From Language to Driving: A Dual-Loop SLM-Enhanced Framework for Multi-Planner Scheduling via a Domain-Specific Language

Jiawei Liu[1,8] **Xun Gong**[1,8*] **Muli Yang**[4] **Xingrui Yu**[3] **Fen Fang**[4] **Xulei Yang**[4]
Ivor Tsang[3,9] **Yunfeng Hu**[6] **Hong Chen**[5] **Qing Guo**[2,7*]
[1]School of Artificial Intelligence, Jilin University [2]VCIP, CS, Nankai University
[3]CFAR and IHPC, Agency for Science, Technology and Research (A*STAR), Singapore
[4]Institute for Infocomm Research (I^{2}R), A*STAR, Singapore
[5]College of Electronics and Information Engineering, Tongji University
[6]Department of Control Science and Engineering, Jilin University
[7]NKIARI, Shenzhen Futian, China [8]Engineering Research Center of Knowledge-Driven Human-Machine Intelligence, MOE, China
[9]College of Computing and Data Science, Nanyang Technological University

Abstract

Advancing from usable to collaborative autonomy requires driving systems to execute passenger instructions safely and reliably. This work formulates instruction realization as scheduling across multiple motion planners and presents a dual-loop framework that provides a transparent decision chain from natural language to vehicle control. The outer loop uses a small language model (SLM) for high-level, low-frequency semantic reasoning and schedule generation, while the inner loop performs low-level, high-frequency schedule execution and vehicle control. To compensate for the SLM's limited capacity, the framework integrates receding-horizon scheduling to segment long-horizon instruction tasks, a domain-specific language (DSL) that restricts SLM outputs to a scheduling-oriented subspace, and reinforcement learning in high-fidelity urban traffic to refine the SLM's DSL proficiency and scheduling performance. Experiments show that the framework improves instruction-completion rates while maintaining high safety and compliance relative to multiple baselines.

1 Introduction

A key requirement for autonomous driving (AD) systems (e.g., robotaxis) to evolve from merely *usable* to truly *collaborative* is the safe realization of passenger instructions (Xing et al., 2021). As illustrated in Figure 1, such instructions are typically open-ended, colloquial, and context-dependent. Reliable instruction realization enhances passenger experience and supports human–vehicle interaction (HVI) in complex traffic.

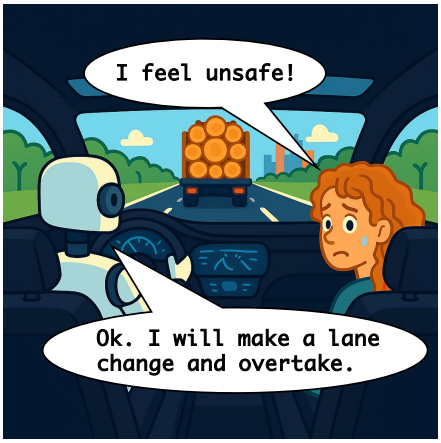

Figure 1: Passenger instruction realization. For clarification, our goal is reliable execution of *occasional* maneuver-level passenger instructions rather than *continuous* real-time language control.

Mapping language to driving is challenging. This task can be cast as a partially observable markov decision process (POMDP) over open-ended language, stochastic traffic states, and continuous vehicle control (Lauri et al., 2023). The resulting combinatorial explosion makes conventional deterministic template- or rule-based methods either brittle due to limited rule coverage, or expensive to maintain, since each added rule may conflict with many existing ones (Hu et al., 2025). Consequently, mainstream onboard HVI systems are confined to infotainment and navigation, leaving the realization of maneuver-level open-ended instructions an underexplored problem.

Recent large language model (LLM)-based AD research offers new avenues to address this challenge (Mahmud et al., 2025). Leveraging internet-scale knowledge and semantic reasoning, LLMs are being integrated into AD pipelines primarily

*Co-corresponding authors. gongxun@jlu.edu.cn, qing.guo@nankai.edu.cn

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 29957–29973

via: (i) LLM-agent methods, which emphasize explicit reasoning, external memory, and structured workflow to interpret traffic semantics, interaction intents, and risks, generating text-based control decisions. (ii) Vision–Language–Action (VLA) methods, which realize an end-to-end "see–speak–control" loop by aligning linguistic and visual representations of traffic scenes and augmenting a vision–language model (VLM) with additional modules that regress continuous vehicle control commands.

Despite early progress, most LLM-based AD methods still face the following challenges, to varying degrees, in realizing passenger instructions. **A) Passenger Instruction Understanding & Execution**. Most existing approaches rely on fixed and standardized prompts (e.g., "*predict the trajectory for the next 3 seconds*", "*turn left at the next intersection*") to condition LLM-based driving decision-making (Luo et al., 2025; Wen et al., 2024), rather than interpreting and executing open-ended language requests. *In this paradigm, language is positioned as an in-vehicle auxiliary modality for improving driving performance, rather than as a HVI interface accessible to passengers.* **B) Efficient & Traceable Language-to-Action Mapping.** VLA models typically lack explicit consistency constraints between generated text and action, leaving the decision process as an end-to-end black box (Huang et al., 2024; Wang et al., 2025b). Although LLM agents can expose their decision process (Chi et al., 2025), they often depend on complex workflows (Qian et al., 2025; Sima et al., 2024) or large, highly capable LLMs (Wen et al., 2024), creating substantial deployment burdens under onboard compute and latency constraints. **C) High-Fidelity Evaluation & Continuous Evolution.** Most LLM-based AD studies are restricted to open-loop evaluation on datasets such as nuScenes (Caesar et al., 2020) and Waymo, or validation in game-like simulators such as Highway-Env (Leurent, 2018) and CARLA (Dosovitskiy et al., 2017). It remains unclear whether LLMs can operate reliably in high-fidelity closed-loop simulation with real-world urban traffic, and whether they can further support continuous evolution in such environments.

To address these challenges, a dual-loop instruction–realization framework is proposed. The outer loop employs a language model for *high-level, low-frequency semantic reasoning*, interpreting instruction intent and generating scheduling plans for multiple motion planners. The inner loop performs *low-level, high-frequency scheduling execution and vehicle control*, establishing a safe and transparent decision-making chain from language to driving. To further enable a small language model (SLM) with limited capacity to handle scheduling, first, a receding-horizon scheduling mechanism is proposed to decompose long-horizon instruction realization into shorter segments, reducing the reasoning complexity of each SLM decision while increasing scheduling adaptability. Second, a domain-specific language (DSL), SchedulingDSL, is introduced to constrain the SLM's output to a subspace semantically aligned with the scheduling task, facilitating concise and precise expression of complex scheduling logic. Last, reinforcement learning (RL) in the nuPlan simulator (Karnchanachari et al., 2024) is used to refine the SLM, supporting continuous evolution of both its mastery of SchedulingDSL and its scheduling performance in complex urban traffic. Closed-loop evaluation in nuPlan shows that the proposed 0.5B-parameter SLM framework increases instruction realization rate by 8% over a 671B-parameter baseline while matching specialized AD comfort and safety and remaining safety-robust under inference-latency perturbations.

