AUTOAGENT0: AN AGENTIC RUNTIME FOR SAFE CLOSED-LOOP DRIVING

Anonymous authors
Paper under double-blind review

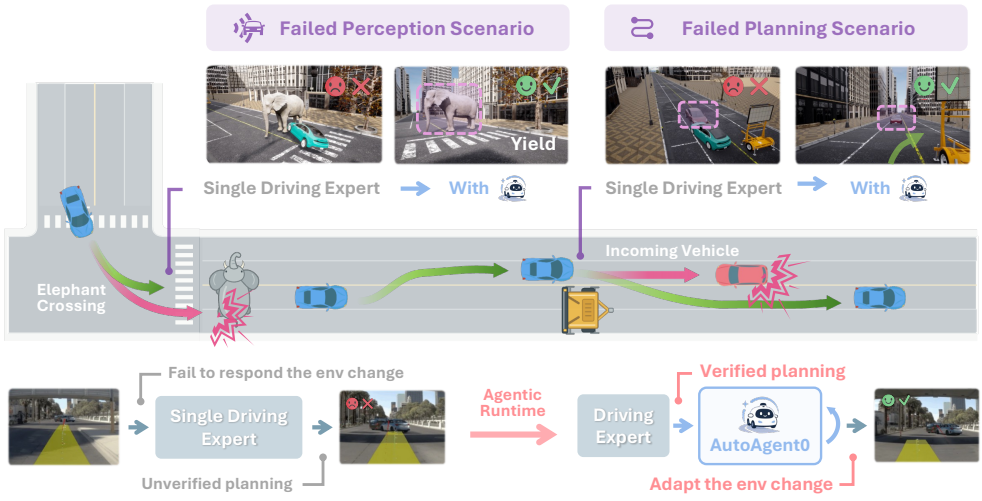


Figure 1: **Motivation and overview.** Inaccurate scene context and limited generalization to off-nominal states motivate runtime supervision. AUTOAGENT0 combines scene refinement, trajectory verification, and recovery planning. Top: a Fail2Drive case illustrates yielding and bypass recovery. Bottom: a NavSafe-∞ example compares standalone and AUTOAGENT0-augmented expert.

ABSTRACT

Safe autonomous driving requires more than accurate planning at a single frame: it demands reliable closed-loop execution in dynamic environments. Although end-to-end driving policies have advanced rapidly, the accumulated closed-loop errors, especially in unseen scenarios, can push vehicles into states from which the same policy struggles to recover. We address the challenge by treating closed-loop driving safety as a test-time execution problem. We introduce **AUTOAGENT0**[1], an agentic runtime and harness framework that augments existing driving experts with complementary detection skills, rule-based action verification, and feedback-driven recovery. To address inaccurate scene perception, an independent detection module constructs scene context using complementary perception skills, and a rule-based verifier checks proposed trajectories against this context before execution. When the nominal trajectories are unreliable, the orchestrator of the runtime activates **Recovery Planning**, where an agentic loop constructs alternative action proposals from action primitives and iteratively refines them using verification feedback. This design combines independent scene assessment with feasibility checks and online recovery, addressing both perception errors and the limited recovery capabilities of individual experts. Evaluations on Fail2Drive and NavSafe-∞ demonstrate improved closed-loop robustness over single-expert execution in long-tail and safety-critical scenarios. These results demonstrate that runtime supervision can improve closed-loop robustness across multiple driving experts without retraining the underlying policies.

[1] https://autoagent0.github.io/AutoAgent0

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.

1 INTRODUCTION

Autonomous driving, a core application of physical AI (Agarwal et al., 2026; Ali et al., 2025), operates in dynamic, open-ended environments and imposes stringent requirements for closed-loop safety and generalizability. The field has progressed from modular pipelines (Li et al., 2025b; Liu et al., 2025a) toward end-to-end driving systems (Hu et al., 2023; Zhou et al., 2025b;c). Nevertheless, safe closed-loop driving remains challenging: real-world scenarios follow a long-tailed distribution that cannot be exhaustively covered during training, while small decision errors can compound through interaction and eventually lead to safety-critical failures.

The dominant paradigm trains a single expert policy to directly produce actions across diverse scenarios (Li et al., 2024b; Xu et al., 2026). Despite this progress, such policies often generalize poorly to unfamiliar long-tail events in closed-loop driving. (Jia et al., 2024; Zhao et al., 2026). The accumulated errors may leave the vehicle stuck and unable to be recovered by the same policy that caused the deviation. For example, Fail2Drive (Gerstenecker et al., 2026) reports an average performance drop of 16.3% across seven driving models under long-tail closed-loop evaluation. Moreover, single learning-based expert remain largely opaque, making their decisions difficult to trust across diverse long-tail scenarios (Kim et al., 2026). These limitations raise a central question: *Should safe long-tail driving rely solely on an increasingly capable expert policy, or should safety also be addressed at through explicit runtime supervision test-time?*

Beyond generating high-quality actions, safe closed-loop driving requires a system to detect failures, propose alternatives, verify their feasibility, and recover online (Karkus et al., 2025). Recent agentic systems have demonstrated strong closed-loop execution capabilities by coordinating specialized skills, inspecting execution traces, analyzing failures, and recovering through feedback (Anthropic, 2025; Liu et al., 2025b; Zhang et al., 2026). This progress motivates an agentic approach to driving that combines the reasoning capabilities of vision language models (VLMs) with the basic driving capabilities of existing experts. Rather than relying on a single expert in all situations, a runtime can orchestrate these complementary capabilities, verify proposed actions, and initiate recovery when the expert fails to maintain safe progress under changing scene conditions.

To this end, we introduce **AUTOAGENT0**, an agentic runtime for safe closed-loop driving, as illustrated in Figure 2. AUTOAGENT0 is motivated by two sources of failure in end-to-end driving: inaccurate scene perception and limited capability to recover when single-expert execution breaks down, as shown in Figure 1. Our AUTOAGENT0 addresses these limitations through a runtime that integrates complementary detection skills, rule-based action verification, and feedback-driven recovery. First, for robust and interpretable verification before execution, an independent detection module constructs scene context using complementary detection tools. Second, a rule-based verifier checks action feasibility given the detected scene. When the nominal trajectory from the existing driving policy is deemed unreliable, the orchestrator activates **Recovery Planning**, where an agentic loop then constructs alternative maneuvers from action primitives and iteratively refines them using feedback from the rule-based verifier to restore feasible driving behavior.

We evaluate AUTOAGENT0 on challenging closed-loop benchmarks, including Fail2Drive (Gerstenecker et al., 2026) and NavSafe-∞ (Bao et al., 2026), with an emphasis on long-tail scenarios. The results demonstrate that runtime verification and feedback-driven recovery substantially improve robustness over single-expert execution. Our contributions are summarized as follows:

- We introduce **AUTOAGENT0**, an agentic runtime that augments existing policy with complementary detection skills, rule-based action verification, and feedback-driven recovery.
- We introduce an independent rule-based verifier for each trajectory before execution, while the **Recovery Planning** constructs and iteratively refines recovery maneuvers from action primitives when the nominal trajectory from the existing policy is unreliable.
- We evaluate AUTOAGENT0 on Fail2Drive and NavSafe-∞, demonstrating improved closed-loop robustness over single-expert execution in long-tail scenarios.

2 RELATED WORK

Closed-Loop Autonomous Driving Long-tail scenarios remain challenging for autonomous driving because rare interactions are underrepresented during training but can dominate closed-loop

safety. Recent approaches include structured planners such as UniAD and VAD (Hu et al., 2023; Jiang et al., 2023), multimodal and diffusion planners such as Hydra-MDP, DiffusionDrive, and SparseDriveV2 (Li et al., 2024a; Liao et al., 2025; Sun et al., 2026), and vision-language-action policies or world model policies such as AutoVLA and DriveVLA-W0 (Zhou et al., 2025a; 2026; Li et al., 2026c). BridgeSim studies the open-loop to closed-loop gap, while Fail2Drive demonstrates persistent degradation on matched long-tail routes (Zhao et al., 2026; Gerstenecker et al., 2026). Bench2Drive provides additional interactive evaluations (Jia et al., 2024). World Engine improves the expert through post-training on reconstructed critical interactions (Li et al., 2026a). AUTOAGENT0 instead addresses long-tail failures at runtime without retraining the expert.

Closed-Loop Runtime Monitoring Runtime supervision combines evidence for intervention with a corrective response. RSS defines safe-distance rules and proper responses (Shalev-Shwartz et al., 2017), while predictive safety filters optimize corrective controls under modeled dynamics and safety constraints (Wabersich & Zeilinger, 2021). Fallback-Safe MPC couples a calibrated perception monitor with planning that maintains fallback feasibility (Sinha et al., 2023); SODA-MPC uses conformally calibrated ensemble uncertainty to switch to reachability-based control (Contreras et al., 2025). Recent studies further investigate language-model-assisted intervention selection (Sinha et al., 2024) and plan repair under perception failures (Chakraborty et al., 2024). Building on these approaches, AUTOAGENT0 integrates scene-evidence refinement with parameterized recovery around a frozen driving expert. Scene refinement updates the verifier's inputs, while recovery proposals are revised using verifier and execution feedback.

Agentic Runtime Harnesses Agentic runtimes coordinate tools, retain interaction state, and revise decisions using environmental feedback. ReAct and Toolformer establish reasoning-and-action and tool-use interfaces (Yao et al., 2023; Schick et al., 2023), while AutoGen and SWE-agent provide programmable execution harnesses (Wu et al., 2024; Yang et al., 2024). Embodied systems such as Voyager, ENPIRE, and ASPIRE use execution feedback to explore and refine reusable skills (Wang et al., 2024; Xiao et al., 2026; Lu et al., 2026), and Agent-Driver applies tools, memory, and reasoning to autonomous driving (Mao et al., 2024). Our harness applies these tool-use and memory mechanisms to driving recovery, with a rule-based verifier determining which proposed actions may reach the controller.

3 AUTOAGENT0: A RUNTIME HARNESS FOR CLOSED-LOOP DRIVING

3.1 PROBLEM SETTING AND CLOSED-LOOP RUNTIME FORMULATION

Problem Setting. We formulate end-to-end closed-loop driving as a route-conditioned POMDP process following prior works (Gerstenecker et al., 2026; Jia et al., 2024). The original trajectory planner defines the default Nominal Policy π_{N}, using learning-based or rule-based trajectory planners. At decision point t, this policy generates finite-horizon trajectory proposals $\tau \sim \pi_{\mathrm{N}}(\cdot \mid h_t, q)$. Here $h_t = (o_0, \tau_0, \ldots, o_t)$ denotes the history of observations and previously executed trajectories, with o_t the current observation. The objective is to navigate to the goal region specified by the route instruction q while maintaining safe execution throughout the closed-loop interaction.

Limitations of Nominal Planning. End-to-end nominal policies face two challenges in closed-loop driving. *(1) Inaccurate scene context.* Their internal representations may miss safety-critical actors or mischaracterize drivable space, leading to unsafe trajectory proposals. *(2) Limited generalization to off-nominal states.* Unexpected events or accumulated execution errors may lead to states poorly represented in training, where the policy may repeatedly propose unsafe trajectories or fail to make progress. These challenges motivate a runtime that refines scene context to support trajectory verification and invokes recovery planning when rejecting nominal proposals alone cannot restore safe progress.

Closed-loop Runtime Formulation. A closed-loop runtime surrounds the nominal policy with a safety-enforcement layer comprising two complementary mechanisms (Sinha et al., 2023; Wabersich & Zeilinger, 2021). *(1) Proposal verification in nominal planning.* Given runtime context c_t, comprising the observation history, route instruction, and runtime feedback, a verifier V checks nominal proposals against specified safety criteria. We denote the verified nominal policy by π_{N}^{V},

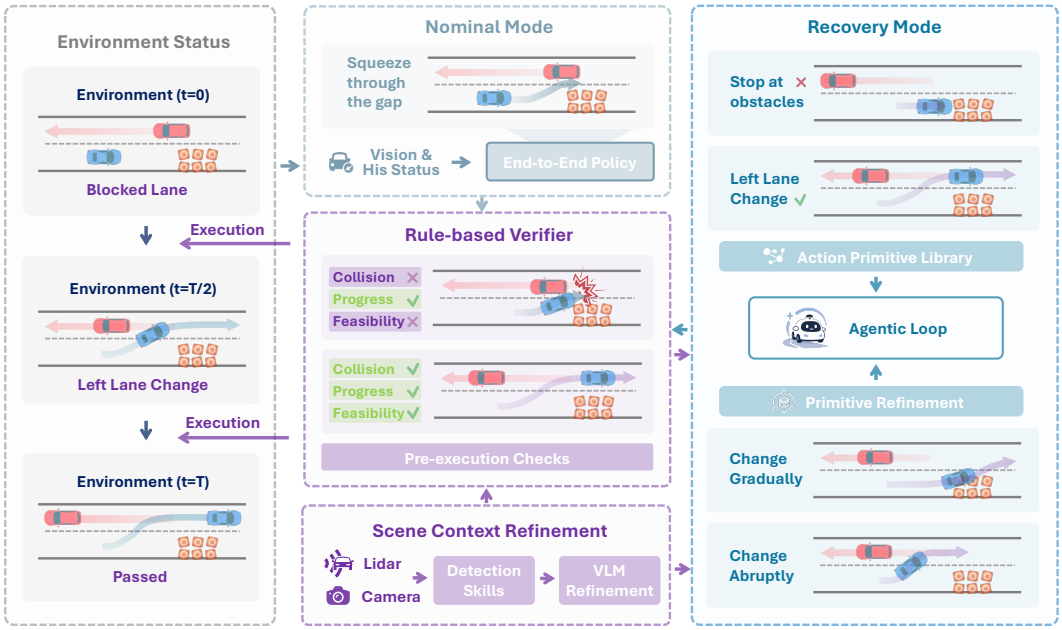


Figure 2: **Overview of the AUTOAGENT0 runtime.** AUTOAGENT0 surrounds the nominal end-to-end policy with a) scene context refinement, which uses perception skills to augment the evidence available to a rule-based verifier, and b) agentic recovery planning, which generates and revises recovery planning trajectories when no nominal proposal is admitted. The verifier evaluates both nominal proposals and recovery planning trajectories before commitment to the tracking controller.

which selects the highest-ranked admitted proposal according to the planning objective. *(2) Recovery planning.* When no nominal proposal is admitted, the runtime invokes a recovery policy π_{R}. Thus the executed trajectory τ_t at decision step t could be written as:

$$\tau_t = \mathrm{Runtime}(c_t) = \begin{cases} \pi_{\mathrm{N}}^{V}(c_t), & \text{if a nominal proposal is admitted,} \\ \pi_{\mathrm{R}}(c_t), & \text{otherwise.} \end{cases} \tag{1}$$

3.2 AUTOAGENT0 RUNTIME

AUTOAGENT0 comprises a runtime harness that coordinates scene context refinement, trajectory verification, and recovery planning. The harness invokes VLM reasoning for semantic assessment and recovery maneuver selection, integrating these judgments with perception outputs, verification results, and execution feedback. Figure 2 illustrates this structure; Section 3.3 details the recovery planner.

Agent as the Runtime Operator. Following the agentic operator formulation of AVO (Chen et al., 2026), the agent implements Eq. equation 1 using scene-refinement skills $\mathcal{F}$, parameterized recovery primitives $\mathcal{P}$, and a rule-based verifier V:

$$\mathrm{Runtime}_{\mathrm{agent}}(c_t) = \mathrm{Agent}(c_t; \mathcal{F}, \mathcal{P}, V). \tag{2}$$

Here, the agent denotes the runtime harness, which invokes VLM reasoning for semantic assessment and recovery decisions.

Scene Context Refinement. Reliable verification requires scene evidence that can expose errors in nominal proposals. The agent applies perception skills from $\mathcal{F}$ to remove false positive and false negative detections in c_t, producing refined context $\widetilde{c}_t$ for verification and recovery planning. The skills include cross-modal checks that identify discrepancies between camera and LiDAR observations, and vision-language reasoning provides semantic validation of relevant objects and scene conditions. These operations revise the scene representation used for verification, addressing missed actors and spurious detections that can otherwise cause incorrect admission of nominal proposals or unnecessary activation of recovery planning.

Rule-based Verification. The rule-based verifier V evaluates nominal proposals and recovery planning trajectories against $\widetilde{c}_t$, returning an admission decision and feedback on violated criteria. Its fixed admission criteria combine a threshold on a PDMS-based trajectory score (Dauner et al., 2023) with route-consistency and execution guards. The score assesses collision and drivable-area compliance, time to collision, lane keeping, and ego progress; the guards account for conditions such as persistent blockage and confirmed semantic inconsistency. When no nominal proposal is admitted, the agent implements π_{R} by generating recovery planning trajectories from $\mathcal{P}$ and using verifier feedback to revise primitive parameters or select another primitive. Each revision is checked against $\widetilde{c}_t$ under the same admission criteria. The complete scoring function, hard gates, and admission threshold are specified in Appendix E. Section 3.3 specifies this iterative refinement loop.

3.3 Recovery Planning via Agentic Parameter Search

Recovery from off-nominal states requires corrective actions that the nominal policy, trained on largely nominal demonstrations, may fail to propose. AutoAgent0 reduces this dependence by decomposing recovery into components that do not rely on the expert's training data: parameterized primitives generate trajectories analytically from the current geometry, the rule-based verifier checks them against explicit criteria, and the VLM contributes only high-level maneuver selection, which draws on broad pretrained reasoning rather than driving demonstrations. Moreover, recovery does not need to be correct in a single pass: rejection feedback allows the runtime to revise parameters or reconsider the maneuver before execution. The resulting generalization is bounded by the coverage of the primitive library (Section D.1).

