \bmod 5=0$. Suppression does not reset the counter. Hence $a_t\le b_t$ after warm-up, and the controller cannot create a request.

When $a_t=0$, the base trajectory is executed directly; otherwise, the existing search module returns its selected trajectory. The evaluated candidate configuration includes the base anchor, route proposals, and anchor expansions with lateral offsets of -1 and $+1$ m. Current searched component scores provide offline supervision only and are never inputs to the request predictor at that step.