2 Related Work

This section reviews recent work on integrating LLMs into AD systems, organized into two categories: LLM-based methods and LLM-enhanced methods.

2.1 LLM-Based AD Methods

LLM-based AD has emerged as a prominent paradigm for integrating LLMs into AD systems (Mahmud et al., 2025). These methods place LLMs (or VLMs) directly in the vehicle control loop, jointly modeling semantic reasoning and driving decisions within a unified representation space. Broadly, they fall into two categories: LLM-agent and VLA approaches. LLM-agent methods process scene descriptions, multi-view images, or BEV representations and, using CoT reasoning, external memory, or structured workflows, generate text-based driving actions, trajectories, or control commands (e.g., GPT-Driver (Mao et al., 2023), DiLu (Wen et al., 2024), Agent-Driver (Mao et al., 2024), Poutine (Rowe et al., 2025), OmniDrive (Wang et al., 2025a), FutureSightDrive (Zeng et al., 2025b)). In contrast, VLA methods augment LLMs with action heads or expert modules that decode

multimodal features into low-level trajectories or control (e.g., LMDrive (Shao et al., 2024), DriveLM (Sima et al., 2024), Traj-LLM (Liu et al., 2025), AutoVLA (Zhou et al., 2025), Senna (Jiang et al., 2024), SimLingo (Renz et al., 2025), IRL-VLA (Jiang et al., 2025), Alpamayo-R1 (Wang et al., 2025b)) .

2.2 LLM-Enhanced AD Methods

LLM-enhanced AD methods do not treat LLMs as primary generators of low-level control. Instead, LLMs serve as auxiliary modules that semantically augment off-the-shelf AD pipelines, typically during the offline training phase or within low-frequency online decision layers. Representative directions include: using LLMs offline to synthesize interpretable rules or decision trees for scenario-specific policies (e.g., ADRD (Zeng et al., 2025a)); employing LLMs during training to provide semantically rich supervision signals and language-conditioned targets (e.g., VLM-AD (Xu et al., 2024), DiMA (Hegde et al., 2025), Words2Wheel (Han et al., 2024)); and mapping language inputs through LLMs into optimization weights, cost functions, executable scripts at run time to enable language-conditioned control (e.g., LanguageMPC (Sha et al., 2025), Diffusion-ES (Yang et al., 2024), Lampilot (Ma et al., 2024), DriveAsSay (Cui et al., 2024)).

Despite notable gains in driving performance, LLM-based and LLM-enhanced approaches, taken together, still face the aforementioned challenges for HVI scenarios, including passenger instruction realization, language–action consistency, and constrained onboard computational resources. Moreover, *most LLM-enhanced methods operate as open-loop semantic parsers and lack a principled mechanism to reconcile slow LLM inference with high-frequency vehicle control* (Liu et al., 2026).

Drawing on control design principles (e.g., hierarchical decoupling (Gong et al., 2021), timescale separation (Kokotovic et al., 1976) and event-triggered scheduling (Tabuada, 2007)), a dual-loop framework is proposed. This decoupled design preserves decision traceability while mitigating latency impact on safety. Receding-horizon scheduling, SchedulingDSL, and RL post-training further enable a compact SLM to perform scheduling.

3 Method

3.1 Problem Formalization

This study formulates instruction realization as a language-guided partially observable markov decision process (POMDP) (Lauri et al., 2023):

$$(\mathcal{S}, \mathcal{A}, \mathcal{Y}, \mathcal{T}, \mathcal{Z}, \mathcal{I}, \mathcal{R}), \tag{1}$$

where $x_t \in \mathcal{S}$, $u_t \in \mathcal{A}$, and $y_t \in \mathcal{Y}$ denote the state, action, and observation at time t. The transition and observation kernels are $\mathcal{T}(x' \mid x, u)$ and $\mathcal{Z}(y \mid x)$. A belief state $b_t(x) = \mathbb{P}(x_t = x \mid h_t) \in \mathcal{B}$ is maintained given the history $h_t = \{y_0, u_0, \ldots, u_{t-1}, y_t\}$. Given a maneuver-level instruction $\iota \in \mathcal{I}$, it can be mapped into a driving behavior sequence $\{\kappa_j\}_{j=1}^{M(\iota)}$, with $M(\iota) \in \mathbb{N}_{>0}$. Each κ_j has a completion region $\mathcal{G}_j \subseteq \mathcal{S}$ and is completed once $x_t \in \mathcal{G}_j$.

Let $n_t \in \{0, \ldots, M(\iota)\}$ be the number of completed behaviors and $\tilde{x}_t = (x_t, n_t)$ the augmented state. Progress along $\{\kappa_j\}$ is rewarded by

$$\mathcal{R}(\tilde{x}_t, u_t, \tilde{x}_{t+1}) = \begin{cases} \rho_{n_t+1}, & n_{t+1} = n_t + 1, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where $\rho_j > 0$ is the scalar reward for the completion of the behavior κ_j,

$$n_{t+1} = \begin{cases} n_t + 1, & x_{t+1} \in \mathcal{G}_{n_t+1}, \\ n_t, & \text{otherwise}, \end{cases} \tag{3}$$

with the initial condition $n_0 = 0$, and the episode terminates once $n_t = M(\iota)$.

Let Θ collect learnable parameters of an instruction-conditioned policy, it can be obtained by maximizing the expected cumulative reward:

$$\mathbb{E}_{\iota \sim \mathcal{P}(\mathcal{I})} \left[\sum_{t=0}^{N(\iota)-1} \mathbb{E}_{u_t \sim \pi_\Theta(\cdot \mid b_t, \iota)} \mathcal{R}(\tilde{x}_t, u_t, \tilde{x}_{t+1}) \right]$$
$$\text{s.t.} \quad \mathbb{P}[x_t \in \mathcal{S}_{\text{safe}},\ \forall t < N(\iota) \mid b_0] \geq 1 - \varepsilon, \tag{4}$$

where $\mathcal{P}(\mathcal{I})$ denotes the instruction distribution, $N(\iota)$ the instruction-dependent time horizon, $\mathcal{S}_{\text{safe}}$ the safe state set, and ε the tolerated risk.

Equation (4) defines an ideal objective under partial observability. In practice, the belief is approximated by a textual traffic-graph encoding of the current observation (Wen et al., 2024), which serves as the information state for outer-loop scheduling. The chance constraint is not enforced explicitly; safety is instead enforced online via constrained inner-loop MPC with a safety fallback.