Action Primitive Library. We define an action primitive library $\mathcal{P}$, where each primitive $\ell \in \mathcal{P}$ is associated with a trajectory generator G_ℓ. A parameter vector $\alpha \in \mathcal{A}_\ell$ controls quantities such as lateral displacement, target speed, and execution duration, with $\mathcal{A}_\ell$ specifying the primitive's bounded parameter domain. For example, braking and creeping adjust longitudinal motion, lateral bypass changes the path around an obstacle, and reverse actions create space for subsequent forward progress. These primitives initially generate recovery planning trajectories that adapt to local geometry and route requirements through continuous parameter selection.

Verifier-Guided Planning Refinement. The recovery policy π_{R} alternates primitive selection, trajectory generation, and verification (Figure 3). At each iteration, the agent selects a primitive ℓ and parameters α to generate a recovery planning trajectory: $tau_{\text{recovery}} = G_\ell(\widetilde{c}_t, \alpha)$. Starting from nominal rejection feedback, the agent reasons about scene constraints and why proposals failed to decide whether to revise the current primitive's parameters or select a different maneuver. It explores a parameter set for the selected primitive and evaluates candidates with the verifier V, retaining proposals and feedback for subsequent decisions. For example, the agent may infer from collision feedback and surrounding geometry that forward is constrained, and choose to reverse to create room for a subsequent turn. Algorithm 1 provides the complete procedure. Since recovery planning is triggered only when no nominal proposal is admitted, VLM reasoning is reserved for situations the nominal policy cannot resolve, rather than being invoked at every control step. Appendix B reports the full latency analysis.

4 Experiments

4.1 Experimental Protocol

We evaluate whether AutoAgent0 serves as a reliable runtime for safe closed-loop driving without retraining the underlying driving expert. Our experiments mainly aim to answer the following research questions: **RQ1:** Can our runtime improve the closed-loop robustness of existing driving experts in long-tailed and safety-critical scenarios without retraining them? **RQ2:** How do key harness components and VLM backbone choice affect the effectiveness of verification and recovery planning in AutoAgent0? and **RQ3:** what advantages do agentic closed-loop runtime bring compared with traditional closed-loop runtime monitors?

Benchmarks and Evaluation Settings. We mainly conduct experiments in Fail2Drive (Gerstenecker et al., 2026), which is a CARLA-based closed-loop benchmark that tests driving abilities under long-tailed scenarios. Additionally, we show the generalizability of AutoAgent0 in NavSafe-

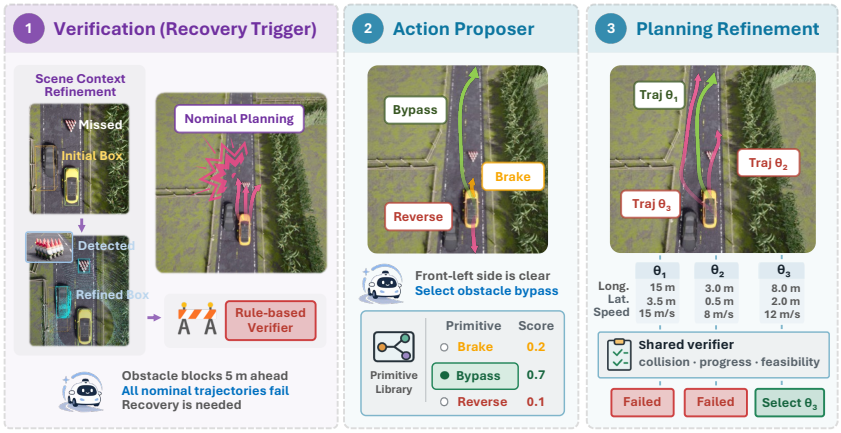


Figure 3: **Agentic Recovery Planning.** Rejection of nominal proposals initiates action primitive selection and parameterized trajectory generation. The agent uses scene context and verifier feedback to revise recovery planning trajectories before execution. During execution, new observations can trigger further revision, with each replacement trajectory checked before commitment.

∞ (Bao et al., 2026), which is photorealistic closed-loop benchmark that evaluates driving ability across safety-critical events. The evaluation metrics include success rate (SR), driving score (DS), along with HM on Fail2Drive and driving efficiency, comfort, and capability-specific scores on NavSafe-∞. Detailed metric definitions are provided in Appendix. Closed-loop results are averaged over three matched evaluation repetitions unless otherwise stated. The runtime-monitor evaluation in Section 4.4 uses 160 non-calibration routes, comprising 80 In-Distribution/Generalization pairs.

Implementation Details. Unless stated otherwise, we use SparseDriveV2 (Sun et al., 2026) as the driving expert, BridgeDrive (Liu et al., 2026a) as the sensor-based detector, and Opus 5 (Anthropic, 2026) for semantic supervision and recovery selection. Recovery variants use the same front-camera input to the VLM and the same deterministic primitive runtime. For Fail2Drive, we augment the frozen BridgeDrive detector with trained pedestrian and static-obstacle modules while keeping the driving expert frozen; Appendix G details the adaptation and offline evaluation. NavSafe-∞ uses native BridgeDrive perception.

4.2 Main Results (RQ1)

Results on Fail2Drive. Table 1 compares AutoAgent0-augmented UniAD, SparseDriveV2, and BridgeDrive with their standalone counterparts, alongside representative driving baselines. AutoAgent0 consistently improves DS, SR, and HM across all three experts on both splits, with larger DS gains on generalization routes. The gains mainly come from avoided collisions: most newly successful routes previously ended in collisions, and pedestrian and animal collisions are nearly eliminated. Because standalone experts collide more often on generalization routes, removing these failures yields larger gains under distribution shift. Some avoided collisions instead end in stops when no admissible recovery maneuver is found, reflecting the runtime's fail-safe preference and the coverage limits of the primitive library (Appendix A.5).

Results on NavSafe-∞. To show the generalizability of AutoAgent0, we compare the performance enhancement in photorealistic closed-loop benchmark NavSafe-∞ (Table 2). AutoAgent0 improves task success and all capability scores across base experts, with larger gains for weaker experts, narrowing the performance gap among them. This indicates that the runtime compensates for expert-specific failures rather than adding a uniform improvement. The augmented experts also surpass DrivoR and GTRS-Dense–V2-99 in DS and SR, at the cost of lower comfort, since recovery primitives prioritize admissibility and progress over smoothness.

Scenario-wise Analysis. Figure 4 shows that the gains of AutoAgent0 depend on how the expert fails in each scenario group. Scene context refinement mainly resolves missed pedestrians and animals in Visual–Longitudinal scenarios, while recovery planning drives the largest gains in Behavior scenarios by bypassing or waiting out static blockages. In Visual–Lateral scenarios, however,

Table 1: **Benchmark Results on Fail2Drive.** Average reports the arithmetic mean of in-distribution and generalization performance for each metric. Parentheses in Average indicate absolute improvements over the corresponding base method.

Method	Inputs		In-Distribution			Generalization			Average		
	RGB	LiDAR	DS ↑	SR (%) ↑	HM ↑	DS ↑	SR (%) ↑	HM ↑	DS ↑	SR (%) ↑	HM ↑
TCP (Wu et al., 2022)	✓	✗	24.7	39.1	30.3	24.5	31.4	27.5	24.6	35.3	28.9
Orion (Fu et al., 2025)	✓	✗	53.0	52.0	52.5	51.2	46.0	48.5	52.1	49.0	50.5
HiP-AD (Tang et al., 2025)	✓	✗	74.1	70.7	72.4	67.1	56.7	61.5	70.6	63.7	67.0
SimLingo (Renz et al., 2025)	✓	✗	82.6	79.3	80.9	71.7	55.0	62.2	77.2	67.2	71.6
AlignDrive (Wu et al., 2026)	✓	✗	74.7	75.7	75.2	68.6	62.7	65.5	71.7	69.2	70.4
BevAD (Holtz et al., 2026)	✓	✗	87.4	83.3	85.3	82.3	68.7	74.9	84.9	76.0	80.1
TF++ (Jaeger et al., 2023)	✓	✓	83.3	78.5	80.8	75.4	61.1	67.5	79.4	69.8	74.2
TFv6 (Nguyen et al., 2026)	✓	✓	90.2	93.3	91.7	79.5	70.7	74.8	84.9	82.0	83.3
UniAD (Hu et al., 2023)	✓	✗	47.5	36.3	41.2	44.0	27.6	33.9	45.8	32.0	37.6
+ AutoAgent0	✓	✓	68.4	69.0	68.7	69.2	69.0	69.1	68.8 +23.0	69.0 +37.0	68.9 +31.3
SparseDriveV2 (Sun et al., 2026)	✓	✗	70.0	74.0	71.9	62.9	52.7	57.4	66.5	63.4	64.7
+ AutoAgent0	✓	✓	80.7	85.7	83.1	79.2	81.0	80.1	80.0 +13.5	83.4 +20.0	81.7 +17.0
BridgeDrive (Liu et al., 2026a)	✓	✓	91.6	95.0	93.3	81.9	75.0	78.3	86.8	85.0	85.8
+ AutoAgent0	✓	✓	**91.7**	**95.3**	**93.5**	**85.7**	**85.3**	**85.5**	**88.7** +1.9	**90.3** +5.3	**89.5** +3.7

Table 2: **Closed-loop Evaluation on NavSafe-∞.** All reported values are three-seed means over the 280-scenario evaluation set. Parentheses indicate absolute improvement over the corresponding base method.

Method	Closed-Loop Metrics				Ability Scores			
	DS ↑	SR(%) ↑	DE ↑	Comf. ↑	TC ↑	VRUC ↑	TV ↑	TI ↑
DrivoR (Kirby et al., 2026)	64.7	50.7	188.0	87.9	58.7	31.7	52.4	43.3
GTRS-Dense–V2-99 (Li et al., 2025a)	64.3	49.5	163.9	95.9	52.0	40.0	54.2	36.7
AutoVLA (Zhou et al., 2025a)	51.1	29.5	186.0	65.4	33.0	15.0	29.7	36.7
RAP (Feng et al., 2026)	47.7	27.3	**220.8**	48.4	32.3	14.2	28.8	22.2
MTDrive–SFT (Li et al., 2026b)	48.8	31.8	173.8	94.5	30.3	15.0	35.8	44.4
LTF (Dauner et al., 2024)	53.0	27.7	163.7	**97.8**	31.7	25.0	28.5	15.6
+ AutoAgent0	74.8 +21.8	60.5 +32.8	160.0	78.8	66.0 +34.3	45.8 +20.8	63.0 +34.5	52.2 +36.6
DiffusionDrive (Liao et al., 2025)	44.8	21.0	165.1	97.7	21.3	11.7	23.6	22.2
+ AutoAgent0	70.9 +26.1	52.3 +31.3	160.2	72.3	52.0 +30.7	37.5 +25.8	58.8 +35.2	48.9 +26.7
SparseDriveV2 (Sun et al., 2026)	65.3	48.8	168.0	93.4	54.0	43.3	48.2	41.1
+ AutoAgent0	**77.7** +12.4	**63.6** +14.8	165.3	76.0	**68.0** +14.0	**50.0** +6.7	**64.5** +16.3	**63.3** +22.2

many avoided collisions end in stops when no admissible bypass fits the tight geometry, which remains the main open case. On NavSafe-∞, the largest gains appear in Traffic Incidents, where the examples in (c) show verified yielding and turning (Appendix A.5).

4.3 Component Analysis and Ablation Studies (RQ2)

Complementary contributions of runtime components. Table 3 presents cumulative ablations on frozen SparseDriveV2, using a shared controlled fallback when recovery planning is disabled. Each component addresses the bottleneck left by the previous one. Verification substantially reduces collisions but slightly lowers task success, since rejecting unsafe proposals without alternatives leaves only conservative fallback. Scene context refinement then improves both safety and task success, as recovering missed actors prevents collisions and removing spurious detections avoids unnecessary interventions. Recovery planning further raises task success and sharply reduces trigger frequency, indicating that its maneuvers resolve failures rather than repeatedly contain them, although moving past blockages increases collision events relative to braking (Appendix A.1). Replacing refined context with ground truth yields further gains, identifying perception as the main remaining bottleneck.

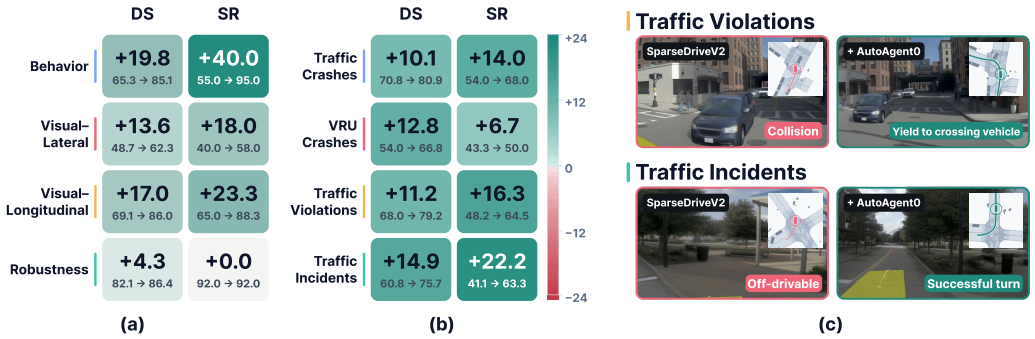


Figure 4: **Scenario-wise improvements and qualitative examples.** DS and SR gains from augmenting SparseDriveV2 with AUTOAGENT0 on (a) Fail2Drive and (b) NavSafe-∞. (c) NavSafe-∞ examples comparing standalone and augmented driving behavior.

Table 3: **Ablation of the three runtime modules.** Col. is the mean number of collision events per route. Rec./km denotes recovery-trigger events per kilometer; standalone SparseDriveV2 has no recovery module, so its Rec./km is zero by construction. Routes reaching the 180 s simulation cap are scored as scenario timeouts (Appendix F.2). † Simulator ground-truth scene context replaces the output of scene context refinement. ‡ Rec./km is not available for this run.

Variant	Context Refinement	Rule-based Verification	Recovery Planning	DS ↑	SR (%) ↑	Col. ↓	Rec./km
SparseDriveV2	–	–	–	66.5	63.4	0.445	0.0
+ Rule-based verification‡	–	✓	–	69.1	60.0	0.095	–
+ Scene context refinement	✓	✓	–	76.0	74.0	0.050	54.2
+ Recovery planning (full)	✓	✓	✓	80.0	83.4	0.145	20.3
AutoAgent0†	GT	✓	✓	81.4	90.3	0.110	26.2

Effect of VLM backbone choice. We compare Qwen3.5-9B (Qwen Team, 2026), Opus 5 (Anthropic, 2026), and GPT-6 Astra (OpenAI, 2026) as the VLM backbone within the full AUTOAGENT0 runtime on Fail2Drive (Appendix A.4). All three backbones outperform the variant without recovery planning, although Qwen yields lower task success and more collisions. This gap mainly reflects differences in scene understanding and error correction. Qwen more frequently hallucinates scene elements and interprets the same scene inconsistently across observations. Opus and Astra also misjudge motion at times, but they use failure feedback to revise their interpretations and act more conservatively under uncertainty. Effective recovery therefore depends not only on accurate scene understanding, but also on the ability to correct errors when perception is imperfect.

4.4 COMPARISON WITH TRADITIONAL RUNTIME MONITORS (RQ3)

Compared runtimes. We compare AUTOAGENT0 with three runtime monitors around the frozen SparseDriveV2 expert. The RSS-based monitor triggers braking under safe-distance violations (Shalev-Shwartz et al., 2017). The longitudinal MPC baseline adjusts speed along the nominal path subject to dynamics and obstacle constraints (Wabersich & Zeilinger, 2021). The uncertainty monitor triggers deceleration based on candidate-score entropy and margins. Monitor proposals replace the nominal trajectory only when they pass verification and achieve a higher PDMS score. Appendix F.4 provides the experimental setup and implementation details.