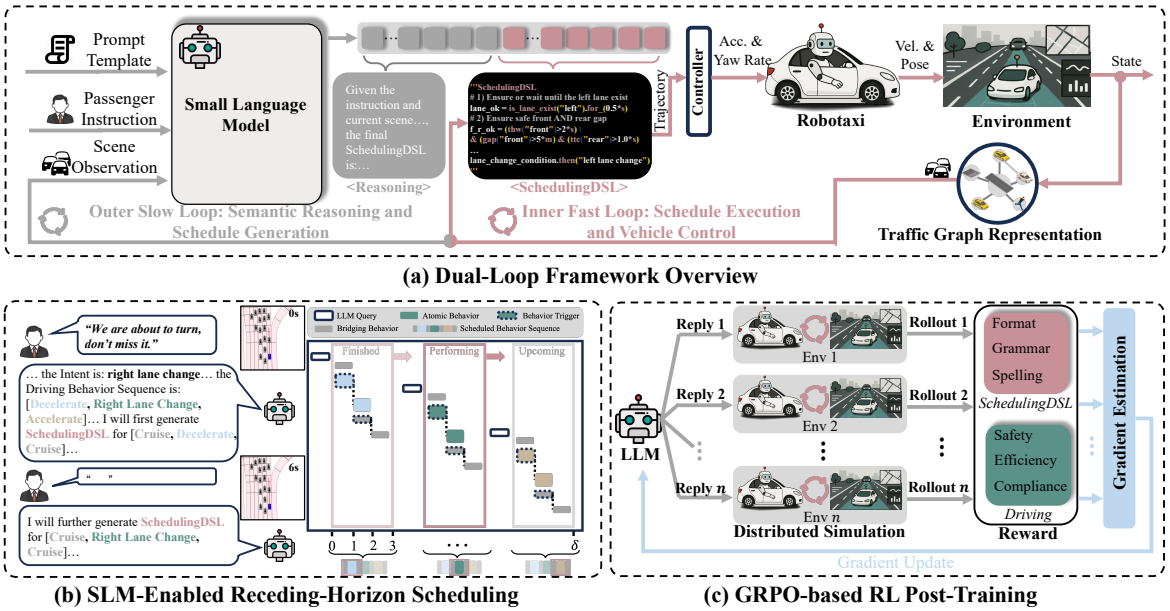

Figure 2: A dual-loop SLM-enhanced instruction-realization framework. (a) The SLM-enhanced outer loop, operating on the slow time scale τ, performs semantic reasoning and generates SchedulingDSL scripts ϕ. The inner loop, operating on the fast time scale t, executes ϕ and computes the control actions u_t. (b) Given an instruction ι (e.g., "*We are about to turn. Don't miss it.*"), the SLM g_θ first maps it to a driving behavior sequence $\{\kappa_j\}_{j=1}^{M(\iota)}$ (e.g., [`Decelerate`, `Right Lane Change`, `Accelerate`], so $M(\iota) = 3$). It is then prompted to generate, for each κ_j, a SchedulingDSL script implementing the segment [`Cruise`, κ_j, `Cruise`]. The resulting scheduling process can be viewed as condition-based switching among multiple MPC-based motion planners, driven by SchedulingDSL-defined predicates and real-time environmental feedback. (c) A GRPO-based RL post-training stage is introduced to improve the SLM's proficiency in SchedulingDSL semantics, syntax, and planner scheduling.

3.2 Framework Design

3.2.1 Dual-Loop Framework Overview

Figure 2 (a) illustrates the dual-loop framework, which approximates the solution to Equation (4) via two nested receding-horizon processes. This decoupled design follows a straightforward philosophy: *assign each subsystem the portion of the problem it is inherently suited to address.* The outer loop runs at low frequency and employs an SLM as a high-level semantic scheduler, translating passenger instructions into human-readable SchedulingDSL scripts that schedule multiple motion planners over receding horizons. The inner loop runs at high frequency and acts as a low-level execution module, invoking the selected planner and solving a receding-horizon model predictive control (MPC) problem to produce safe control commands (Darby and Nikolaou, 2012).

3.2.2 Outer Loop: SLM-Enabled Scheduling

Outer-loop decision epochs. As shown in Figure 2 (b), the outer loop is cast as a receding-horizon scheduling process. Let $\mathcal{Q} = \{1, \ldots, q\}$ denote the index set of available motion planners and $\phi \in \Phi$ the SchedulingDSL script. The realization of instruction ι can be partitioned into outer decision epochs

$$0 = \tau_0 < \tau_1 < \tau_2 < \cdots < \tau_k < \ldots \quad . \tag{5}$$

At epoch τ_k, the system has access to the current belief b_{τ_k} and the number of completed behaviors n_{τ_k}. The next behavior to execute is indexed by

$$j_k = n_{\tau_k} + 1, \tag{6}$$

where ι is considered complete once $j_k > M(\iota)$.

SLM-enhanced structured policy. Given the belief b_{τ_k} and the next behavior κ_{j_k}, the SLM g_θ generates a scheduling script

$$\phi_k = g_\theta(b_{\tau_k}, \kappa_{j_k}), \tag{7}$$

to be executed over an outer-loop horizon $N^{\text{out}} \in \mathbb{N}_{>0}$. In implementation, b_{τ_k} is approximated by a textual traffic graph from the observation y_{τ_k}. The instruction ι is omitted in Equation (7) because it is parsed only once at instruction arrival to generate the behavior sequence $\{\kappa_j\}_{j=1}^{M(\iota)}$, which already encodes the instruction intent. Formally, the semantic

interpretation of the script is given by

$$\phi_k : \{0, \dots, N^{\text{out}} - 1\} \times \mathcal{B} \to \mathcal{Q}, \tag{8}$$

which, for each local time index $t - \tau_k$ and $b_t \in \mathcal{B}$, selects a motion planner index via

$$q_t = \phi_k(t - \tau_k, b_t), \quad \text{s.t.} \quad \tau_k \le t < \tau_k + N^{\text{out}}. \tag{9}$$

Given q_t, the corresponding inner-loop planner $\pi^{(q_t)}$ produces the control action

$$u_t = \pi^{(q_t)}(b_t). \tag{10}$$

This factorization defines a *structured policy class* for Equation (4), where the policy π_Θ is realized as the composition of the SLM-based scheduler g_θ and the planner family $\{\pi^{(q)}\}_{q \in \mathcal{Q}}$.

Outer-loop scheduling objective. At each outer decision epoch τ_k, the SLM generates a script ϕ_k intended to maximize the cumulative reward over the next N^{out} steps:

$$\phi_k^\star \in \arg\max_{\phi \in \Phi} \mathbb{E}\Bigg[\sum_{t=\tau_k}^{\tau_k + N^{\text{out}} - 1} \mathcal{R}(\tilde{x}_t, u_t, \tilde{x}_{t+1}) \;\Bigg|\; b_{\tau_k}, n_{\tau_k}, \phi \Bigg]. \tag{11}$$

The SLM g_θ parameterizes a scheduling policy that approximately optimizes Equation (11).

Receding-horizon update rule. The outer loop updates the scheduling script in a *receding-horizon* manner. Let n_t denote the completed behavior number at t. The next outer decision epoch τ_{k+1} is given by

$$\tau_{k+1} = \min\left\{ t > \tau_k : \; n_t > n_{\tau_k} \text{ or } t = \tau_k + N^{\text{out}} \right\}. \tag{12}$$

Thus, a new outer-loop decision is triggered either when (i) a new behavior is completed (i.e., the ego vehicle enters the next completion region $\mathcal{G}_{n_{\tau_k}+1}$ so that $n_t > n_{\tau_k}$), or (ii) the outer horizon N^{out} is reached, whichever occurs first. At τ_{k+1}, the SLM re-evaluates the current traffic context to generate the next script ϕ_{k+1}, thereby implementing a receding-horizon scheduling scheme.