Results and analysis. Figure 5(a) shows that AUTOAGENT0 achieves higher DS and SR than all evaluated monitors on both In-Distribution and Generalization routes. The monitors only adjust speed along the expert's path, leaving three failure sources unaddressed: incomplete obstacle evidence, stalled progress under trajectories that pass the score threshold, and blockages that braking alone cannot resolve. AUTOAGENT0 targets these through scene context refinement, execution feedback, and recovery maneuvers, respectively (Section 4.3). Figure 5(b,c) illustrates the latter

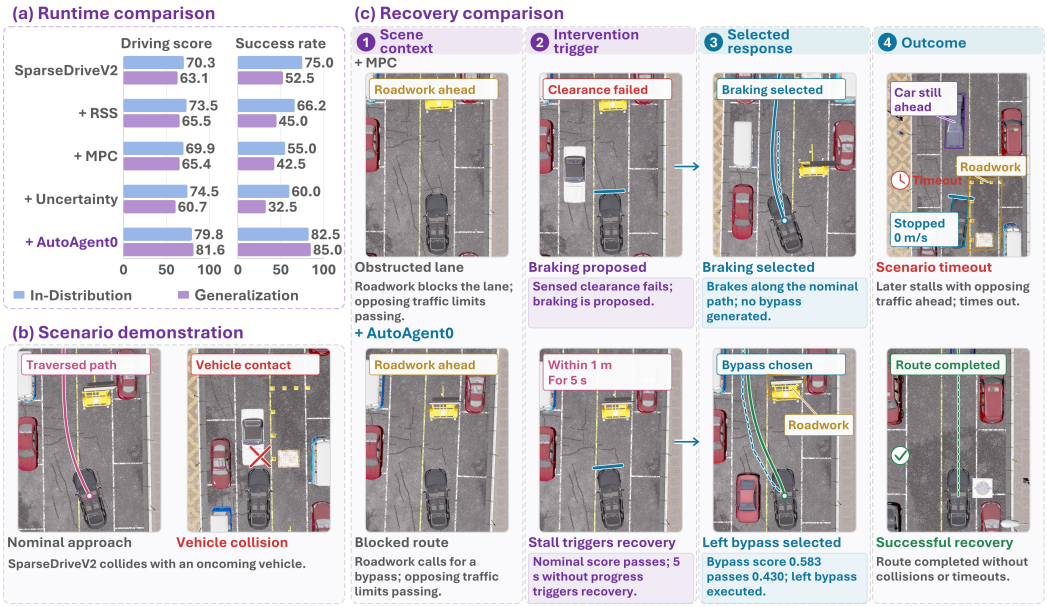


Figure 5: **Runtime performance and intermediate recovery decisions.** **(a)** DS and SR on 80 In-Distribution/Generalization route pairs, excluding calibration routes, with one rollout per route. **(b)** SparseDriveV2's maneuver ends in a vehicle collision. **(c)** MPC brakes but later stalls and times out; AUTOAGENT0 detects stalled progress and executes a checked left bypass, completing the route without collisions or timeouts. Qualitative rollouts use different configurations; see Appendix F.4.

two in a construction-obstacle scenario: MPC avoids a collision by braking but then stalls behind opposing traffic and times out, whereas AUTOAGENT0 detects the stalled progress and executes a verified left bypass, completing the route. Moreover, a single intervention signal does not reliably ensure safe execution. The uncertainty monitor's SR drops from 60.0% to 32.5% on Generalization routes. Trace inspection reveals two failure modes: brief risk spikes that never form a sustained warning, and requested slowdowns that lose score-based arbitration to proposals with greater predicted progress (Appendix A.3). Avoiding collisions thus requires recognizing hazards from scene evidence, while restoring progress requires responses beyond speed adjustment.

5 DISCUSSION AND LIMITATIONS

Discussion and Conclusion. Our results highlight runtime supervision as a complementary direction for improving closed-loop driving beyond advances in the underlying policy. Improvements across multiple frozen experts and two benchmarks suggest that robustness can be enhanced by governing how expert proposals are assessed and how execution failures are handled, without retraining the experts themselves. The component ablations highlight the complementary roles of scene assessment, trajectory verification, and recovery planning, suggesting that effective supervision requires both identifying unreliable actions and finding alternatives that restore progress. The comparison with the evaluated runtime monitors further shows the value of using scene context and execution feedback to recover from situations that conservative interventions alone cannot resolve. Taken together, these findings support the central design of AUTOAGENT0: integrating independent scene assessment, rule-based verification, and feedback-driven recovery within a unified runtime. AUTOAGENT0 demonstrates how an agentic runtime can complement existing driving experts to improve closed-loop robustness in the evaluated long-tail and safety-critical scenarios.

Limitations. Although AUTOAGENT0 improves closed-loop robustness, its verification and recovery remain constrained by scene estimation accuracy and the coverage of the available recovery primitives. While AUTOAGENT0 improves task success on NavSafe-∞, the accompanying reductions in comfort and driving efficiency indicate room for improving the quality of recovery behavior. Recovery decisions currently incur second-level VLM latency (Appendix B), which could be reduced through smaller or distilled models.

AI USE STATEMENT

Generative AI tools were used to improve the clarity, grammar, and readability of the manuscript, and to assist with writing and debugging experimental code and executing experiments. All AI-generated suggestions, code, and outputs were reviewed and verified by the authors. The experimental design, analysis of results, scientific claims, and final manuscript were determined and validated by the authors. The authors take full responsibility for the content of this work.

REPRODUCIBILITY STATEMENT

We describe the runtime formulation of AUTOAGENT0 in Section 3 and provide its control flow and scene context refinement procedures in Appendix C. Appendix D details the recovery primitives, parameter search, VLM prompting protocol, and execution procedure, while Appendix E specifies the verifier terms and admission rule. Appendix F documents model configurations, sensor inputs, simulator version, hardware, verifier parameters, benchmark metrics, and evaluation protocols. Closed-loop results are averaged over three matched evaluation repetitions unless otherwise stated; Appendix B reports the runtime profiling procedures and inference conditions, and Appendix G describes the training and evaluation settings for the supplementary perception adaptation study.

REFERENCES

Niket Agarwal, Arslan Ali, Jon Allen, Martin Antolini, Adeline Aubame, Alisson Azzolini, Junjie Bai, Maciej Bala, Yogesh Balaji, Josh Bapst, et al. Cosmos 3: Omnimodal world models for physical AI. *arXiv preprint arXiv:2606.02800*, 2026.

Arslan Ali, Junjie Bai, Maciej Bala, Yogesh Balaji, Aaron Blakeman, Tiffany Cai, Jiaxin Cao, Tianshi Cao, Elizabeth Cha, Yu-Wei Chao, et al. World simulation with video foundation models for physical AI. *arXiv preprint arXiv:2511.00062*, 2025.

Anthropic. Claude code: Best practices for agentic coding, 2025. URL https://www.anthropic.com/engineering/claude-code-best-practices. Accessed April 2026.

Anthropic. Claude Opus 5 system card. https://www-cdn.anthropic.com/b514064af1408018e64b1ad24e7d5e75850b4ffd/Claude%20Opus%205%20System%20Card.pdf, 2026. Accessed September 2026.

Yuxin Bao, Hongwei Ruan, Luobin Wang, Seth Z. Zhao, Ziyang Leng, Zihan Zhang, Yu Zeng, Rowan McAllister, Henrik Christensen, and Bolei Zhou. Navsafe-∞: Benchmarking closed-loop driving safety in photorealistic environments. *arXiv preprint arXiv:2609.26618*, 2026. URL https://arxiv.org/abs/2609.26618.

Kaustav Chakraborty, Zeyuan Feng, Sushant Veer, Apoorva Sharma, Boris Ivanovic, Marco Pavone, and Somil Bansal. System-level safety monitoring and recovery for perception failures in autonomous vehicles, 2024. URL https://arxiv.org/abs/2409.17630.

Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15590–15599, 2021.

Terry Chen, Zhifan Ye, Bing Xu, Zihao Ye, Timmy Liu, Ali Hassani, Tianqi Chen, Andrew Kerr, Haicheng Wu, Yang Xu, Yu-Jung Chen, Hanfeng Chen, Aditya Kane, Ronny Krashinsky, Ming-Yu Liu, Vinod Grover, Luis Ceze, Roger Bringmann, John Tran, Wei Liu, Fung Xie, Michael Lightstone, and Humphrey Shi. AVO: Agentic variation operators for autonomous evolutionary search. *arXiv preprint arXiv:2603.24517*, 2026. URL https://arxiv.org/abs/2603.24517.

Jose Leopoldo Contreras, Ola Shorinwa, and Mac Schwager. Safe, out-of-distribution-adaptive MPC with conformalized neural network ensembles. In *Proceedings of the 7th Annual Learning for Dynamics and Control Conference*, volume 283 of *Proceedings of Machine Learning Research*, pp.

194–207, 2025. URL https://proceedings.mlr.press/v283/contreras25a.html.

Daniel Dauner, Marcel Hallgarten, Andreas Geiger, and Kashyap Chitta. Parting with misconceptions about learning-based vehicle motion planning. In *Proceedings of the 7th Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1268–1281. PMLR, 2023. URL https://arxiv.org/abs/2306.07962.

Daniel Dauner, Marcel Hallgarten, Tianyu Li, Xinshuo Weng, Zhiyu Huang, Zetong Yang, Hongyang Li, Igor Gilitschenski, Boris Ivanovic, Marco Pavone, Andreas Geiger, and Kashyap Chitta. NAVSIM: Data-driven non-reactive autonomous vehicle simulation and benchmarking. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.15349.

Lan Feng, Yang Gao, Eloi Zablocki, Quanyi Li, Wuyang Li, Sichao Liu, Matthieu Cord, and Alexandre Alahi. RAP: 3d rasterization augmented end-to-end planning. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.04333.

Haoyu Fu, Diankun Zhang, Zongchuang Zhao, Jianfeng Cui, Dingkang Liang, Chong Zhang, Dingyuan Zhang, Hongwei Xie, Bing Wang, and Xiang Bai. ORION: A holistic end-to-end autonomous driving framework by vision-language instructed action generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 24823–24834, 2025. URL https://arxiv.org/abs/2503.19755.

Simon Gerstenecker, Andreas Geiger, and Katrin Renz. Fail2Drive: Benchmarking closed-loop driving generalization. *arXiv preprint arXiv:2604.08535*, 2026. URL https://arxiv.org/abs/2604.08535.

David Holtz, Niklas Hanselmann, Simon Doll, Marius Cordts, and Bernt Schiele. What matters for scalable and robust learning in end-to-end driving planners? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Findings*, 2026. URL https://arxiv.org/abs/2603.15185.

Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. URL https://arxiv.org/abs/2212.10156.

Bernhard Jaeger, Kashyap Chitta, and Andreas Geiger. Hidden biases of end-to-end driving models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. URL https://arxiv.org/abs/2306.07957.

Xiaosong Jia, Zhenjie Yang, Qifeng Li, Zhiyuan Zhang, and Junchi Yan. Bench2Drive: Towards multi-ability benchmarking of closed-loop end-to-end autonomous driving. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.03877.

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 (ICCV)*, 2023. URL https://arxiv.org/abs/2303.12077.

Peter Karkus, Maximilian Igl, Yuxiao Chen, Kashyap Chitta, Jef Packer, Bertrand Douillard, Ran Tian, Alexander Naumann, Guillermo Garcia-Cobo, Shuhan Tan, Alperen Değirmenci, Alexander Popov, Nikolai Smolyanskiy, Urs Muller, Boris Ivanovic, and Marco Pavone. Beyond behavior cloning in autonomous driving: A survey of closed-loop training techniques. *TechRxiv preprint*, 2025. doi: 10.36227/techrxiv.176591565.57674132/v1. URL https://www.techrxiv.org/doi/full/10.36227/techrxiv.176591565.57674132/v1.

Jungho Kim, Jiyong Oh, Seunghoon Yu, Hongjae Shin, Donghyuk Kwak, and Jun Won Choi. Safedrive: Fine-grained safety reasoning for end-to-end driving in a sparse world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24854–24864, 2026.

Ellington Kirby, Alexandre Boulch, Yihong Xu, Yuan Yin, Gilles Puy, Éloi Zablocki, Andrei Bursuc, Spyros Gidaris, Renaud Marlet, Florent Bartoccioni, Anh-Quan Cao, Nermin Samet, Tuan-Hung Vu, and Matthieu Cord. Driving on registers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 32058–32069, 2026.

Tianyu Li, Li Chen, Caojun Wang, Haochen Liu, Kashyap Chitta, Zhenjie Yang, Yuhang Lu, Naisheng Ye, Yihang Qiu, Yufei Wang, Luoxi Zou, Jiaxin Peng, Jin Pan, Zhaoyu Su, Andrei Bursuc, Shengbo Eben Li, Andreas Geiger, Peng Su, and Hongyang Li. World Engine: Towards the era of post-training for autonomous driving. *arXiv preprint arXiv:2606.19836*, 2026a. URL https://arxiv.org/abs/2606.19836.

Xidong Li, Mingyu Guo, Chenchao Xu, Bailin Li, Wenjing Zhu, Yangang Zou, Rui Chen, and Zehuan Wang. MTDrive: Multi-turn interactive reinforcement learning for autonomous driving. *arXiv preprint arXiv:2601.22930*, 2026b. URL https://arxiv.org/abs/2601.22930.

Yingyan Li, Shuyao Shang, Weisong Liu, Bing Zhan, Haochen Wang, Yuqi Wang, Yuntao Chen, Xiaoman Wang, Yasong An, Chufeng Tang, Lu Hou, Lue Fan, and Zhaoxiang Zhang. DriveVLA-W0: World models amplify data scaling law in autonomous driving. In *International Conference on Learning Representations (ICLR)*, 2026c. URL https://arxiv.org/abs/2510.12796.

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*, 2024a. 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*, 2025a. URL https://arxiv.org/abs/2506.06664.

Zhiqi Li, Zhiding Yu, Shiyi Lan, Jiahan Li, Jan Kautz, Tong Lu, and Jose M. Alvarez. Is ego status all you need for open-loop end-to-end autonomous driving? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b. URL https://arxiv.org/abs/2312.03031.

Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Qiao Yu, and Jifeng Dai. BEVFormer: Learning bird's-eye-view representation from LiDAR-camera via spatiotemporal transformers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(3):2020–2036, 2025b.

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)*, 2025. URL https://arxiv.org/abs/2411.15139.

Haochen Liu, Zhiyu Huang, Wenhui Huang, Haohan Yang, Xiaoyu Mo, and Chen Lv. Hybrid-prediction integrated planning for autonomous driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(4):2597–2614, 2025a.

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. In *International Conference on Learning Representations (ICLR)*, 2026a. URL https://arxiv.org/abs/2509.23589.

Xinyi Liu, Mohammadreza Fani Sani, Zewei Zhou, Julius Wirbel, Bahram Zarrin, and Roberto Galeazzi. Robopilot: Generalizable dynamic robotic manipulation with dual-thinking modes. *arXiv preprint arXiv:2510.00154*, 2025b.

Xinyu Liu, Darryl C. Jacob, Yuxin Liu, Xinsong Du, Muchao Ye, Bolei Zhou, and Pan He. Understanding real-world traffic safety through RoadSafe365 benchmark. *arXiv preprint arXiv:2602.07212*, 2026b. URL https://arxiv.org/abs/2602.07212.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic /skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026. URL https://arxiv.org/abs/2607.00272.

Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. A language agent for autonomous driving. In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2311.10813.

Long Nguyen, Micha Fauth, Bernhard Jaeger, Daniel Dauner, Maximilian Igl, Andreas Geiger, and Kashyap Chitta. Lead: Minimizing learner-expert asymmetry in end-to-end driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. URL https://arxiv.org/abs/2512.20563.

OpenAI. Introducing GPT-6 Astra. https://openai.com/index/gpt-6-astra/, 2026. Accessed September 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Katrin Renz, Long Chen, Elahe Arani, and Oleg Sinavski. Simlingo: Vision-only closed-loop autonomous driving with language-action alignment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11993–12003, 2025. URL https://arxiv.org/abs/2503.09594.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023. URL https://proceedings.neurips.cc/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.

Rohan Sinha, Edward Schmerling, and Marco Pavone. Closing the loop on runtime monitors with fallback-safe MPC. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2023. doi: 10.1109/CDC49753.2023.10383965. URL https://arxiv.org/abs/2309.08603.

Rohan Sinha, Amine Elhafsi, Christopher Agia, Matthew Foutter, Edward Schmerling, and Marco Pavone. Real-time anomaly detection and reactive planning with large language models, 2024. URL https://arxiv.org/abs/2407.08735.

Wenchao Sun, Xuewu Lin, Keyu Chen, Zixiang Pei, Xiang Li, Yining Shi, and Sifa Zheng. SparseDriveV2: Scoring is all you need for end-to-end autonomous driving. *arXiv preprint arXiv:2603.29163*, 2026.

Yingqi Tang, Zhuoran Xu, Zhaotie Meng, and Erkang Cheng. HiP-AD: Hierarchical and multi-granularity planning with deformable attention for autonomous driving in a single decoder. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. URL https://arxiv.org/abs/2503.08612.

Kim P. Wabersich and Melanie N. Zeilinger. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica*, 129, 2021. URL https://arxiv.org/abs/1812.05506.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Penghao Wu, Xiaosong Jia, Li Chen, Junchi Yan, Hongyang Li, and Yu Qiao. Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline. In *Advances in Neural Information Processing Systems*, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Yanhao Wu, Haoyang Zhang, Fei He, Rui Wu, Yanhu Shan, Congpei Qiu, Liang Gao, Wei Ke, and Tong Zhang. AlignDrive: Aligned lateral-longitudinal planning for end-to-end autonomous driving. *arXiv preprint arXiv:2601.01762*, 2026. URL https://arxiv.org/abs/2601.01762.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026. URL https://arxiv.org/abs/2606.19980.

Runsheng Xu, Hubert Lin, Wonseok Jeon, Hao Feng, Yuliang Zou, Liting Sun, John Gorman, Kate Tolstaya, Sarah Tang, Brandyn White, Ben Sapp, Mingxing Tan, Jyh-Jing Hwang, and Dragomir Anguelov. WOD-E2E: Waymo open dataset for end-to-end driving in challenging long-tail scenarios. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Jiazhao Zhang, Gengze Zhou, Hale Yin, Yiyang Huang, Zixing Lei, Qihang Peng, Haoqi Yuan, Jie Zhang, Xudong Guo, Xiaoyue Chen, et al. Qwen-RobotNav technical report: A scalable navigation model designed for an agentic navigation system. *arXiv preprint arXiv:2606.18112*, 2026.