Behavior-sequence segmentation with buffers. The proposed receding-horizon scheduling embeds each driving behavior κ_j into a local segment

$$[\texttt{Cruise}, \kappa_j, \texttt{Cruise}], \tag{13}$$

where `Cruise` acts as a buffer behavior that smooths the transition into and out of the task-specific maneuver κ_j. For a generic sequence $\{\kappa_j\}_{j=1}^{M(\iota)}$, this construction yields the decomposition $\{[\texttt{Cruise}, \kappa_j, \texttt{Cruise}]\}_{j=1}^{M(\iota)}$, introducing explicit entry and exit phases around each driving behavior.

3.2.3 Inner Loop: MPC-Based Planning

The inner loop realizes the selected high-level behavior by activating one of the MPC-based planners $\{\pi^{(q)}\}_{q \in \mathcal{Q}}$ (cascaded with an LQR controller) and generating continuous control actions. Each planner $\pi^{(q)}$ encodes a distinct behavioral mode (e.g., `Cruise`, `Left/Right Lane Change`, `Acceleration`, `Deceleration`). Given the active planner index q_t and the belief b_t, the inner loop extracts a state estimate $\hat{x}_t$ from b_t and solves a receding-horizon MPC problem over an *inner-loop horizon* $N^{\text{in}} \in \mathbb{N}_{>0}$.

Let f denote the nominal vehicle dynamics, and let $\ell^{(q)}$ and $V_f^{(q)}$ be the stage and terminal cost functions. At time t, the active planner $\pi^{(q_t)}$ solves:

$$\begin{aligned} (u_t^\star, \dots, u_{t+N^{\text{in}}-1}^\star) = \arg \min_{u_{t:t+N^{\text{in}}-1}} \; & \sum_{k=0}^{N^{\text{in}}-1} \ell^{(q_t)}(x_{t+k}, u_{t+k}) + V_f^{(q_t)}(x_{t+N^{\text{in}}}) \\ \text{s.t.} \quad & x_t = \hat{x}_t, \\ & x_{t+k+1} = f(x_{t+k}, u_{t+k}), \\ & x_{t+k} \in \mathcal{S}_{\text{safe}}, \; u_{t+k} \in \mathcal{A}, \\ & k = 0, \dots, N^{\text{in}} - 1, \end{aligned} \tag{14}$$

where $\mathcal{S}_{\text{safe}}$ and $\mathcal{A}$ are the safe-state and admissible-action sets used in Equation (4). Only the first control input of the optimal sequence is applied, i.e., $u_t = u_t^\star$, and the problem (14) is re-solved at the next time step $t+1$ with updated state estimate and (possibly updated) planner index q_{t+1}, yielding a standard receding-horizon MPC loop. The inner loop also implements a safety fallback: if a risk metric (e.g., time-to-collision) falls below a threshold, a conservative safety planner (e.g., `Emergency Brake`) overrides the scheduled planner $\pi^{(q_t)}$.

3.2.4 Dual-Loop Framework Advantages

The dual-loop design provides the following advantages. i) Decomposing long-horizon behavior sequences into short local segments reduces SLM inference complexity, shortens the credit-assignment

horizon during reinforcement learning, and enables dense, segment-level rewards. ii) At each decision epoch, the SLM regenerates a scheduling plan based on the current scene to adapt to traffic changes, while `Cruise` phases buffer transitions between instruction-related maneuvers and enforce comfort constraints. iii) The hierarchical architecture exposes a traceable chain from high-level instructions to low-level actions, while the MPC loop remains responsible for enforcing safety constraints under risky instruction realizations and SLM inference latency.

3.3 SchedulingDSL

3.3.1 DSL Design Objectives

SchedulingDSL is a linear temporal logic (LTL)-inspired DSL for succinct scheduling plan specification by SLMs. It encodes traffic-safety semantics as predicates over interpretable, safety-relevant quantities and uses standard logical and temporal operators with descriptive predicate names, thereby constraining the output space and improving RL tractability for capacity-limited SLMs.

3.3.2 Syntax, Primitives, and Operators

SchedulingDSL consists of the following basic constructs:

Unit symbols. `m`, `s`, and `m_s`, which make safety thresholds explicit in the DSL and help prevent unit inconsistencies across physical quantities.

State query functions. `gap(role)`, `ttc(role)`, `thw(role)`, and `speed(role)` for accessing the current traffic state, where `role` $\in \{$`front`, `left_rear`, ...$\}$ indexes surrounding vehicles by relative position. `is_lane_exist` and `is_lane_change_completed`, which capture discrete structural information about the road topology and the ego vehicle state.

Operators. Boolean operators $\sim$ (not), & (and), | (or); comparison operators $>, \leq$; temporal operators `for`, `until`; and a scheduling operator `then`, for composing complex safety conditions and scheduling procedures.

Syntax. A SchedulingDSL formula ψ is defined *recursively* as:

$$\begin{aligned}\psi ::= p \mid \sim\psi \mid \psi_1 \& \psi_2 \mid \psi_1 | \psi_2 \mid \\ \psi_1.\texttt{until}(\psi_2) \mid \psi.\texttt{for}(\ell) \mid \psi.\texttt{then}(q).\end{aligned} \tag{15}$$

Here, p is an atomic predicate over interpretable state quantities; $\psi.\texttt{for}(\ell)$ requires that ψ hold continuously for at least duration ℓ along the recent trajectory; $\psi_1.\texttt{until}(\psi_2)$ requires that ψ_1 hold at every step until ψ_2 first becomes true; and $\psi.\texttt{then}(q)$ triggers the selection of planner $q \in \mathcal{Q}$ once ψ has held true, thereby instantiating the mapping $q_t = \phi_k(t - \tau_k, b_t)$ in Equation (9). A comparison between general-purpose language (GPL) and SchedulingDSL is provided in Appendix C.