Seth Z. Zhao, Luobin Wang, Hongwei Ruan, Yuxin Bao, Yilan Chen, Ziyang Leng, Abhijit Ravichandran, Honglin He, Zewei Zhou, Xu Han, Abhishek Peri, Zhiyu Huang, Pranav Desai, Henrik Christensen, Jiaqi Ma, and Bolei Zhou. BridgeSim: Unveiling the OL–CL gap in end-to-end autonomous driving. *arXiv preprint arXiv:2604.10856*, 2026. URL https://arxiv.org/abs/2604.10856.

Zewei Zhou, Tianhui Cai, Seth Z. Zhao, Yun Zhang, Zhiyu Huang, Bolei Zhou, and Jiaqi Ma. AutoVLA: A vision-language-action model for end-to-end autonomous driving with adaptive reasoning and reinforcement fine-tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 27920–27956, 2025a.

Zewei Zhou, Hao Xiang, Zhaoliang Zheng, Seth Z Zhao, Mingyue Lei, Yun Zhang, Tianhui Cai, Xinyi Liu, Johnson Liu, Maheswari Bajji, et al. V2XPnP: Vehicle-to-everything spatio-temporal fusion for multi-agent perception and prediction. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 25399–25409. IEEE, 2025b.

Zewei Zhou, Seth Z Zhao, Tianhui Cai, Zhiyu Huang, Bolei Zhou, and Jiaqi Ma. TurboTrain: Towards efficient and balanced multi-task learning for multi-agent perception and prediction. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4391–4402. IEEE, 2025c.

Zewei Zhou, Ruining Yang, Yiluan Guo, Sherry X Chen, Tao Feng, Kateryna Pistunova, Yishan Shen, Lili Su, and Jiaqi Ma. SpanVLA: Efficient action bridging and learning from negative-recovery samples for vision-language-action model. *arXiv preprint arXiv:2604.19710*, 2026.

A Additional Results and Mechanism Analysis

This appendix extends the evaluation of AutoAgent0 by examining four questions. First, which recovery mechanisms account for the observed performance gains (Section A.1)? Second, how do scene interpretation and feedback affect recovery behavior across VLM backbones (Sections A.2 and A.4)? Third, where are the gains and remaining limitations concentrated across scenario groups and distribution shifts (Section A.5)? Fourth, how does the verifier threshold affect the balance between collision avoidance and task completion (Section A.6)? We address these questions through recovery ablations, execution traces, backbone comparisons, scenario level results, and a targeted threshold sweep.

A.1 Recovery Mechanism Ablations

Discrete skills raise tasks success but add collisions. Adding discrete recovery skills raises SR from 74.0% to 77.0% and DS slightly from 76.0 to 76.3, while Col. increases from 0.050 to 0.125 (Table 4). Skills add maneuvers such as reversing and bypassing, but each skill runs with fixed displacement, speed, and duration. We therefore tested whether choosing these parameters helps increase the DS and SR.

Parameter selection does not fix a poor maneuver choice. It does not: with parameter selection, DS stays at 76.3, SR rises by only 0.3 points, and Col. increases from 0.125 to 0.200. Changing a maneuver's geometry does not help when the maneuver itself is wrong. For example, a longer forward bypass cannot create room to reverse when the vehicle is too close to the obstacle. This variant also selects once and brakes if the proposal is rejected, so it never tries a different maneuver. Recovery therefore needs a way to revise the maneuver, not only its parameters, and more flexibility can increase collisions.

Feedback-guided revision improves task completion. Compared with single-shot parameter selection, verifier-guided refinement raises DS from 76.3 to 80.0, SR from 77.3% to 83.4%, and Rec.-SR from 62.5% to 67.4%, and lowers Col. from 0.200 to 0.145. Compared with discrete skills, SR rises by 6.4 points. When a proposal is rejected, refinement asks for another one and excludes the rejected primitive family, so the system can switch maneuvers before executing a trajectory that failed verification. Some of this gain could simply come from the extra attempts. To test whether the improvement comes merely from additional attempts, we include a retry baseline with the same maximum budget of three VLM calls per recovery episode, including the initial call. Each retry uses the unchanged initial input, without additional information about previous attempts or their outcomes. The retry baseline achieves 77.8 DS and 77.3% SR, whereas feedback-guided refinement achieves 80.0 DS and 83.4% SR. This comparison supports the benefit of feedback-guided revision beyond repeated generation under the same VLM-call limit. It does not carry over to collisions: Col. is 0.145 with refinement versus 0.125 with retries. Rec.-SR measures local returns to normal driving, while SR measures completed routes, so the two gains are separate evidence.

A.2 Recovery Traces and Failure Cases

Changing actions does not correct a poor understanding of the scene. In Figure 6, Qwen3.5-9B switches from creeping to a left swerve and subsequently returns to creeping, yet the rollout ends with a collision and a timeout at 72.9% route completion. During these actions, the model incorrectly interpreted the motion of the blocking vehicle, the correct bypass direction, and the drivable area. This suggests that successive decisions do not establish a reliable account of the constraints preventing progress. An alternative action can therefore repeat the underlying planning error even when its primitive differs from the previous attempt. The useful distinction is whether a retry incorporates evidence about why the previous attempt was ineffective and which alternatives remain feasible. Accordingly, retry count and action diversity alone are insufficient measures of recovery quality; the decision sequence must show how execution evidence changes the next proposal.

An intermediate action can make a subsequent maneuver feasible. Opus 5 initially selects a right bypass, responds to unsuccessful forward and lateral attempts by reversing, and then retries the right bypass before completing the route without collisions. This sequence shows that an initial

Table 4: **Recovery mechanism ablations.** All variants use Opus 5, share the recovery-triggering configuration, verifier threshold, and controller settings. Controlled fallback is the "+ Scene context refinement" configuration of Table 3: when no nominal proposal is admitted, it applies conservative braking instead of recovery planning. Because it executes no recovery primitives other than braking, it produces no recovery events and Rec.-SR is undefined (–). Skill denotes discrete primitive selection, Param denotes parameter selection, and Feedback denotes verifier-guided pre-execution revision. Budget-matched retry and verifier-guided refinement each allow at most three VLM calls per recovery episode; retries reuse the initial input without incorporating feedback. Col. is the mean number of collision events per route. The controlled-fallback row scores routes reaching the 180 s simulation cap as scenario timeouts (Appendix F.2).

Variant	Skill	Param	Feedback	DS ↑	SR (%) ↑	Col. ↓	Rec.-SR (%) ↑
Controlled fallback	–	–	–	76.0	74.0	0.050	–
+ Skill recovery	✓	–	–	76.3	77.0	0.125	61.7
+ Parameterized recovery	✓	✓	–	76.3	77.3	0.200	62.5
+ Budget-matched retry	✓	✓	–	77.8	77.3	0.125	63.5
+ Verifier-guided refinement	✓	✓	✓	**80.0**	**83.4**	0.145	**67.4**

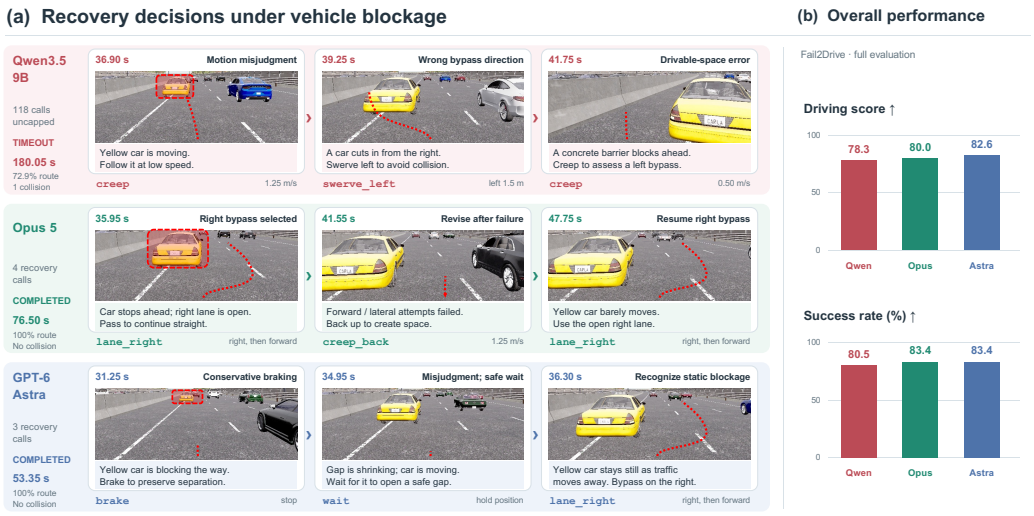


Figure 6: **Recovery decisions under vehicle blockage.** Qwen3.5-9B changes its actions while repeatedly misinterpreting motion and scene constraints, and the rollout ends in timeout with a collision. Opus 5 responds to unsuccessful maneuver attempts by reversing before retrying a right bypass. GPT-6 Astra initially brakes and waits, then revises its motion interpretation and selects a bypass. Both Opus and Astra complete the displayed route without collisions. Text annotations summarize the models' stated scene interpretations. The right panel reports aggregate benchmark performance.

unsuccessful bypass may not be the incorrect maneuver; as we saw, it may put the vehicle in a more favorable position to perform a reverse. Although reversing may reduce progress, it allows the model to eventually complete a more favorable bypass. This implies that recovery primitives should be assessed partly by the conditions they create for subsequent actions, rather than solely by their immediate contribution to route completion.

Conservative actions allow poor scene judgments to be corrected. GPT-6 Astra also misjudges the blocking vehicle's motion, but it first brakes and waits before recognizing the persistent blockage and selecting a right bypass. This shows that recovery capability is determined not only by how accurately a model interprets the scene, but also by which actions it selects under incorrect interpretations and whether those actions give the system an opportunity to revise them. A temporary motion misjudgment can remain recoverable when the vehicle stays distant from the object,

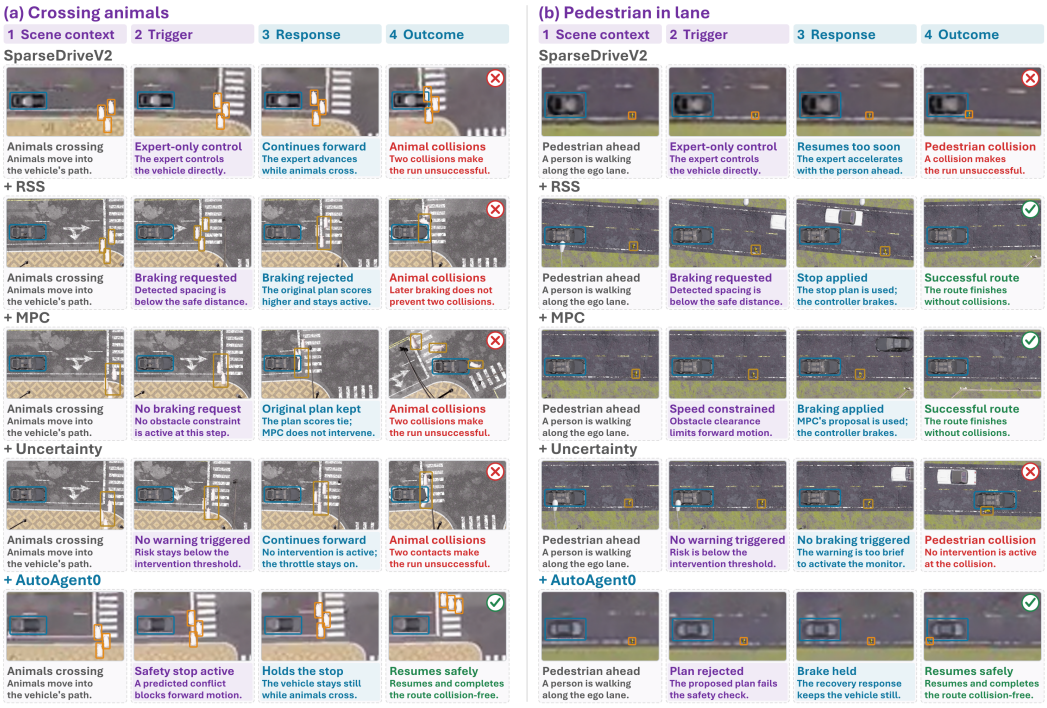


Figure 7: **Runtime decisions in two safety-critical encounters.** Rows compare SparseDriveV2, RSS, MPC, uncertainty monitoring, and AUTOAGENT0 across scene context, intervention trigger, selected response, and outcome. **(a)** Crossing animals: only AUTOAGENT0 completes without collisions. **(b)** Pedestrian in lane: RSS, MPC, and AUTOAGENT0 complete without collisions; SparseDriveV2 and uncertainty monitoring record a pedestrian collision. Blue outlines identify the ego vehicle; gold outlines mark hazards.

whereas committing to a maneuver that closes that distance under the same uncertainty can expose it to a collision. However, waiting only helps if the model keeps reassessing the scene; otherwise it stays stopped behind a persistent blockage until it times out. Good recovery therefore combines cautious actions with a timely switch to another maneuver once the evidence changes.

A.3 QUALITATIVE COMPARISON WITH TRADITIONAL RUNTIME MONITORS

Figure 7(a) distinguishes missed interventions from rejected responses. RSS requests braking, but the displayed proposal loses arbitration to the higher-scoring nominal plan; later braking does not prevent collisions. MPC has no active obstacle constraint at the shown step, and uncertainty monitoring does not trigger during the encounter. AUTOAGENT0 instead suppresses forward motion and maintains a stop while the animals cross. The distinction is how safety evidence affects execution: detecting a conflict is insufficient if the resulting correction can lose arbitration, whereas enforcing a stop prevents continued motion into the conflict.

In Figure 7(b), the uncertainty warning does not persist for the three consecutive updates required to trigger intervention, and the vehicle collides with the pedestrian. RSS, MPC, and AUTOAGENT0 complete without collisions; AUTOAGENT0 rejects a proposed trajectory and holds the brake before resuming. Here, no alternative path is needed, and the failure arises because candidate-score uncertainty does not produce a sustained warning, while AUTOAGENT0's trajectory checks provide a separate basis for rejecting forward motion without waiting for the expert's scores to indicate uncertainty.

Table 5: **VLM backbone comparison on Fail2Drive**. Rec.-SR denotes recovery success rate.

VLM backbone	DS ↑	SR (%) ↑	Col. ↓	Rec.-SR (%) ↑
Qwen3.5-9B (Qwen Team, 2026)	78.3	80.5	0.210	64.6
Opus 5 (Anthropic, 2026)	80.0	83.4	0.145	67.4
GPT-6 Astra (OpenAI, 2026)	82.6	83.4	0.125	71.0

A.4 VLM Backbone Comparison

Backbone choice affects both task completion and execution quality. Table 5 shows that replacing Qwen3.5-9B with Opus 5 increases DS by 1.7 points and SR by 2.9 percentage points, while reducing Col. from 0.210 to 0.145. The higher completion rate therefore accompanies fewer collisions, indicating that the gain is not simply a consequence of more aggressive execution. The traces in Section A.2 illustrate a plausible explanation: revising scene interpretations and unsuccessful maneuvers can restore progress while avoiding repeated commitments to unsuitable actions. GPT-6 Astra further increases DS by 2.6 points and reduces Col. to 0.125, despite matching Opus 5's SR of 83.4%. Because SR records binary route outcomes while DS incorporates completion and infraction penalties, this difference shows that a plateau in success rate can conceal improvements in execution quality. Although GPT-6 Astra achieves the highest DS, we use Opus 5 as the default backbone throughout the paper so that all ablations and comparisons share a single VLM configuration and remain directly comparable.

Local recovery gains must be assessed through subsequent route outcomes. Astra improves Rec.-SR over Opus 5 from 67.4% to 71.0% without increasing aggregate route success. The metrics evaluate different stages of execution: a recovery event ends after a sustained return to nominal driving, whereas route success requires completing the remaining task. A locally successful recovery can consequently precede another blockage, collision, or timeout; different backbones may also encounter different recovery-event populations. Thus, a higher Rec.-SR does not directly establish that more routes were rescued, and equal SR does not imply that the same routes succeeded. This distinction suggests a concrete way to assess recovery quality: track the final outcomes of routes that recover, and distinguish recurrence of the original blockage from failures caused by subsequent hazards. Such an analysis would reveal whether a backbone merely restores nominal execution temporarily or places the vehicle in a state from which the task can be completed reliably.

A.5 Scenario Breakdown and Generalization

Gains are concentrated in scenarios that require behavioral adaptation. Figure 4(a) shows the largest SR improvement in the Behavior group: DS increases by 19.8 points and SR by 40.0 percentage points. Thus, the increase in successful routes is much larger than the change in DS. This pattern is consistent with resolving failures that averages can obscure, including blockages after substantial progress and unsafe executions that still traverse much of the route. The Visual–Longitudinal group exhibits a related pattern: DS increases by 17.0 points and SR by 23.3 percentage points. Here too, the benefit is expressed primarily in successful and penalty aware execution.

Improved success does not imply uniform improvement. For Visual–Lateral scenarios, DS and SR increase by 13.6 points and 18.0 percentage points, respectively, but both remain the lowest of the four groups. The Robustness group has a smaller DS increase of 4.3 points and unchanged aggregate SR. The current results therefore do not show an increase in successful routes for every scenario group. The runtime's benefits depend on the type of failure encountered, and some conditions remain outside the capabilities demonstrated by the present recovery mechanisms.

Recovery includes deciding when to move and when to wait. The two Behavior examples in Figure 8 illustrate complementary responses. In the construction case, the standalone expert stalls at 47.9% completion, whereas a verified bypass completes the route. In the road-image wall case, the standalone execution collides; the augmented system detects the obstacle, waits, and resumes after clearance. The first case requires an alternative path, while the second requires withholding motion until the environment permits progress. A recovery policy that treats all low-progress states

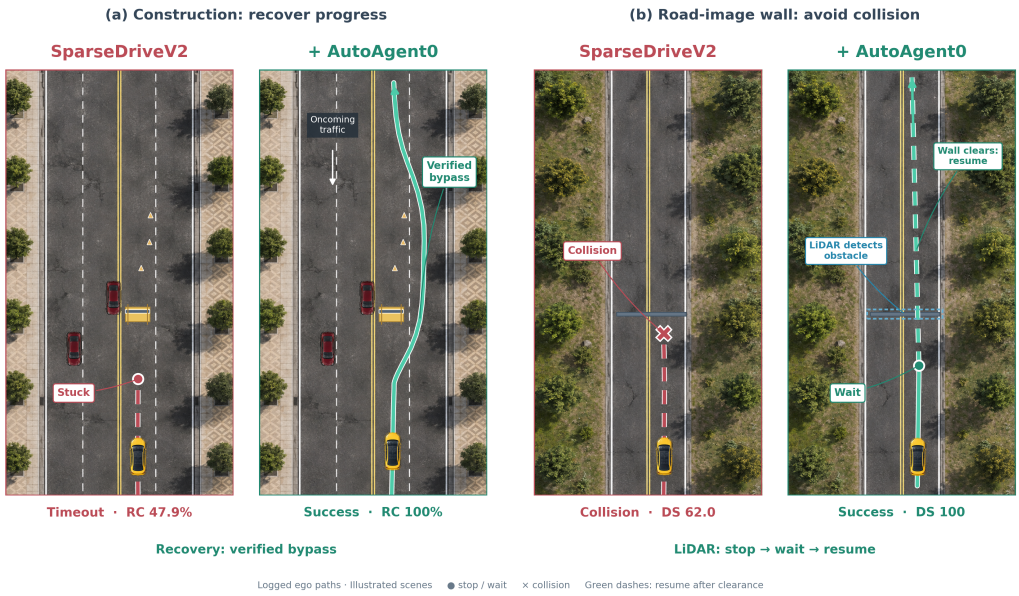


Figure 8: **Qualitative examples of complementary runtime behaviors.** Left: a verified bypass resolves a construction blockage. Right: obstacle detection supports stopping before a road-image wall and resuming after it clears. The examples illustrate improvements through both active maneuvering and appropriate waiting.

as a reason to maneuver would fail to distinguish these situations. Scene context determines whether progress should be restored through action or through waiting.

Generalization gains exceed in-distribution gains. Using the split-level values in Table 1, the DS generalization gap, $\mathrm{DS}_{\mathrm{ID}} - \mathrm{DS}_{\mathrm{Gen}}$, changes from 3.5 to -0.8 for UniAD, from 7.1 to 1.5 for SparseDriveV2, and from 9.7 to 6.0 for BridgeDrive. The corresponding DS gains on in-distribution versus generalization routes are 20.9 versus 25.2 points for UniAD, 10.7 versus 16.3 points for SparseDriveV2, and 0.1 versus 3.8 points for BridgeDrive. These differences are computed from the rounded values reported in the table.

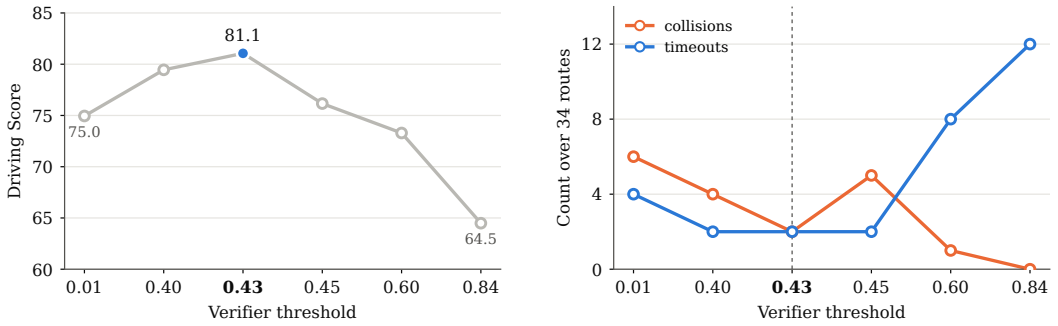
The gap reductions therefore accompany improvements on both splits, rather than degradation of in-distribution performance. BridgeDrive is particularly informative: its already high in-distribution performance changes little, while its generalization SR increases from 75.0% to 85.3%. This suggests that runtime augmentation can address residual long-tail failures even when ordinary-route performance leaves limited room for improvement. UniAD's slightly negative final gap describes these evaluated splits; it should not be interpreted as evidence that arbitrary distribution shifts become easier.

A.6 Verifier Threshold Sensitivity

Results and interpretation. Table 6 and Figure 9 show that stricter admission does not monotonically improve closed-loop outcomes: DS and SR both peak at 0.43 and fall on either side. Below 0.43, the verifier admits plans that collide, and the collision rate at 0.01 is three times that at 0.43. At the stricter thresholds, the verifier rejects the lateral maneuvers needed to get around an obstacle, so the ego stalls until the simulation cap. The rejection rate and the timeout count rise together past 0.45, and by 0.84 more than a third of the routes time out. The threshold therefore trades collisions against stalls, and 0.43 is where both failure modes are near their minimum. We use this value for AutoAgent0.

Table 6: **Verifier threshold sweep (17 scenario families).** One Base/Generalization route pair per Fail2Drive scenario family (34 routes): 17 pairs of routes are drawn uniformly at random within their families. Full AUTOAGENT0 runtime with SparseDriveV2, 180 s simulation cap; routes reaching the cap are scored as scenario timeouts (Appendix F.2). HM is the harmonic mean of DS and SR. Timeouts counts routes that reach the simulation cap. Rejected is the fraction of control ticks at which the verifier rejects the selected proposal and triggers recovery.

Threshold	DS ↑	SR (%) ↑	HM ↑	Col. ↓	Timeouts ↓	Rejected (%)
0.01 (hard gates only)	75.0	70.6	72.7	0.176	4	1.8
0.40	79.5	85.3	82.3	0.118	2	0.7
0.43 (ours)	**81.1**	**88.2**	**84.5**	0.059	2	1.1
0.45	76.2	82.4	79.1	0.147	2	1.0
0.60 (TTC becomes a hard gate)	73.3	76.5	74.9	0.029	8	4.2
0.84 (any soft defect rejects)	64.5	64.7	64.6	0.000	12	8.6



(a) Driving Score

(b) Failure modes

Figure 9: **Verifier threshold sweep.** Same 34-route sweep as Table 6. (a) Driving Score peaks at 0.43 and falls on both sides. (b) Collision events and routes that reach the simulation cap, counted over the 34 routes: collisions dominate below 0.43 and timeouts above it.

B RUNTIME EFFICIENCY AND LATENCY

B.1 MODULE-LEVEL LATENCY

We profile the execution time of individual modules to characterize the computational cost of AUTOAGENT0. Table 7 reports local module latencies measured during closed-loop driving. Nominal planning and perception inference have median latencies of 397.4 and 303.9 ms, respectively. Batch trajectory verification takes 78.6 ms, while single-candidate verification takes 12.0 ms. Recovery temporal search takes 468.6 ms per invocation, and the tracking controller takes 3.3 ms. The contribution of these operations to overall runtime also depends on their invocation frequency, which we examine in the next subsection.

Profiling protocol. We use SparseDriveV2 as the nominal planner, the same adapted BridgeDrive detector used in the closed-loop evaluations as the perception module, and Opus 5 as the VLM backend. The profiling set contains 12 Fail2Drive routes, comprising one in-distribution route and one generalization route from each of six scenario types. Each route is evaluated once, yielding 12,289 control steps in total. Excluding the first 20 steps of each route leaves 12,049 steps for the reported statistics. This exclusion provides a common initial warm-up window but does not guarantee that modules first invoked later have already warmed up. We measure elapsed wall-clock time at module boundaries, including inter-process communication, and record initialization separately. The experiments use shared NVIDIA L40S GPUs, with at most one profiling episode per GPU at a time. The reported latencies therefore characterize the observed shared-resource conditions.

Table 7: Local module wall-clock latency across 12 profiling routes. Statistics pool completed calls after excluding the first 20 control steps of each route. Timings include inter-process communication and nested local operations, so rows are not additive. VLM request latency is measured separately in Table 8.

Module	Calls	Mean (ms)	Median (ms)	P95 (ms)
Nominal planning	9,227	598.9	397.4	1,326.7
Perception inference	3,017	460.5	303.9	939.4
Perception update without inference	9,032	112.9	106.2	169.1
Perception fusion and refinement	3,017	178.2	169.7	344.3
Scene-context resolution	12,049	0.21	0.19	0.37
Batch trajectory verification	9,227	80.2	78.6	153.1
Single-candidate verification	15,439	13.9	12.0	31.4
Recovery temporal search	68	503.5	468.6	1,100.2
Tracking controller	12,049	3.28	3.27	6.07
Post-step visualization and logging	12,049	82.7	66.4	197.3

Table 8: Standalone VLM request latency on the same 20 saved recovery contexts. Each model processes one request per context. Qwen runs locally on a shared L40S, while Astra and Opus use remote APIs. The measurements exclude model loading but include input preparation and response parsing.

Model	Requests	Mean (s)	Median (s)	P95 (s)
Qwen3.5-9B	20	8.33	8.89	9.68
GPT-6 Astra	20	4.72	4.36	6.82
Opus 5	20	4.77	4.69	6.39

VLM request latency. We separately measure VLM request latency by replaying the same 20 saved recovery-decision contexts through Qwen3.5-9B, GPT-6 Astra, and Opus 5. The contexts are drawn from 11 of the profiling routes. Each context preserves the original task prompt, system prompt, and two front-camera images, ordered from the historical observation to the current observation. Each model processes all 20 contexts serially, without running CARLA or the driving pipeline. Model loading is excluded, while the first request is retained; no additional warm-up requests or automatic retries are used. We time the complete request call, including input preparation and response parsing. For API backends, this measurement also includes network and service latency.

Table 8 summarizes the results. Median request latency is 8.89 s for Qwen3.5-9B, 4.36 s for GPT-6 Astra, and 4.69 s for Opus 5. All 60 requests return parseable JSON without reported request errors or output truncation. These results characterize recovery-decision requests; stop-sign and semantic-takeover checks are not included in this replay experiment.

VLM configuration and measurement scope. Qwen3.5-9B uses BF16 weights, greedy decoding, and disabled thinking, without quantization or CPU offloading. Its attention implementation uses SDPA, while the DeltaNet layers use the PyTorch implementation because the optional accelerated kernels are unavailable in the measurement environment. Opus 5 uses low effort with thinking disabled, while GPT-6 Astra uses low reasoning effort. The output allowance is 800 tokens; the Astra adapter adds a 1,024-token reasoning reserve, giving a combined output cap of 1,824 tokens. The existing API adapters are retained, including Astra's additional JSON-format instructions and JSON output mode. Consequently, these measurements compare the specified inference configurations rather than identical decoding policies or hardware. The API measurements do not isolate model computation, and Qwen's local measurements include contention from other GPU workloads.

B.2 Invocation Frequency and Waiting Behavior

Per-call latency alone does not determine the runtime cost of a module. We consider invocation frequency, execution behavior while results are pending, and the age of the observations supporting an applied decision.

Invocation frequency. Over the 12,049 control steps retained for the local latency analysis, nominal planning and batch verification are each invoked 9,227 times. Perception inference is invoked 3,017 times, while the remaining 9,032 steps use perception updates without a new inference call. Recovery temporal search is invoked 68 times across 11 of the 12 routes, corresponding to approximately 0.56 calls per 100 control steps. Thus, although recovery search has a substantial per-call cost, it is invoked much less frequently than nominal planning or perception inference in this sample. Single-candidate verification is invoked 15,439 times, reflecting that a control step may contain multiple verification calls.

The full closed-loop traces, including the initial 20 steps of each route, contain 60 completed recovery VLM calls, 17 completed stop-sign checks, and seven completed semantic-takeover checks. These counts refer to Opus 5 calls in the closed-loop experiment, rather than the standalone replay requests in Table 8. They exclude requests that remain unfinished when an episode ends. Their observation window includes the initial steps excluded from the local module statistics above.

Waiting behavior. In the synchronous Fail2Drive execution path, recovery VLM requests block the current agent step. The evaluator does not advance the simulation while a recovery request is pending. Thus, a several-second wall-clock request does not correspond to the same duration of simulated driving or braking. After the request returns, recovery candidates are verified using the scene context of the same agent step, and the returned control is applied at the next simulation tick. Object-level semantic review follows a separate asynchronous path, with returned decisions checked for recency and track consistency before use. In the closed-loop Opus 5 traces, the median backend round-trip time is 4.18 s for recovery requests and 2.95 s for pooled stop-sign and semantic-takeover requests.

B.3 Route-level Efficiency

We measure the closed-loop cost of adding AutoAgent0 to SparseDriveV2 on the same 200 Fail2Drive routes. AutoAgent0 increases the median wall-clock time per route from 9.88 to 11.44 min ($+15.7\%$), while reducing the median simulated time from 44.4 to 38.9 s (-12.5%). Table 9 summarizes both quantities.

Protocol. Both evaluations use identical route IDs, and the recorded route lengths match for all 200 routes. Wall-clock time spans the first to the last agent step of the final completed attempt, excluding world loading and agent setup. Simulated time is the CARLA game time at the last step. The AutoAgent0 evaluation enforces a 180 s simulation limit, whereas the baseline evaluation did not, and 19 baseline routes ran for up to 372 s. We truncate those routes at 180 s of simulated time and use the wall-clock time at that step.

Results. AutoAgent0 takes longer in wall-clock time on 132 of 200 routes, with a median per-route ratio of $1.17\times$. The mean increases by only 2.6% because a few slow baseline routes dominate the baseline mean. Simulating one second of driving takes 16.61 s of wall-clock time with AutoAgent0, compared with 14.85 s for the baseline ($+11.8\%$ pooled). Shorter episodes partly offset this per-step overhead: AutoAgent0 finishes in less simulated time on 148 routes, and nine routes tie at the 180 s limit.

Scope. The two evaluations ran as separate experiments. Each route used one GPU and a dedicated CARLA server in both runs, so we compare per-route times rather than total experiment duration. The runs nevertheless differ in host and concurrency: about five routes ran simultaneously for AutoAgent0 on average, compared with about two for the baseline, and part of the baseline run overlapped a period of GPU contention. The reported overhead is therefore an estimate from existing runs, not a controlled timing benchmark.

Table 9: Per-route runtime on the same 200 Fail2Drive routes under a matched 180 s simulation budget. Wall-clock time is in minutes and simulated time in seconds. Wall-clock per simulated second is the real time needed to simulate one second of driving; the pooled rate divides total wall-clock by total simulated time. Δ is the relative change of AUTOAGENT0 in percent.

Quantity	Statistic	SparseDriveV2	+ AutoAgent0	Δ
Wall-clock (min)	Mean	15.49	15.90	$+2.6$
	Median	9.88	11.44	$+15.7$
	Total (h)	51.6	53.0	$+2.6$
Simulated (s)	Mean	62.6	57.4	-8.2
	Median	44.4	38.9	-12.5
	Routes at 180 s	19	21	–
Wall-clock per simulated second (s)	Pooled	14.85	16.61	$+11.8$
	Median	12.13	17.17	$+41.5$

C RUNTIME ARCHITECTURE AND SCENE CONTEXT REFINEMENT

C.1 RUNTIME CONTROL FLOW

Algorithm 1 summarizes the closed-loop runtime. At each step, the runtime updates its memory with execution feedback and refines the scene context. When a new planning decision is needed, nominal proposals are checked first; recovery search is invoked only if all nominal candidates are rejected. The search uses verifier feedback to revise recovery proposals within a bounded budget, committing a proposal only after it passes verification. An ongoing recovery maneuver is advanced under execution guards before a new planning decision is made. If no trajectory is available, the runtime applies conservative braking.

Here, ℓ denotes a recovery primitive and Θ a set of parameter vectors within its allowed domain. GENERATECANDIDATES applies the primitive's trajectory generator to each parameter vector using the refined context $\widetilde{c}_t$. The variables *candidates* and *accepted* are reused for nominal and recovery proposals, while *feedback* records the verifier's assessment.

C.2 SCENE CONTEXT REFINEMENT

Geometric Refinement and Disagreement Detection. The perception pipeline supports bounded refinement of predicted object boxes using associated LiDAR and radar returns. LiDAR refinement uses nearby non-ground points, while radar refinement applies to eligible forward vehicle and static-object boxes. An optional disagreement module compares the original detector boxes with LiDAR evidence from the same sensor capture. It identifies two types of candidates: detected objects with insufficient LiDAR support, and non-ground LiDAR clusters not explained by current detections. These disagreements identify objects for further inspection rather than directly establishing false positives or false negatives. Invalid LiDAR evidence does not count as evidence that an object is absent.

Optional Object-level Semantic Validation. When enabled, the disagreement module associates candidates across observations using their positions and dimensions, and accumulates disagreement observations within a sliding time window. Candidates become eligible for VLM review after reaching a configured observation count, subject to request cooldowns and backend availability. The VLM receives a camera image with the candidate's projected location marked, an unmarked crop of the same region, and a description of the sensor disagreement. It returns a `keep`, `drop`, or `uncertain` decision together with a confidence value and a short explanation. Requests run asynchronously, allowing the control loop to continue while a review is pending.