3.4 RL-based Post Training

To enhance the SLM's proficiency in SchedulingDSL and planner scheduling, generated scripts are executed in a closed-loop high-fidelity simulator. The resulting scalar rewards are then used to update the SLM parameters via a Group Relative Policy Optimization (GRPO)-based RL objective:

$$\begin{aligned}&\mathcal{J}(\theta) = \mathbb{E}_{(\kappa_{j_k}, b_{\tau_k})\sim\mathcal{D},\ \{\phi_k^i\}_{i=1}^G \sim g_{\theta,old}(.|b_{\tau_k},\kappa_{j_k})} \\ &[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\phi_k^i|}\sum_{h=1}^{|\phi_k^i|}(\min(r_{i,h}(\theta)\hat{A}_{i,h}, \text{clip}(r_{i,h}(\theta), \\ &1-\epsilon, 1+\epsilon)\hat{A}_{i,h})) - \beta D_{KL}(g_\theta||g_{\theta,old})],\end{aligned} \tag{16}$$

where $(\kappa_{j_k}, b_{\tau_k})$ are sampled from the training data. For each $(\kappa_{j_k}, b_{\tau_k})$, the SLM samples a group of G individual SchedulingDSL scripts $\{\phi_k^i\}_{i=1}^G$. The advantage of the i-th SchedulingDSL program is calculated by normalizing the group-level reward $\{R_i\}_{i=1}^G$ via

$$\hat{A}_i = \frac{R_i - \text{mean}(\{R_i\}_{i=1}^G)}{\text{std}(\{R_i\}_{i=1}^G)}, \tag{17}$$

which is broadcast to all tokens, i.e., $\hat{A}_{i,h} := \hat{A}_i$. Token-wise likelihood ratios are computed as

$$r_{i,h}(\theta) = \frac{g_\theta(\phi_{k,h}^i | b_{\tau_k}, \kappa_{j_k}, \phi_{k,<h}^i)}{g_{\theta,old}(\phi_{k,h}^i | b_{\tau_k}, \kappa_{j_k}, \phi_{k,<h}^i)}. \tag{18}$$

To ease RL exploration, a two-level reward $R \in [0, 1]$ is used: $R = 0.5$ if the generated script is syntactically valid, and $R = 1.0$ if the valid script further completes the task without safety violations.

4 Experiment

4.1 Experimental Setting

Dataset. Experiments use a dataset derived from the nuPlan simulator (Karnchanachari et al., 2024), which reconstructs approximately 1,300 hours of

Method	LM Backbone	Realization ↑	Collision-Free ↑	TTC ↑	Drivable ↑	Speed ↑	Direction ↑	Comfort ↑
Specialized AD Method								
LogReplay	N.A.	N.A.	0.86	0.84	**1.00**	0.98	**1.00**	**0.98**
IDM	N.A.	N.A.	0.87	0.76	0.90	**1.00**	0.98	0.93
PDMClosed	N.A.	N.A.	0.97	0.86	0.98	**1.00**	**1.00**	0.98
DiLu+	DeepSeekV3-671B	N.A.	**1.00**	0.84	0.96	0.99	**1.00**	0.34
Instruction-Realization Methods								
Diffusion-ES	DeepSeekV3-671B	0.28	0.82	0.80	0.80	0.99	**1.00**	0.95
DiLu++	DeepSeekV3-671B	0.51	0.92	0.73	0.96	0.97	**1.00**	0.34
OSGS	DeepSeekV3-671B	0.84	0.99	0.88	0.97	0.97	**1.00**	0.70
Ours	Qwen2.5-0.5B	**0.91**	0.99	**0.91**	0.97	**0.97**	**1.00**	0.95

Table 1: Quantitative comparison of instruction realization performance. Each metric is in the [0, 1] range, with average metrics computed across multiple random seeds.

real-world driving. The dataset comprises 1,050 instruction-scenario pairs, where each scenario is specified by simulation initialization data such as road geometry and surrounding traffic.

Metric. Evaluation metrics include: 1) instruction realization—the proportion of instructions successfully completed; 2) collision avoidance—the fraction of scenarios completed without ego-fault collisions; 3) TTC—the minimum time-to-collision margin; 4) drivable area—the fraction of time the ego vehicle remains within drivable regions; 5) speed limit—the fraction of time the vehicle obeys posted limits; 6) direction compliance—the fraction of time the vehicle travels in the correct lane direction; and 7) comfort—the fraction of time vehicle dynamics stay within predefined comfort bounds. Metrics are normalized to $[0, 1]$ to summarize how effectively open-ended instructions are translated into safe, rule-compliant driving.

Baselines. Baselines are grouped into specialized AD methods that track a global route from expert demonstrations, and instruction-realization methods that prioritize passenger instructions (potentially deviating from the route). Specialized AD baselines include: 1) LogReplay (Karnchanachari et al., 2024) (expert-trajectory replay); 2) IDM (Treiber et al., 2000) (longitudinal single-lane car-following); 3) DiLu+, adapted from DiLu (Wen et al., 2024) by replacing discrete actions with 1Hz motion-planner selection for continuous control; and 4) PDM (Dauner et al., 2023) (nuPlan state-of-the-art simulation baseline). Instruction-realization baselines include: 5) Diffusion-ES (Yang et al., 2024), which uses an LLM to modulate planning objective for diffusion-based trajectory generation; 6) DiLu++, which extends DiLu+ with historical actions and environment context; and 7) OSGS (Liu et al., 2026) (One-Shot GPL Scheduling), which uses an LLM to generate one-shot Python scripts to schedule multiple motion planners.

See Appendix A for dataset detail, distributed training environment construction, and hyperparameter selection.

4.2 Quantitative Comparison

Table 1 compares the proposed method with baselines. For instruction realization, the proposed method with a 0.5B-parameter SLM scores 0.91. OSGS uses a 671B-parameter general-purpose LLM with GPL-based scheduling and scores 0.84. This gap arises mainly from the weak semantic alignment between GPL and scheduling (Shi et al., 2024). DiLu++ scores 0.51, reflecting low decision consistency. Diffusion-ES scores 0.28, likely because it relies on expert-demonstration priors and provides limited control over trajectory generation (see Appendix C for detailed explanation).

For safety and regulatory compliance, the proposed method matches specialized AD methods, despite operating under more risky instruction-realization conditions. This is enabled by a two-tier safety architecture: SchedulingDSL allows planner switching only when safety preconditions are met, while MPC-based motion planners use real-time environmental feedback to compute trajectories that meet safety constraints.

For ride comfort, the proposed method attains a score of 0.95. A key contributor is the treatment of `Cruise` as a buffer state in receding-horizon scheduling, which biases the vehicle toward stable, low-jerk behavior between highly dynamic maneuvers. In contrast, OSGS, which lacks this buffer, often switches directly between aggressive behaviors (e.g., acceleration immediately after lane

Method	RL Training	SchedulingDSL	Receding	Realization
Ours	×	×	×	0.00
	✓	×	×	0.00
	✓	✓	×	0.77
	✓	✓	✓	**0.91**

Table 2: Ablation study on key components including receding-horizon scheduling, SchedulingDSL, and RL-based post-training.

change), yielding a comfort score of 0.74. The decision inconsistency problem leads to the lowest comfort scores for DiLu+ and DiLu++. Diffusion-ES, while limited in instruction realization, benefits from expert-demonstration priors that yield natural trajectories and considerable comfort performance.

4.3 Training Dynamics

Figure 3 traces the evolution of reward, response length, and realization score during training. The steadily increasing reward indicates progressive acquisition of SchedulingDSL syntax and semantics by the SLM. From early to mid training, the growth in response length suggests that the SLM begins to include more details and intermediate reasoning steps in its scheduling plans (Guo et al., 2025).

In later training, however, both response length and testing realization score drop sharply even as the training reward continues to rise. Case studies attribute this pattern to overfitting: the SLM collapses to nearly fixed scheduling plans across training scenarios and suppresses intermediate reasoning, memorizing specific training instances instead of learning a generalizable scheduling procedure (Kirk et al., 2024; Fan et al., 2025). As a result, responses become shorter and generalization to unseen cases deteriorates.