Applying Validated Corrections. A returned decision is accepted only if it satisfies the confidence threshold, remains sufficiently recent, and still matches the active candidate. An accepted `keep` decision retains a matching object or adds the candidate to the obstacle list if no match is present. An accepted `drop` decision removes matching objects from that list. Uncertain, malformed, or expired responses do not introduce a new correction. Cached decisions expire after a

Algorithm 1 AUTOAGENT0 Runtime with Scene Context Refinement and Agentic Recovery Planning

Require: Nominal policy π_{N}, route q, scene-refinement skills $\mathcal{F}$, action primitive library $\mathcal{P}$, rule-based verifier V, search budget B

Initialize $memory \leftarrow \varnothing$, $recovery_state \leftarrow$ None, and $t \leftarrow 0$
while the route is active **do**
  Receive observation o_t and update history h_t
  Record execution feedback from o_t and $recovery_state$ in $memory$
  $c_t \leftarrow \textsc{BuildContext}(h_t, q, memory)$
  Scene Context Refinement
  $\widetilde{c}_t \leftarrow \textsc{SceneContextRefinement}(\mathcal{F}, c_t)$
  $\tau_t \leftarrow$ None
  if $recovery_state \neq$ None **then**
    Advance committed recovery using $\widetilde{c}_t$ and execution guards; update τ_t, $recovery_state$, and $memory$
  end if
  if $\tau_t =$ None **then** ▷ A new planning decision is needed
    $candidates \leftarrow \textsc{ProposeNominal}(\pi_{\mathrm{N}}, c_t)$
    Rule-based Verification
    $(accepted, feedback) \leftarrow \textsc{VerifyCandidates}(V, candidates, \widetilde{c}_t)$
    if $accepted \neq \varnothing$ **then**
      $\tau_t \leftarrow \textsc{SelectBest}(accepted)$ ▷ Verified nominal policy π_{N}^{V}
    else ▷ All nominal candidates were rejected
      Recovery Planning via Agentic Parameter Search
      Record $candidates$ and $feedback$ in $memory$
      for $j = 1, \ldots, B$ **do**
        $(\ell, \Theta) \leftarrow \textsc{SelectAndRefine}(\mathcal{P}, \widetilde{c}_t, feedback, memory)$
        $candidates \leftarrow \textsc{GenerateCandidates}(\ell, \Theta, \widetilde{c}_t)$
        $(accepted, feedback) \leftarrow \textsc{VerifyCandidates}(V, candidates, \widetilde{c}_t)$
        Record ℓ, Θ, $candidates$, and $feedback$ in $memory$
        if $accepted \neq \varnothing$ **then**
          $\tau_t \leftarrow \textsc{SelectBest}(accepted)$
          $recovery_state \leftarrow \textsc{CommitRecovery}(\ell, \tau_t, \widetilde{c}_t)$
          break
        end if
      end for
    end if
  end if
  if $\tau_t =$ None **then**
    $\tau_t \leftarrow \textsc{ConservativeBraking}(o_t)$ ▷ No proposal admitted
  end if
  $\textsc{ExecuteSegment}(\tau_t)$ ▷ Tracking controller; feedback arrives in o_{t+1}
  $t \leftarrow t + 1$
end while

configured lifetime, and renewed agreement between the sensors invalidates an earlier decision for the corresponding track. The resulting object list supplies the corrected obstacle evidence used in subsequent trajectory verification.

D AGENTIC RECOVERY PLANNING

D.1 PRIMITIVE LIBRARY AND TRAJECTORY GENERATION

Table 10 summarizes the fifteen primitives in the recovery library. Each primitive defines a maneuver that the deterministic runtime converts into a candidate trajectory. The table reports default durations and parameter ranges; the available actions and effective parameter ranges may be restricted by the runtime configuration and current scene. For example, the runtime excludes `swerve_left` and `swerve_right` when the ego speed is below 1 m/s.

The VLM selects from the available primitives, while the runtime handles trajectory generation and verification. Parameter search is described in Section D.2, and execution-time safety checks are described in Section D.4.

Table 10: **Recovery primitive library.** Durations and parameter ranges describe the default action specifications. Actual execution may end early or use a configuration-dependent duration. A dash indicates that no parameter range is listed in the default specification.

Primitive	Behavior	Default duration	Parameter range
brake	Brake to a full stop	1.0 s	–
wait	Remain stopped	1.0 s	–
creep	Crawl straight forward at low speed	1.5 s	speed 0.5–2.0 m/s
creep_back	Stop, then back straight up with heading preserved	3.0 s	speed 0.5–2.0 m/s
swerve_left	Swerve left around an obstacle, then stop	2.5 s	lateral 0.5–2.5 m
swerve_right	Swerve right around an obstacle, then stop	2.5 s	lateral 0.5–2.5 m
back_left	Stop, reverse while steering left, then stop	5.0 s	lateral 0.3–1.5 m
back_right	Stop, reverse while steering right, then stop	5.0 s	lateral 0.3–1.5 m
back_bypass_left	Reverse, then drive forward around the obstacle on the left	6.0 s	lateral 1.0–3.0 m
back_bypass_right	Reverse, then drive forward around the obstacle on the right	6.0 s	lateral 1.0–3.0 m
three_point	Perform a bounded three-point turn	6.0 s	–
lane_left	Change one lane to the left and finish aligned	3.0 s	–
lane_right	Change one lane to the right and finish aligned	3.0 s	–
lane_bypass_left	Borrow the adjacent left lane, pass a static obstacle, and return to the original lane	6.0 s	–
lane_bypass_right	Borrow the adjacent right lane, pass a static obstacle, and return to the original lane	6.0 s	–

D.2 PARAMETER SEARCH AND VERIFIER-GUIDED REFINEMENT

Recovery planning starts with the VLM selecting a primitive based on the scene context and the reason for nominal-plan rejection. The runtime then samples parameters within the selected primitive's allowed range and generates candidate trajectories. Each candidate is evaluated by the rule-based verifier described in Appendix E, and the highest-scoring accepted candidate is selected for execution.

If no candidate passes verification, the rejection feedback is returned to the VLM to guide another primitive selection. This process continues within a bounded search budget. If no acceptable trajectory is found, the runtime falls back to conservative braking.

D.3 VLM PROMPTS AND INVOCATION PROTOCOL

The VLM receives front-camera observations, the route instruction, scene context, and verifier feedback, together with the available recovery primitives. The prompt asks it to identify the blockage, determine a suitable escape direction, and select a recovery primitive. When earlier attempts have failed, their outcomes are included to inform the next decision.

The response specifies a primitive and a brief explanation. The runtime checks that the selected primitive belongs to the available set before performing parameter search. Numerical trajectory generation and verification are handled by the runtime. Invalid responses or exhausted invocation budgets trigger the fallback procedure.

Prompt format. We provide a condensed template of the recovery-plan prompt below. Scene-dependent fields are filled at runtime, and front-camera images are supplied as visual inputs. The available action list is filtered by the runtime before invocation. The VLM identifies the blockage, selects an escape direction, and proposes candidate primitives for subsequent sampling and verification.

```
[System]
You are a driving recovery planner.
Use only exact primitive IDs supplied in the user prompt.
Return only valid JSON.

[Visual input]
<Front-camera image(s)>

[Driving context]
Route direction: <route instruction>
Road context: <lane and road information>

[Previous recovery result]
<Previous action and its outcome>

[Recovery memory]
<Relevant earlier attempts and feedback>

[Safe primitive candidates]
<Available primitive IDs, descriptions, and speed ranges>

[Task]
1. Describe the visible blockage in one short sentence.
2. Choose an escape direction:
   forward, left, right, back, or stop.
3. Return up to five candidate primitives with probabilities
   and brief reasons.
4. Use only the supplied primitive IDs.
5. Specify target_speed_mps only when supported;
   otherwise use null.
6. If no candidate is appropriate, choose stop and
   return an empty candidates list.

[Output]
Return JSON only, without Markdown fences.
```

Response format. The following example illustrates the response structure. The action must belong to the supplied candidate list.

```
{
  "front_blockage": "A static obstacle blocks the route lane.",
  "escape_direction": "left",
  "candidates": [
    {
      "primitive": {
```

```
            "action": "lane_left",
            "target_speed_mps": null
          },
          "probability": 1.0,
          "reasoning": "The adjacent left lane provides space."
        }
      ]
    }
```

The runtime validates the response and uses the candidate probabilities to allocate the sampling budget. Generated trajectories are verified before execution. When another selection is requested, feedback from rejected attempts is included in the next prompt.

D.4 Execution, Interruption, and Fallback

Once a recovery trajectory passes verification, the runtime commits it to the tracking controller. The trajectory is anchored in the world frame and advanced as the vehicle moves, allowing multi-stage maneuvers to proceed without restarting at each control step.

During execution, the runtime monitors progress and applies maneuver-specific safety guards using updated scene observations. These execution guards are distinct from the full admission check used during candidate selection. A detected hazard or stalled maneuver can interrupt execution and trigger braking or replanning.

When the maneuver finishes, the runtime reassesses nominal proposals. Nominal control resumes when a proposal is admitted; otherwise, recovery planning is invoked again. If no admissible trajectory is available, the runtime applies conservative braking as a contingency response. This fallback is not conditioned on passing the PDMS admission threshold and does not carry the same admission assurance as a verified candidate.

E Rule-based Verifier

The rule-based verifier evaluates nominal proposals and recovery trajectories using the scene context supplied by the runtime. This section specifies the scoring rules and their admission criterion. Section A.6 evaluates threshold sensitivity in closed-loop driving, while Appendix F documents the experimental configuration and evaluation protocols.

Plan representation. The expert trajectory representation described here consists of 8 poses at 0.5 s intervals. The verifier densifies the plan to 0.1 s and adjusts its arc-length progress using the measured ego speed and acceleration and braking bounds of 3.5 and 4.0 m/s^2, respectively. This adjustment limits the progress assigned to sample k at time $k \cdot 0.1$ s according to these longitudinal bounds. The adjusted trajectory is used for safety checks, while ego progress is evaluated on the original proposal by default.

E.1 Scoring Function and Rule Definitions

Score. Our rule-based verifier adapts the PDM Score of NavSim (Dauner et al., 2024) to live CARLA state. For a candidate plan τ, it computes

$$\mathrm{PDMS}(\tau) = \mathrm{NC} \cdot \mathrm{DAC} \cdot \mathrm{TTC}_{\mathrm{VRU}} \cdot \frac{5\,\mathrm{TTC} + 5\,\mathrm{EP} + 2\,\mathrm{LK}}{12}, \tag{3}$$

where NC and $\mathrm{TTC}_{\mathrm{VRU}}$ are binary gates, DAC is a tiered gate in $\{0, 0.5, 1\}$, and TTC, EP, and LK are soft terms in $[0, 1]$. TTC and LK take binary values in the implementation, whereas EP varies continuously. The same function scores candidate batches during trajectory selection.

Gates. No collisions (NC) sweeps the ego bounding box along the adjusted plan against all detected agents. Vehicles are forecast with the kinematic bicycle model of World on Rails (Chen et al., 2021), with the CARLA parameters used by TransFuser++ (Jaeger et al., 2023), under held controls. Walkers classified as moving using a 0.3 m/s speed threshold are forecast with a minimum speed

of 1.5 m/s along their heading, while stationary walkers are held at their measured pose. Collisions within the configured veto horizons zero the NC gate: 1.5 s for vehicles and static objects and 3.0 s for pedestrians, each extended to the ego stopping time plus 0.5 s when that is longer. Drivable area compliance (DAC) queries the HD map at the checked plan endpoint and assigns a tier of 1.0 for any driving or parking lane, 0.5 on the shoulder, and 0.0 for off-road locations or sidewalks. The VRU veto ($\mathrm{TTC}_{\mathrm{VRU}}$) zeros the score when the TTC term fails because of a pedestrian. This additional gate prevents a pedestrian-related TTC failure from being compensated by the remaining soft terms, which could otherwise yield a score as high as $7/12 \approx 0.583$ and pass the admission threshold.

Soft terms. Time to collision (TTC) probes the plan at forward offsets of 0.2, 0.3, 0.5, 0.8, and 1.0 s, with surrounding agents predicted at the corresponding future times. An overlap within the applicable veto horizon at any of these offsets sets TTC to 0. Ego progress (EP) is the arc-length progress of the original plan along the route, divided by the progress expected at 5 m/s over the plan horizon and clipped to $[0, 1]$. Lane keeping (LK) is 0 when the checked plan endpoint is more than 1.5 m from the nearest driving lane centerline, and 1 otherwise.

Deviations from NavSim. Comfort was replaced with lane keeping (LK) because executed accelerations are set by the controller rather than the plan. Our drivable area compliance (DAC) calculation is tiered because CARLA exposes lane types rather than drivable area polygons. Collision gates are limited to a veto horizon instead of the full plan, and vehicles are forecast with a bicycle model instead of constant velocity. We add a VRU veto to place an emphasis on avoiding pedestrian collisions.

E.2 Admission Criterion and Score Bounds

Admission and runtime integration. A plan passes the score-based admission test when $\mathrm{PDMS} \geq \theta$, with $\theta = 0.43$ in the default configuration. Otherwise, the plan is rejected. When no nominal candidate is admitted, control passes to recovery planning when enabled, with controlled fallback used when no executable proposal is available. The runtime control flow is specified in Section C.1, and recovery execution, interruption, and fallback are described in Section D.4. Route-consistency checks impose additional conditions during candidate selection and are not terms in the PDMS equation. After a recovery trajectory is committed, maneuver-specific execution guards monitor its continuation using updated observations; these guards are distinct from the full admission check used during candidate selection.

Threshold choice. The default threshold of 0.43 lies just above the score boundary $5/12 \approx 0.417$. With TTC and LK binary and EP continuous, a TTC failure permits a score of at most $7/12 \approx 0.583$, while an LK miss permits at most $10/12 \approx 0.833$. Here, a zero-valued soft term denotes a complete failure of that term; a partially reduced EP is not treated as a binary defect. If any two of TTC, EP, and LK are zero, the score cannot exceed $5/12 \approx 0.417$. A threshold of 0.43 therefore rejects every such combination of two zero-valued soft terms. These upper bounds assume that the multiplicative gates equal one; a reduced DAC tier or a zero-valued gate can only lower the score. A threshold at or below $5/12$ can admit a plan with both TTC and LK equal to zero when EP is sufficiently high and the multiplicative gates equal one. A single zero-valued term does not guarantee admission, because the remaining terms must still provide sufficient score. For example, with TTC$= 0$, LK$= 1$, and all multiplicative gates equal to one, admission at 0.43 requires EP≥ 0.632. Thresholds above $7/12$ reject every plan with TTC$= 0$ and every zero-progress plan with EP$= 0$, while thresholds above $10/12$ also reject every plan with LK$= 0$. These score bounds characterize the admission rule but do not establish an optimal threshold for closed-loop driving. The empirical comparison of thresholds and its scenario-dependent outcomes are reported in Section A.6.

Scope of admission. Admission denotes satisfaction of the configured rules under the supplied scene estimate and motion forecasts, within the configured checking horizons. It does not require every soft term to be nonzero: a vehicle-related TTC failure can be compensated by the remaining soft terms when all multiplicative gates permit admission, whereas an NC failure or a pedestrian-related TTC failure vetoes the candidate. In the SparseDriveV2 configuration, DAC and LK evaluate only the checked endpoint, so passing these checks does not establish map compliance at inter-

mediate trajectory positions. These conditions do not establish continuous collision avoidance or drivable-area compliance over the entire trajectory or its subsequent closed-loop execution.

Configuration reference. The values above specify the verifier settings described in this section. The consolidated experimental configuration belongs in Appendix F, together with any experiment-specific overrides.

F Experimental Protocols and Reproducibility

F.1 Models, Sensors, and Hardware

Driving models. The runtime integrates pretrained SparseDriveV2, UniAD, and BridgeDrive through model-specific adapters. The SparseDriveV2 configuration uses the Bench2Drive model defined by `projects/configs/sparsedrive_stage2.py` and the checkpoint `sparsedrive_small_b2d_stage2.pth`. The UniAD adapter uses the Bench2Drive configuration `base_e2e_b2d.py` and checkpoint `uniad_base_b2d.pth`. The BridgeDrive adapter loads the `bridgedrive_k60` checkpoint directory through the LEAD implementation. Model inference is separated from the CARLA agent through worker processes, allowing each model to use its own software environment.

Vision-language models. The local VLM configuration specifies `Qwen/Qwen3.5-9B`. Separate API configurations specify `claude-opus-5` and `gpt-6-astra` for the Opus 5 and GPT-6 Astra variants, respectively. These recovery configurations use the front-camera view. Prompt construction, temporal image inputs, and recovery invocation are described in Section D.3.

Sensor inputs and perception. SparseDriveV2 uses its fixed six-camera Bench2Drive rig, covering the front, front-left, front-right, rear, rear-left, and rear-right views. In the SparseDriveV2 runtime configuration, the shared adapted BridgeDrive detector runs as a separate service with an additional front RGB camera, two LiDAR sensors, four radar sensors, IMU, and GNSS inputs. The additional front camera has a resolution of 384×384 and a field of view of $60°$. The reduced perception rig supplies the central BridgeDrive camera view and fills the two unavailable side-camera inputs with zeros. A separate configuration option enables all three BridgeDrive camera views. When BridgeDrive is itself the driving model, its adapter registers the three native RGB cameras at yaw angles of $-54.5°$, $0°$, and $54.5°$. Each of these cameras has a resolution of 384×384 and a field of view of $60°$.

The SparseDriveV2 sensor-based configuration obtains surrounding-object detections from BridgeDrive and enables radar- and LiDAR-based box refinement. It disables the configured sources of simulator object annotations while retaining HD-map lane information for verification and route handling. The perception confidence threshold is 0.35, inference is scheduled every four control ticks, and the maximum permitted detection age is 1.0 s. The scene-refinement procedures are described in Section C.2. The traditional monitors use the same adapted BridgeDrive detector; their intervention mechanisms are specified in Section F.4.

On Fail2Drive, the runtime uses the same adapted BridgeDrive detector when paired with SparseDriveV2, UniAD, or BridgeDrive as the driving expert. Simulator ground-truth scene context is used only in the explicitly labeled oracle ablation.

Control and candidate evaluation. The SparseDriveV2 configuration uses a 20 Hz control and replanning schedule in simulated time. This does not imply a wall-clock throughput of 20 Hz. Its candidate scorer uses a default top-30 native-proposal branch and enables a separate region-sampling branch with up to three proposals per region and sampling seed 0. The region-sampling seed is distinct from the simulator evaluation seed. The SparseDriveV2 configuration selects candidates using PDMS ranking rather than enabling the optional native-ranking-after-PDMS-gating mode. The BridgeDrive configuration instead sets the native-proposal limit to 60 and enables native ranking among proposals that pass the PDMS gate. The UniAD configuration produces six poses at 0.5 s intervals and performs a new model and orchestration pass every five control ticks. Between these passes, its cached trajectory is reprojected and checked on each control tick.

Table 11: **Verifier configuration for the SparseDriveV2 sensor-based setting.** Values are shared by the inspected local-VLM, Opus 5, and GPT-6 Astra configurations. Settings for other driving-model adapters may differ.

Parameter	Value
Admission threshold	0.43
Verification sampling interval	0.1 s
Vehicle/static-object NC horizon	1.5 s
Pedestrian veto horizon	3.0 s
Stop-conditioned horizon extension	Enabled
Reaction-time margin	0.5 s
TTC probe offsets	$\{0.2, 0.3, 0.5, 0.8, 1.0\}$ s
TTC debounce length	1 check
Pedestrian TTC veto	Enabled
Vehicle forecast model	Kinematic bicycle
Moving-pedestrian minimum forecast speed	1.5 m/s
Additional stationary-vehicle lateral margin	0.0 m
HD-map evaluation mode	Endpoint only
DAC mode	Tiered
Configured DAC horizon	1.0 s
Configured DAC exit grace	Enabled
LK deviation limit	1.5 m
Configured LK window	2.0 s
Re-timing acceleration bound	3.5 m/s^2
Re-timing braking bound	4.0 m/s^2
EP evaluated on re-timed trajectory	Disabled
Additional normal-mode EP gate	Disabled

Verifier configuration. Table 11 records the verifier settings shared by the inspected SparseDriveV2 sensor-based configurations, including the local-VLM, Opus 5, and GPT-6 Astra variants. The definitions of the scoring terms are given in Appendix E, and the threshold sensitivity experiment is reported in Section A.6. The table specifies configuration values rather than an additional set of scoring rules.

The collision-checking horizons are extended when the current stopping time plus the 0.5 s reaction margin exceeds their configured values. Endpoint-only HD-map evaluation passes only the final checked position to the DAC and LK calculations. Consequently, the configured DAC horizon, DAC exit-grace option, and LK window do not provide an additional along-trajectory check in this mode. The inspected UniAD configuration instead disables endpoint-only HD-map evaluation. The inspected BridgeDrive configuration sets an additional stationary-vehicle lateral margin of 1.0 m, whereas the SparseDriveV2 configuration uses 0.0 m. These differences should be distinguished from the shared numerical admission threshold.

Route-instruction settings. The inspected configurations enable the route-instruction scorer. Its endpoint-bearing classifier uses a straight-heading threshold of $10°$, a turn-heading threshold of $15°$, and a minimum forward displacement of 0.3 m. The scorer assigns weights of 1.0, 0.5, and 0.0 to matching, ambiguous, and contradictory directions, respectively. These settings belong to candidate selection and are separate from the PDMS soft-term weights.

Simulation and Hardware. We use the Fail2Drive distribution of CARLA 0.9.15 on a server equipped with eight NVIDIA L40S GPUs. Each concurrent route slot runs a dedicated CARLA server and evaluator, with GPU assignment and the number of simultaneous slots configured by the evaluation launcher. Local driving and perception models run on the assigned GPUs, while API-based VLM variants use external inference services. The execution conditions for the runtime comparison are detailed in Appendix B.

F.2 Fail2Drive Setup and Metrics

Fail2Drive (Gerstenecker et al., 2026) contains 100 paired routes covering 17 unseen long-tail scenario types. Each pair consists of an in-distribution base route and a shifted route with matched start, goal, route geometry, and non-target factors. We evaluate all 200 routes for the main result and report the paired generalization gap.

Our primary metric is the official driving score (DS), defined by the benchmark as route completion (RC, in percent) multiplied by the infraction penalty (IP). We also report route success rate (SR), where a route is successful only if the evaluator records no infraction other than minimum-speed and outside-route-lane events, so collisions, traffic-rule violations, scenario timeouts, blockages, route timeouts, and route deviations all count as failures. We also report the mean number of collision events per route (Col.). We also report HM, the harmonic mean of DS and SR. DS measures the safety–progress trade-off, RC separates progress from penalties, and SR exposes catastrophic closed-loop failures that can be obscured by averages. Each route is evaluated for at most 180 s of simulated time to bound evaluation cost. Routes that reach this limit without completing were stationary or near-stationary, and we score them as scenario timeouts: they count as unsuccessful for SR, and the benchmark's scenario-timeout penalty ($\times 0.7$) is applied to their infraction penalty when computing DS. For recovery analysis, we report the recovery success rate (Rec.-SR) over temporally grouped recovery events: consecutive retries form one event, which succeeds when a verifier-admitted action completes and nominal driving then remains healthy for five consecutive simulation seconds.

For SparseDriveV2, the controller and planner run at 20 Hz; UniAD instead runs a full model pass every five control ticks and re-verifies its cached trajectory on each tick. BridgeDrive is evaluated every four control ticks with a score threshold of 0.35 and a maximum detection staleness of 1 s. In the SparseDriveV2 sensor-based configuration, the end-to-end proposal branch retains the expert's top 30 candidates. For this configuration, the PDMS admission threshold is 0.43; the longitudinal re-timing bounds are $3.5\,\mathrm{m/s^2}$ for acceleration and $4.0\,\mathrm{m/s^2}$ for braking; the NC gate horizon is 1.5 s for vehicles and static objects and 3.0 s for pedestrians, each extended to the ego stopping time plus 0.5 s when that is longer; DAC and LK are evaluated at the checked plan endpoint; and TTC is probed at 0.2, 0.3, 0.5, 0.8, and 1.0 s.

F.3 NavSafe-∞ Setup and Metrics

Driving experts and perception source. On NavSafe-∞, we evaluate runtime augmentation of LTF, DiffusionDrive, and SparseDriveV2. For all three augmented experts, the runtime obtains scene observations from BridgeDrive's native perception module. The additional pedestrian and static-obstacle modules studied in Appendix G are not used in these NavSafe-∞ experiments. BridgeDrive supplies perception to the runtime, while the corresponding driving expert supplies nominal driving proposals.

For completeness we restate the metric definitions of NavSafe-∞ (Bao et al., 2026) used in Table 2. We refer the reader to the original paper for the event specification, the semi-reactive traffic protocol, and the validity checks that determine which rollouts are scored.

Each event begins after a two-second log-replay warm-up and runs until success, task failure, or timeout. Every policy is evaluated on the same preregistered seeds per event specification, and we report the mean across seeds. For a specification b, let $\mathcal{V}_b$ denote its rollouts that satisfy the benchmark's validity checks, and let $\mathcal{B}^+ = \{b : |\mathcal{V}_b| > 0\}$ be the specifications retaining at least one such rollout. Metrics average first over the rollouts in $\mathcal{V}_b$ and then over the specifications in $\mathcal{B}^+$, so that every specification contributes equally to the reported score.

Driving Score (DS). DS combines task-specific completion $\mathrm{EC}_{br} \in [0,1]$ with multiplicative penalties $p_q \in [0,1]$ for recorded infractions,

$$\mathrm{DS} = \frac{100}{|\mathcal{B}^+|} \sum_{b \in \mathcal{B}^+} \frac{1}{|\mathcal{V}_b|} \sum_{r \in \mathcal{V}_b} \mathrm{EC}_{br} \prod_{q \in \mathcal{I}_{br}} p_q. \tag{4}$$

Where partial completion is not meaningful for a task, EC_{br} is binary. Because the penalties multiply, one severe infraction bounds the score irrespective of how much of the task was completed.

Success Rate (SR). $s_{br} = 1$ only when rollout r satisfies its success predicate within the time budget and incurs no disallowed infraction. With $\mathrm{SR}_b = |\mathcal{V}_b|^{-1} \sum_{r \in \mathcal{V}_b} s_{br}$, the benchmark score is $\mathrm{SR} = 100 |\mathcal{B}^+|^{-1} \sum_{b \in \mathcal{B}^+} \mathrm{SR}_b$. SR admits no partial credit and therefore exposes categorical failures that DS can average away.

Driving Efficiency (DE). DE is the ratio of ego speed to nearby-traffic speed.

Comfort. Comfort is the fraction of segments satisfying the acceleration, jerk, yaw-rate, and yaw-acceleration bounds. DE and Comfort use the same hierarchical averaging as DS and SR and are undefined when their denominator is empty.

Ability scores (AS). Specification-level success rates are grouped by the four RoadSafe365 categories: Traffic Crashes (TC), Vulnerable Road User Crashes (VRUC), Traffic Violations (TV), and Traffic Incidents (TI) (Liu et al., 2026b). Writing $\mathcal{B}_k^+ = \{b \in \mathcal{B}^+ : p(\ell_b) = k\}$ for the specifications whose taxonomy leaf ℓ_b has parent category k,

$$\mathrm{AS}_k = \frac{100}{|\mathcal{B}_k^+|} \sum_{b \in \mathcal{B}_k^+} \mathrm{SR}_b, \qquad k \in \{\mathrm{TC}, \mathrm{VRUC}, \mathrm{TV}, \mathrm{TI}\}. \tag{5}$$

AS therefore inherits SR's binary criterion: progress, efficiency, comfort, and partial penalties cannot raise it. Table 2 reports the four category-specific ability scores.

F.4 Traditional Monitor Implementation Details

Common setup. All conditions execute the same frozen SparseDriveV2 checkpoint at 20 Hz and use the same adapted BridgeDrive detector. The detector provides surrounding-agent detections every four control ticks with a score threshold of 0.35 and a maximum detection age of 1 s. Simulator object annotations are not used. The monitor responses described below are candidate proposals. They replace the nominal trajectory only when they pass verification and achieve a higher PDMS score than the nominal proposal.

Evaluation routes. The uncertainty monitor requires threshold calibration. We reserve 40 routes for calibration and exclude them from the comparison for all methods. We evaluate all remaining 160 routes, comprising 80 In-Distribution/Generalization route pairs covering all 17 Fail2Drive scenario families. All three monitors use the same routes and seed 0, with one rollout per route. SparseDriveV2 and AutoAgent0 results are restricted to the same evaluation routes. Metrics are averaged separately over the 80 routes in each split, weighting each route equally.

RSS. The RSS wrapper evaluates the current longitudinal and lateral separation from each detected agent using a 1 s response time, maximum ego acceleration of $2.0\,\mathrm{m/s^2}$, minimum ego braking of $4.0\,\mathrm{m/s^2}$, assumed leading-agent braking of $8.0\,\mathrm{m/s^2}$, and 0.2 m minimum lateral clearance. An unsafe state triggers a bounded-braking proposal, with a stop proposed below $0.3\,\mathrm{m/s}$; the braking request is released after three consecutive safe updates.

MPC. The MPC wrapper assumes constant-velocity motion for detected agents and optimizes a longitudinal acceleration sequence over the native waypoint horizon at 0.5 s resolution. SLSQP minimizes path-progress, speed, control-effort, and acceleration-smoothness costs subject to acceleration in $[-4.0, 2.4]\,\mathrm{m/s^2}$, speed below $12\,\mathrm{m/s}$, and sensed-agent clearance constraints. It preserves the native path geometry and proposes $4.0\,\mathrm{m/s^2}$ braking when optimization is infeasible.

Predictive uncertainty. We compute an uncertainty score from the normalized entropy and top-two margin of SparseDriveV2's top 32 trajectory candidates. A threshold selected on a disjoint calibration split (0.923) triggers intervention after three consecutive high-risk updates and releases it after five low-risk updates. While intervention is requested, the monitor preserves the native path and proposes a $2.5\,\mathrm{m/s^2}$ deceleration. This score is a failure-detection heuristic rather than a calibrated probability or a formal estimate of epistemic uncertainty.

Metrics and validation. The driving-score gap is $\mathrm{Gap} = \mathrm{DS}_{\mathrm{ID}} - \mathrm{DS}_{\mathrm{Gen}}$, computed from unrounded means and expressed in score points. Positive values indicate lower driving scores on generalization routes. The gap must be interpreted alongside both scores, since a smaller gap can also result from worse in-distribution performance.

Validation. Accepted runs must have complete evaluator results, matching configuration and provenance records, and telemetry confirming that the native SparseDriveV2 argmax is preserved before intervention. Valid driving failures, including blockage and route deviation, remain in the evaluation. Infrastructure-interrupted attempts are retained for review and are not counted as completed evaluations.

F.5 Recovery Metrics and Statistical Aggregation

To assess recovery behavior separately from overall route success, we define recovery success at the level of a temporally grouped recovery event. An event begins when a recovery action is executed, and consecutive retries remain part of the same event until nominal driving remains healthy for five consecutive simulation seconds. Under the verifier-admitted definition, success requires an executed action with a verifier score of at least 0.43 to complete, followed by this uninterrupted interval of nominal, verifier-approved driving without renewed recovery. Routes requiring no recovery do not contribute to the denominator, and events without a verifier-admitted action are excluded from this definition. Outcomes with insufficient evidence or termination before confirmation are reported as unknown. Recovery success rate is computed as $\text{Rec.-SR} = 100 \times N_{\text{success}}/(N_{\text{success}} + N_{\text{failure}})$, excluding unknown outcomes. This metric measures a sustained return to nominal planning; because executor completion includes expiration of an execution-time budget, it does not independently certify geometric maneuver completion or collision-free recovery.

G Targeted Perception Adaptation and Detection Evaluation

Reliable runtime verification requires a scene representation that exposes hazards in proposed trajectories. This appendix describes the adapted BridgeDrive detector used in the Fail2Drive sensor-based runtime evaluations, including the traditional-monitor comparison, and evaluates its detection performance. NavSafe-∞ uses BridgeDrive's native perception module without these additional trained modules. We first examine limitations of the frozen LEAD/BridgeDrive detector (Liu et al., 2026a), then describe the trained pedestrian and static-obstacle modules used to address these gaps. LEAD denotes the unadapted base detector in the figures and tables below; the adapted detector combines this frozen model with the additional trained modules.

G.1 Perception Limitations of BridgeDrive

Category-wise detection gaps. Table 12 reveals a pronounced category imbalance. On eval20 (20 episodes, 3,587 frames), the base detector achieves 92.80% vehicle recall, but only 0.00% pedestrian and 11.06% static-obstacle recall at BEV IoU ≥ 0.5 and score threshold 0.35. All $1{,}053$ human-pedestrian occurrences and $5{,}321$ of $5{,}983$ static-obstacle occurrences are unmatched. Strong vehicle performance therefore does not imply adequate coverage of other driving hazards. These findings concern the evaluated checkpoint and configuration on our recorded scenes, rather than BridgeDrive's capabilities in every setting.

Static-obstacle precision is also only 23.59%, with $2{,}144$ false positives. The scene representation thus contains both omissions and spurious objects. Simply producing more boxes would not address both: a runtime needs detections that cover relevant hazards without excluding free space through false alarms.

Failure patterns and examples. Figure 10 shows that the deficit is already present nearby. Within 10 m, base pedestrian and static-obstacle recall are 0.0% and 28.0%; at 10–20 m, they are 0.0% and 5.0%. Together, these bins contain 864 pedestrian and $4{,}832$ static-obstacle occurrences. The shortfall therefore extends to the nearby space that candidate maneuvers may traverse.

Figure 11 shows nearby pedestrians omitted despite LiDAR returns at their annotated locations, and a partially recovered static-obstacle layout. Some available sensor evidence thus fails to become

Table 12: Detection on eval20 at BEV IoU ≥ 0.5. P, R, and F1 are percentages; $s_{\min}$ is the confidence threshold. The pedestrian reference row changes only the final threshold.