4.4 Ablation Study

Table 2 reports an ablation study of the three components: receding-horizon scheduling, SchedulingDSL, and RL-based post-training. Regardless of the post-training, the SLM attains a realization score of 0 when tasked with generating an entire long-horizon scheduling plan in a single shot using GPL (i.e., Python). This failure reflects both the limited reasoning capability of the SLM and the difficulty of searching a vast, weakly constrained output space that lacks the domain-specific structure needed to reliably guide scheduling decisions (Geng et al., 2023; Shi et al., 2024).

Even with DSL and RL, requiring the SLM to produce a long-horizon schedule in one shot remains challenging: it increases task complexity, reduces adaptability, and yields sparse RL rewards. The receding-horizon scheduling mitigates these issues by decomposing long-horizon tasks into short-horizon subtasks (Mettler and Toupet, 2005; Mattingley et al., 2011). This decomposition lowers the reasoning burden, enables timely schedule revision, and provides denser reward signals, thus resulting in higher realization scores.

Metric	Latency		
	1s	2s	3s
Realization	0.68	0.43	0.37
Collision	0.98	0.98	0.98
TTC	0.92	0.88	0.90

Table 3: Latency sensitivity analysis.

Table 3 shows that the proposed dual-loop framework maintains robust safety performance under language-model inference latency (a detailed analysis is provided in Appendix B). Appendix C explains why SchedulingDSL is better suited to planner scheduling than GPL, and provides qualitative analysis of the instruction-realization methods. Animatied results are provided in the supplementary material.

5 Conclusion

This study formulates passenger maneuver-level instructions as a motion-planning scheduling problem and proposes a dual-loop, SLM-enhanced framework that decouples semantic reasoning from vehicle control across timescales while preserving decision transparency. To compensate for the limited capacity of SLMs, the framework combines receding-horizon scheduling, a SchedulingDSL, and RL post-training to decompose long-horizon instruction tasks, constrain SLM outputs, and improve DSL proficiency and scheduling performance. Experiments on the high-fidelity nuPlan simulator show that, with a 0.5B-parameter SLM, the framework outperforms the 671B-parameter LLM-based baseline OSGS by 8%, achieving an instruction-realization score of 0.91. It also matches the safety, rule-compliance, and comfort of specialized AD methods, and remains safety-robust to language-model inference delays.

6 Limitations

Further work is required before real-vehicle deployment: i) *Visual Integration*: Since visual in-

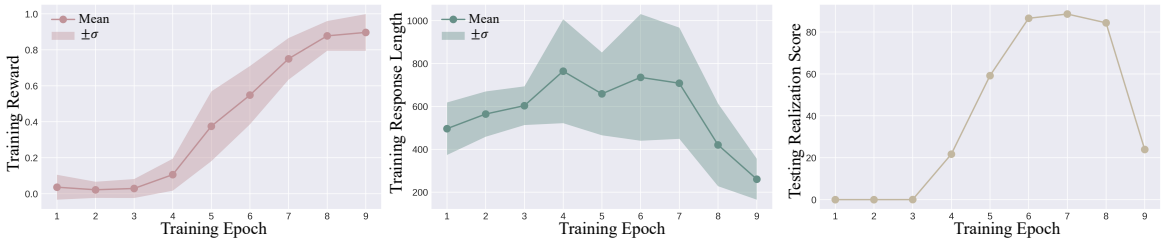

Figure 3: Training dynamics on training and testing data.

puts convey richer semantics than text alone, integrating VLM for instruction understanding remains essential. ii) *Simulation Augmentation*: nuPlan lacks ego-view image rendering, constraining closed-loop evaluation of prevailing VLA frameworks. Integrating 3D reconstruction methods (e.g. 3D Gaussian Splatting) for nuPlan would enable more comprehensive assessment.

7 Ethical considerations

This work explores a preliminary approach to realizing passenger instruction for autonomous driving. Experiments are conducted solely in simulation and are not directly deployable on real vehicles. Misinterpretation or unsafe plan generation (e.g., under adversarial instructions) could lead to hazardous behavior in the real world. Any deployment would require independent safety constraints and rigorous validation/testing. In addition, AI assistant was used for language editing of this paper; no personal or restricted data was shared; the authors are responsible for the paper.

8 Acknowledgments

This work was supported by National Natural Science Foundation of China under Grant (62573209), Development and Reform Commission Foundation of Jilin Province under Grant (2024C003), Doctoral Student Research Innovation Capacity Enhancement Program of the Education Department of Jilin Province under Grant (JJKH20250236BS), the Fundamental Research Funds for the Central Universities (No. XXX-63263254), the Agency for Science, Technology and Research (A*STAR) under its MTC Programmatic Funds (Grant No. M23L7b0021). This research/project was also supported by the National Research Foundation, Singapore under its National Large Language Models Funding Initiative (AISG Award No: AISG-NMLP-2024-003). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. 2020. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631.

Haotian Chi, Zeyu Feng, Yueming Lyu, Chengqi Zheng, Linbo Luo, Yew-Soon Ong, Ivor Tsang, Hechang Chen, Yi Chang, and Haiyan Yin. 2025. Instructflow: Adaptive symbolic constraint-guided code generation for long-horizon planning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, and Ziran Wang. 2024. Receive, reason, and react: Drive as you say, with large language models in autonomous vehicles. *IEEE Intelligent Transportation Systems Magazine*, 16(4):81–94.

Mark L Darby and Michael Nikolaou. 2012. Mpc: Current practice and challenges. *Control Engineering Practice*, 20(4):328–342.

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

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. Carla: An open urban driving simulator. In *Proceedings of the Conference on Robot Learning*, pages 1–16. PMLR.

Lishui Fan, Yu Zhang, Mouxiang Chen, and Zhongxin Liu. 2025. Posterior-grpo: Rewarding reasoning processes in code generation. *arXiv preprint arXiv:2508.05170*.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10932–10952.

Xun Gong, Jieyu Wang, Baolin Ma, Liang Lu, Yunfeng Hu, and Hong Chen. 2021. Real-time integrated power and thermal management of connected hevs based on hierarchical model predictive control. *IEEE/ASME Transactions on Mechatronics*, 26(3):1271–1282.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, and Xiao. Bi. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Xu Han, Xianda Chen, Zhenghan Cai, Pinlong Cai, Meixin Zhu, and Xiaowen Chu. 2024. From words to wheels: Automated style-customized policy generation for autonomous driving. *arXiv preprint arXiv:2409.11694*.

Deepti Hegde, Rajeev Yasarla, Hong Cai, Shizhong Han, Apratim Bhattacharyya, Shweta Mahajan, Litian Liu, Risheek Garrepalli, Vishal M Patel, and Fatih Porikli. 2025. Distilling multi-modal large language models for autonomous driving. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 27575–27585.

Jun Hu, Yuefeng Wang, Shuai Cheng, Jinghan Xu, Ningjia Wang, Bingjie Fu, Zuotao Ning, Jingyao Li, Hualin Chen, and Chaolu Feng. 2025. A survey of decision-making and planning methods for self-driving vehicles. *Frontiers in Neurorobotics*, 19:1451923.

Zhijian Huang, Tao Tang, Shaoxiang Chen, Sihao Lin, Zequn Jie, Lin Ma, Guangrun Wang, and Xiaodan Liang. 2024. Making large language models better planners with reasoning-decision alignment. In *European Conference on Computer Vision*, pages 73–90.

Anqing Jiang, Yu Gao, Yiru Wang, Zhigang Sun, Shuo Wang, Yuwen Heng, Hao Sun, Shichen Tang, Lijuan Zhu, and Jinhao Chai. 2025. Irl-vla: Training an vision-language-action policy via reward world model. *arXiv preprint arXiv:2508.06571*.

Bo Jiang, Shaoyu Chen, Bencheng Liao, Xingyu Zhang, Wei Yin, Qian Zhang, Chang Huang, Wenyu Liu, and Xinggang Wang. 2024. Senna: Bridging large vision-language models and end-to-end autonomous driving. *arXiv preprint arXiv:2410.22313*.

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

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. 2024. Understanding the effects of rlhf on llm generalisation and diversity. In *The Twelfth International Conference on Learning Representations*.

Petar V Kokotovic, Robert E O'Malley Jr, and Peddapullaiah Sannuti. 1976. Singular perturbations and order reduction in control theory—an overview. *Automatica*, 12(2):123–132.

Mikko Lauri, David Hsu, and Joni Pajarinen. 2023. Partially observable markov decision processes in robotics: A survey. *IEEE Transactions on Robotics*, 39(1):21–40.

Edouard Leurent. 2018. An environment for autonomous driving decision-making.

Jiawei Liu, Xun Gong, Fen Fang, Muli Yang, Bohao Qu, Yunfeng Hu, Hong Chen, Xulei Yang, and Qing Guo. 2026. Open-ended instruction realization with llm-enabled multi-planner scheduling in autonomous vehicles. *arXiv preprint arXiv:2604.08031*.

Jiawei Liu, Yanjiao Liu, Xun Gong, Tingting Wang, Hong Chen, and Yunfeng Hu. 2025. Harnessing and evaluating the intrinsic extrapolation ability of large language models for vehicle trajectory prediction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4379–4391.

Yuechen Luo, Fang Li, Shaoqing Xu, Zhiyi Lai, Lei Yang, Qimao Chen, Ziang Luo, Zixun Xie, Shengyin Jiang, and Jiaxin Liu. 2025. Adathinkdrive: Adaptive thinking via reinforcement learning for autonomous driving. *arXiv preprint arXiv:2509.13769*.

Yunsheng Ma, Can Cui, Xu Cao, Wenqian Ye, Peiran Liu, Juanwu Lu, Amr Abdelraouf, Rohit Gupta, Kyungtae Han, and Aniket. Bera. 2024. Lampilot: An open benchmark dataset for autonomous driving with language model programs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15141–15151.

Doaa Mahmud, Hadeel Hajmohamed, Shamma Almentheri, Shamma Alqaydi, Lameya Aldhaheri, Ruhul Amin Khalil, and Nasir Saeed. 2025. Integrating llms with its: Recent advances, potentials, challenges, and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 26(5):5674–5709.

Jiageng Mao, Yuxi Qian, Junjie Ye, Hang Zhao, and Yue Wang. 2023. Gpt-driver: Learning to drive with gpt. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. 2024. A language agent for autonomous driving. In *First Conference on Language Modeling*.

Jacob Mattingley, Yang Wang, and Stephen Boyd. 2011. Receding horizon control. *IEEE Control Systems Magazine*, 31(3):52–65.

Bernard Mettler and Olivier Toupet. 2005. Receding horizon trajectory planning with an environment-based cost-to-go function. In *Proceedings of the 44th*

IEEE Conference on Decision and Control, pages 4071–4076. IEEE.

Kangan Qian, Sicong Jiang, Yang Zhong, Ziang Luo, Zilin Huang, Tianze Zhu, Kun Jiang, Mengmeng Yang, Zheng Fu, and Jinyu Miao. 2025. Agentthink: A unified framework for tool-augmented chain-of-thought reasoning in vision-language models for autonomous driving. *arXiv preprint arXiv:2505.15298*.

Katrin Renz, Long Chen, Elahe Arani, and Oleg Sinavski. 2025. Simlingo: Vision-only closed-loop autonomous driving with language-action alignment. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 11993–12003.

Luke Rowe, Rodrigue de Schaetzen, Roger Girgis, Christopher Pal, and Liam Paull. 2025. Poutine: Vision-language-trajectory pre-training and reinforcement learning post-training enable robust end-to-end autonomous driving. *arXiv preprint arXiv:2506.11234*.

Hao Sha, Yao Mu, Yuxuan Jiang, Li Chen, Chenfeng Xu, Ping Luo, Shengbo Eben Li, Masayoshi Tomizuka, Wei Zhan, and Mingyu Ding. 2025. Languagempc: Large language models as decision makers for autonomous driving. *Preprint*, arXiv:2310.03026.

Hao Shao, Yuxuan Hu, Letian Wang, Guanglu Song, Steven L Waslander, Yu Liu, and Hongsheng Li. 2024. Lmdrive: Closed-loop end-to-end driving with large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15120–15130.

Yu-Zhe Shi, Haofei Hou, Zhangqian Bi, Fanxu Meng, Xiang Wei, Lecheng Ruan, and Qining Wang. 2024. AutoDSL: Automated domain-specific language design for structural representation of procedures with constraints. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12177–12214.

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. 2024. Drivelm: Driving with graph visual question answering. In *European Conference on Computer Vision*, pages 256–274.

Paulo Tabuada. 2007. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic control*, 52(9):1680–1685.

Martin Treiber, Ansgar Hennecke, and Dirk Helbing. 2000. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805.

Shihao Wang, Zhiding Yu, Xiaohui Jiang, Shiyi Lan, Min Shi, Nadine Chang, Jan Kautz, Ying Li, and Jose M Alvarez. 2025a. Omnidrive: A holistic vision-language dataset for autonomous driving with counterfactual reasoning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 22442–22452.

Yan Wang, Wenjie Luo, Junjie Bai, Yulong Cao, Tong Che, Ke Chen, Yuxiao Chen, Jenna Diamond, Yifan Ding, and Wenhao Ding. 2025b. Alpamayo-r1: Bridging reasoning and action prediction for generalizable autonomous driving in the long tail. *arXiv preprint arXiv:2511.00088*.

Licheng Wen, Daocheng Fu, Xin Li, Xinyu Cai, MA Tao, Pinlong Cai, Min Dou, Botian Shi, Liang He, and Yu Qiao. 2024. Dilu: A knowledge-driven approach to autonomous driving with large language models. In *The Twelfth International Conference on Learning Representations*.