Category	System	$s_{\min}$	TP	FP	P	R	F1
Vehicle	LEAD	0.35	17,046	1,195	93.45	92.80	93.12
	+ specialists	0.35	17,046	1,195	93.45	92.80	93.12
Pedestrian	LEAD	0.35	0	15	0.00	0.00	0.00
	+ specialists	0.10	637	298	68.13	60.49	64.08
	+ specialists	0.35	573	88	86.69	54.42	66.86
Static	LEAD	0.35	662	2,144	23.59	11.06	15.06
	+ specialists	0.35	3,177	2,816	53.01	53.10	53.06

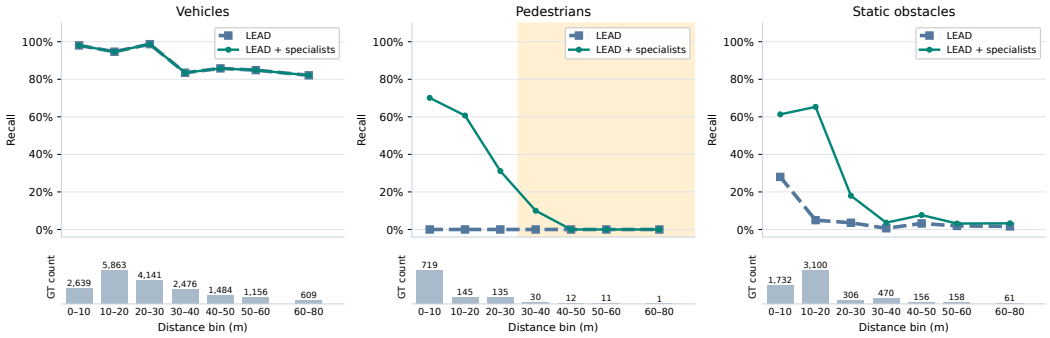


Figure 10: Recall by distance for vehicles, pedestrians, and static obstacles. All panels compare LEAD and LEAD + specialists on the same frames and use identical axes. Operating points match Table 12. Bars show GT object–frame counts; shaded intervals have fewer than 50 occurrences. Lines connect bin statistics without smoothing.

usable object boxes. These examples do not isolate classification, localization, score filtering, or training coverage as the cause; nor do distance bins establish an occlusion-related explanation. The supported diagnosis is incomplete scene evidence at the detector's output.

G.2 Specialist Architecture, Training, and Data

G.2.1 Targeted Perception Adaptation

Category-specific specialists. We augment the frozen LEAD detector with pedestrian and static-obstacle specialists trained on sensor data collected from CARLA scenarios. Both specialists take current dual-LiDAR observations as input, and the static-obstacle branch additionally uses frozen LEAD BEV features. Simulator-generated object annotations provide supervision. Only the additional specialists are trained; the original detector and driving policy remain frozen.

Integration with the frozen runtime components. Pedestrian predictions replace the corresponding outputs of the base detector, while static-obstacle proposals are merged with base detections using non-maximum suppression (NMS). Vehicle predictions remain unchanged. The combined detections preserve the object-box interface used by the verifier and recovery planner. Training and postprocessing details are provided in Section G.2.2.

G.2.2 Training, Postprocessing, and Data Splits

We collect sensor observations and corresponding object annotations through rollouts in CARLA scenarios. The pedestrian and static-obstacle specialists use 1,831 and 2,520 training frames, respectively, with 17 episodes reserved for validation. For the perception dataset, training, validation, and evaluation are partitioned by route, with variants of the same route assigned to the same split.

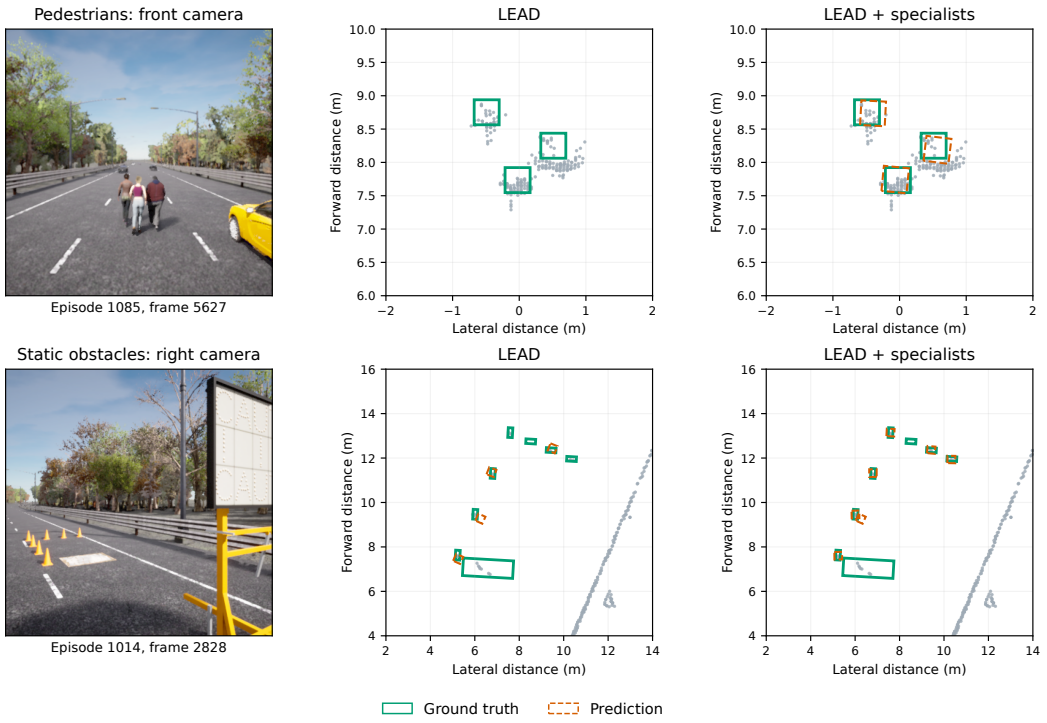


Figure 11: Recorded prediction examples. Left: camera context (front for pedestrians, right for static obstacles). Middle: LEAD. Right: LEAD + specialists. Green outlines denote recorded GT; dashed orange outlines denote predictions. Each row uses the same frame and BEV crop for both methods. Pedestrian thresholds are 0.35 / 0.10 for LEAD / specialists; both static thresholds are 0.35. The static example uses the final single-frame predictions underlying Table 12. Frame identifiers are shown for reproducibility.

Postprocessing applies local box-center refinement and pedestrian shape filtering, followed by NMS to merge static-obstacle proposals with the base detector's predictions. These operations produce the scene representation used by the runtime without updating the original detector or driving policy.

G.3 Detection Results and Remaining Limitations

Detection improvements. We compare predictions on identical frames using class-matched BEV IoU ≥ 0.5; full protocol details appear in Appendix G.2.2. Table 12 shows pedestrian recall increasing to 60.49% and static-obstacle recall to 53.10%, recovering 637 and 2,515 additional occurrences. At the same numerical threshold 0.35, pedestrian recall is 54.42% with 86.69% precision, so the improvement is not solely explained by using a lower threshold. Figure 10 shows substantial gains within 20 m, and Figure 11 illustrates recovered boxes. Vehicle predictions are preserved by construction.

Residual errors and evaluation scope. At 20–30 m, adapted recall is only 31.1% for pedestrians and 18.0% for static obstacles. Static false positives rise from 2,144 to 2,816 despite improved precision. Static gains also depend on subtype: construction cones account for 69.4% of eval20 static occurrences, eval20 influenced development choices and is a diagnostic set, not an untouched test set; counts pool correlated frames. These results quantify offline detection improvements. The adapted detector is used in the sensor-based closed-loop runtime evaluations, including the traditional-monitor comparison. However, those evaluations do not isolate the effect of detector adaptation: a controlled closed-loop comparison between the base and adapted detectors is not reported.

G.4 Implications for Verification and Recovery

Incomplete evidence for verification. An independent detector supplies a separate scene estimate, but can still omit hazards. Without an object's box, the corresponding collision constraint cannot reject an intersecting trajectory. A nominal plan may therefore pass verification despite that hazard. Conversely, a spurious box can reject a feasible trajectory and trigger unnecessary braking or recovery. Refining verification rules alone cannot repair these missing or misleading inputs.

Incomplete context for recovery. Recovery candidates are checked against the same context. Searching more bypass or reverse maneuvers does not reveal an omitted obstacle; an alternative may be admitted using the same incomplete geometry. False positives can instead exclude usable maneuvering space. Both verification and recovery therefore motivate supplementing scene evidence alongside expanding the available actions. The observed category gaps suggest training pedestrian and static-obstacle specialists while retaining vehicle outputs. The following offline study evaluates this focused adaptation; its effect on runtime decisions still requires direct evidence.

H Motivating Research Questions

This section addresses high-level questions about the positioning of AutoAgent0, the interpretation of its comparisons, and its real-time applicability.

H.1 How does an agentic runtime differ from existing driving runtimes?

Question: Augmenting an end-to-end policy with a runtime is a natural idea, and runtime supervision for autonomous driving has been studied extensively. What problem does an *agentic* runtime solve that existing driving runtimes do not?

Answer: Existing driving runtimes share a common structure: a fixed intervention criterion paired with a predefined corrective response. RSS triggers braking when safe-distance rules are violated (Shalev-Shwartz et al., 2017). Predictive safety filters and MPC-based monitors minimally modify the nominal control subject to modeled dynamics and constraints (Wabersich & Zeilinger, 2021; Sinha et al., 2023). Uncertainty-based monitors switch to a fallback controller when an estimated uncertainty exceeds a calibrated threshold (Contreras et al., 2025). These mechanisms are rule-based or optimization-based, and they are designed to *contain* an unsafe proposal rather than to *resolve* the situation that produced it.

This design leaves two gaps in long-tail closed-loop driving. First, the intervention decision is only as reliable as its scene evidence. Missed actors lead to missed interventions, while spurious detections lead to unnecessary ones. Second, the response space is fixed and typically longitudinal. Braking or slowing along the nominal path can avoid a collision, but it cannot restore progress when the nominal path itself is blocked, for example by a construction zone or a stalled vehicle, or when the expert is unwilling to change lanes. In such states, the vehicle stalls and eventually times out.

AutoAgent0 makes both components agentic. For intervention evidence, scene context refinement coordinates complementary perception skills, including cross-modal consistency checks and VLM-based semantic validation, to revise the scene used by the verifier (Section 3.2). For corrective response, recovery planning treats intervention as a search problem rather than a fixed rule. The VLM selects parameterized action primitives such as reversing, creeping, and lateral bypass, and it revises the selection using verifier feedback and a memory of prior attempts (Section 3.3). The VLM never executes actions directly: every proposal, nominal or recovery, must pass the same rule-based admission criterion before it reaches the controller. Prior work has used language models in driving either as the planner itself (Mao et al., 2024) or as an anomaly detector that selects among predefined fallback behaviors. To our knowledge, AutoAgent0 is the first closed-loop driving runtime in which agentic reasoning both refines the evidence for intervention and generates and revises the corrective maneuver, while the frozen expert remains the nominal planner.

H.2 What does the comparison with traditional runtime monitors establish, and why does the agentic runtime perform better?

Question: Section 4.4 shows that AutoAgent0 outperforms RSS, longitudinal MPC, and uncertainty-based monitors. Beyond higher scores, what claim does this comparison support, and what explains the difference?

Answer: The comparison examines whether execution-aware intervention and feedback-guided recovery improve long-tail driving beyond the evaluated conventional monitors. All compared runtimes wrap the same frozen SparseDriveV2 checkpoint, use the same adapted BridgeDrive detector, and are evaluated on the same 160 non-calibration routes. The monitor proposals are checked by the shared verifier and replace the nominal trajectory only when they pass verification and achieve a higher PDMS score (Appendix F.4). Thus, the observed differences cannot be attributed to using different detector backbones or adaptation checkpoints. However, sharing a detector does not by itself isolate the recovery mechanism: runtime scene refinement, intervention logic, and available responses must also be considered. The within-harness comparison between controlled fallback with scene refinement and full recovery holds perception fixed and examines the contribution of recovery planning more directly.

Three pieces of evidence explain why the agentic runtime performs better.

- **Containment alone does not produce task success.** In Table 3, rule-based verification with controlled fallback, which is effectively a conventional verify-and-brake runtime, reduces mean collision events per route from 0.445 to 0.095, while SR decreases from 63.4% to 60.0%. Thus, reducing collisions does not necessarily increase task completion. The MPC example in Figure 5(c) illustrates one possible limitation of braking: MPC brakes in time, then remains stuck behind the obstacle and times out.
- **Recovery restores progress rather than repeatedly containing failures.** Adding recovery planning to the scene-refined verify-and-fallback runtime raises SR from 74.0% to 83.4% and reduces recovery triggers from 54.2 to 20.3 per km, while mean collision events per route increase from 0.050 to 0.145. Fewer triggers with higher success suggest that recovery maneuvers resolve the underlying failure states, whereas fallback leaves the vehicle in states that keep triggering intervention. Part of the trigger reduction may also reflect that a committed multi-step maneuver counts as a single trigger event.
- **Feedback-driven revision improves task completion.** Table 4 shows that a discrete skill library or single-shot parameter selection yields limited gains (SR 77.0% and 77.3%), whereas verifier-guided revision yields SR 83.4%; mean collision events per route are 0.125, 0.200, and 0.145, respectively. This improvement is consistent with the benefit of reconsidering a rejected maneuver before execution, a capability absent from the evaluated fixed-response monitors.

Consistent with this mechanism, the gains are concentrated in the Fail2Drive Behavior group, where SR improves by 40.0 percentage points (Appendix A.5). The large gap between SR and RC gains indicates that the added successes mostly come from routes the expert already traverses substantially but cannot finish. This is the failure mode that a longitudinal-only response is not designed to resolve.

One open question remains: whether the VLM's reasoning is necessary, or whether exhaustive verifier-guided search over the same primitive library would suffice. Our current ablations do not include this non-agentic search baseline.

H.3 Is the relevant comparison "with vs. without a runtime" or "agentic vs. other runtimes"?

Question: Much of the experimental section compares each expert with and without AutoAgent0. If the contribution is the agentic runtime, should the central comparison not be between the agentic runtime and other runtimes?

Answer: We agree that the agentic-versus-conventional comparison is central, and our experiments address it at two levels. Across runtime designs, Section 4.4 compares AutoAgent0 with three

conventional monitors using the same frozen expert, adapted BridgeDrive detector, verifier, and evaluation route set. Within a single harness, the cumulative ablation in Table 3 provides a controlled comparison with and without recovery planning. The "+ Scene context refinement" row is a verify-and-fallback runtime without agentic recovery, and the full row adds only agentic recovery planning while keeping perception, verifier, and threshold fixed. This step raises DS from 76.0 to 80.0 and SR from 74.0% to 83.4%. The recovery mechanism ablation (Table 4) further decomposes this step into skill availability, parameter selection, and feedback-driven revision.

The with-versus-without comparison in Section 4.2 serves a different purpose. It establishes that the runtime is expert-agnostic: gains hold for three experts with different architectures (UniAD, SparseDriveV2, and BridgeDrive) and on a second benchmark (NavSafe-∞), and they are larger on generalization routes than on in-distribution routes. Together, these comparisons show that runtime augmentation is broadly useful and that its agentic components account for gains that conventional runtimes do not achieve.

H.4 Can an agentic runtime react in time to safety-critical emergencies?

Question: Safety-critical events sometimes require a response within milliseconds, while VLM inference takes seconds. Since AutoAgent0 claims to improve safety, how can an agentic runtime respond in time to such emergencies?

Answer: The agentic loop is not designed to act as a millisecond-level reflex, and AutoAgent0 does not place it on that path. The runtime separates two time scales.

Per-step safety path (no VLM). At every control step, candidate trajectories are checked by the deterministic rule-based verifier, committed recovery maneuvers are monitored by execution guards on updated observations, and conservative braking is applied whenever no trajectory is admitted (Algorithm 1; Appendix D.4). Single-candidate verification takes 12.0 ms and the tracking controller 3.3 ms at the median (Table 7). VLM semantic validation in scene refinement runs asynchronously, so the control loop continues while a review is pending (Appendix C.2). Because the VLM only proposes candidates and never executes them without verification, its outputs cannot bypass this admission-and-fallback path. The ablation in Table 3 also shows where the safety benefit comes from. Most of the collision reduction (0.445 to 0.095 per route) comes from the deterministic verifier alone. The agentic recovery loop mainly improves task success (SR 74.0% to 83.4%), at the cost of more collision events than braking alone (0.050 to 0.145).

Decision-level recovery path (VLM). Recovery temporal search is invoked approximately 0.56 times per 100 control steps in the post-warm-up profiling window. This is distinct from the 60 completed recovery VLM calls recorded over the full profiling traces, including the initial warm-up steps. Its median request latency is 4–5 s for API backends (Table 8). In the synchronous Fail2Drive evaluation, recovery requests block simulation stepping until the agent returns a control command. Their wall-clock latency therefore does not imply an equivalent interval of vehicle motion or waiting at rest.

We therefore view AutoAgent0 as complementary to, rather than a replacement for, progress on end-to-end driving policies. High-frequency trajectory planning and high-quality reactive avoidance remain the responsibility of the end-to-end expert and of low-level safety systems such as automatic emergency braking. The recovery primitives prioritize admissibility and progress over trajectory quality, which is consistent with the reduced comfort on NavSafe-∞ (Table 2). AutoAgent0 targets the complementary regime in which the expert's trajectories may be locally smooth but the vehicle cannot make safe progress.

We note two limitations of this analysis. First, our measurements are from simulation on shared GPUs and from remote VLM APIs, and they do not constitute a real-time deployment benchmark. Second, the end-to-end stack itself, including nominal planning (median 397.4 ms) and perception inference (median 303.9 ms), does not operate at a 10 ms time scale in our profiling, so sub-10 ms emergency response must ultimately be provided by dedicated vehicle-level safety systems. Reducing recovery latency through smaller on-device VLMs, asynchronous invocation, or caching of recurring recovery decisions is a direction for future work.