Yang Xing, Chen Lv, Dongpu Cao, and Peng Hang. 2021. Toward human-vehicle collaboration: Review and perspectives on human-centered collaborative automated driving. *Transportation research part C: emerging technologies*, 128:103199.

Yi Xu, Yuxin Hu, Zaiwei Zhang, Gregory P Meyer, Siva Karthik Mustikovela, Siddhartha Srinivasa, Eric M Wolff, and Xin Huang. 2024. Vlm-ad: End-to-end autonomous driving through vision-language model supervision. *arXiv preprint arXiv:2412.14446*.

Brian Yang, Huangyuan Su, Nikolaos Gkanatsios, Tsung-Wei Ke, Ayush Jain, Jeff Schneider, and Katerina Fragkiadaki. 2024. Diffusion-es: Gradient-free planning with diffusion for autonomous and instruction-guided driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15342–15353.

Fanzhi Zeng, Siqi Wang, Chuzhao Zhu, and Li Li. 2025a. Adrd: Llm-driven autonomous driving based on rule-based decision systems. *arXiv preprint arXiv:2506.14299*.

Shuang Zeng, Xinyuan Chang, Mengwei Xie, Xinran Liu, Yifan Bai, Zheng Pan, Mu Xu, and Xing Wei. 2025b. Futuresightdrive: Thinking visually with spatio-temporal cot for autonomous driving. *arXiv preprint arXiv:2505.17685*.

Zewei Zhou, Tianhui Cai, Seth Z. Zhao, Yun Zhang, Zhiyu Huang, Bolei Zhou, and Jiaqi Ma. 2025. Autovla: A vision-language-action model for end-to-end autonomous driving with adaptive reasoning and reinforcement fine-tuning. *Preprint*, arXiv:2506.13757.

A Experimental Setting

Simulator. The proposed framework is evaluated on the nuPlan high-fidelity urban driving simulator (Karnchanachari et al., 2024). It exposes the ego vehicle to realistic urban traffic and supports fully continuous, low-level control: policies must generate acceleration and steering actions that are applied through an explicit vehicle dynamics model. The nuPlan environment includes pedestrians, traffic signals, and complex urban layouts, offering a

Instruction Intent	Examples	Proportion
Lane Change	"Exit is just around the corner, get ready for the escape.", "It is not safe to keep tailgating this car.", "I feels like we are kinda camping out in the fast lane.", ...	41.43%
Overtake	“We’re kind of stuck in slow-mo here — think we can ease by?”, "Look at him, he is basically crawling.", "Let us pass that slowpoke in front!", ...	11.43%
Pull Over	"Can we chill here for a bit? My butt needs a break from all this sitting.", "I could really use a quick pause, don’t you think? Right here works.’, "This feels like a nice place to hit pause for a minute, don’t you think?", ...	12.86%
Accelerate	"Pretty sure the cars behind us are plotting our assassination.", "Bro, we just got smoked by bicycles.", "I thought this was a highway, not a sightseeing parade!", ...	10.00%
Decelerate	"This isn’t a highway, it’s a neighborhood street.", "Are you in a race or something?", "Mind easing up a bit? I’m not in a hurry.", ...	12.86%
Compositional	"Could you gradually move us into the right lane after the turn?", "Could you overtake after we get around the corner?", "Hang tight for three, then slide by.", ...	8.57%
Cruise	"Is the left lane looking better to you?", "Change to right lane now!", "Is the left lane looking better to you?", ...	2.85%

Table 4: Examples and distribution of passenger instructions.

 # Your Task
You are a Robotaxi assistant. The current mode is “Cruise”. Refine and adapt the above DSL code to execute the passenger command “{*driving_behavior*}”, then return to “Cruise”.

 # Warning
- Use only the operators/variables/methods shown in the DSL Examples.
- Ground your DSL in the Current Scene Description.
- Keep the conditions minimal.

 # Current Scene Description
{scene_context}

 # DSL Examples
Here are some DSL examples for only reference.
…

 # Output Format & DSL Example
First answer the questions below, then provide ONE final DSL code block.
Q1: Which actors can affect command execution of “{*driving_behavior*}” in the current scene?
Q2: What safety checks enable swift execution of “{*driving_behavior*}”?
Q3: What factors should be excluded to keep DSL minimal?
The DSL code for execution of “{*driving_behavior*}” and then return to "Cruise" is:
…

Figure 4: Prompt template for receding-horizon scheduling. The scene context is encoded as a structured textual description, following DiLu (Wen et al., 2024), and a DSL example is given in the following content.

closer proxy to the challenges of real-world robotaxi deployment.

Data. The evaluation data comprises 1,050 instruction–scenario pairs. Each scenario is a 15 s traffic simulation sampled at 10 Hz (150 time steps). As shown in Table 4, the instructions are written in natural, conversational language with diverse syntax and vocabulary, and frequently use personal pronouns and modal verbs (e.g., “you,” “we,” “could”) to approximate human interaction. They deliberately avoid explicitly specifying the driving intent, requiring the language model to infer this implicit intent via semantic reasoning grounded in the traffic context. Approximately 70% of the instruction tasks involve high-risk maneuvers in urban environments (e.g., lane changes, overtaking), enabling targeted safety evaluation in complex urban scenarios. All 1,050 annotated instruction–scenario pairs are reserved exclusively for evaluation. Training data are generated by additional simulation rollouts of the IDM policy in the same base scenarios, using alternative initial traffic states that do not overlap with the test episodes.

Training setup. Performing GRPO-based RL fine-tuning in nuPlan is nontrivial. In some text-generation RL settings, a rollout can be scored immediately by comparing outputs with ground-truth targets or by querying a reward model. In this work, the language model outputs a scheduling plan, and reward is obtained by executing the plan in the closed-loop simulation and scoring the resulting vehicle trajectories and safety outcomes. GRPO typically relies on sampling multiple rollouts under an unchanged policy for the same scenario to compute group-relative advantages. For consistent evaluation and higher throughput, these rollouts should be executed in batches within fixed simulation configurations; per-rollout full reset and repeated map/scenario loading can substantially reduce training throughput. Towards this end, a distributed simulation service is built on *Ray*, encapsulating multiple nuPlan simulator instances as long-lived workers for parallel rollout execution. The simulator service accepts batched rollout requests containing scheduling plans and scenario

Metric	Latency		
	1s	2s	3s
Realization	0.68	0.43	0.37
Collision	0.98	0.98	0.98
TTC	0.92	0.88	0.90

Table 5: Latency sensitivity analysis.

identifiers, dispatches them to Ray workers, and returns per-rollout rewards along with trajectory- and safety-related metrics as each simulation completes.

In addition, nuPlan and the RL fine-tuning stack *Verl* require conflicting dependency versions (e.g., *PyTorch*), making it difficult to run both within a single Python runtime environment. To resolve this, the simulator service and the RL training process are separated into two *Conda* environments and connected via a lightweight *FastAPI* interface: the nuPlan side runs simulations and reports reward/metric summaries, while the RL side consumes these results to compute policy gradients. The two components communicate over HTTP, forming an end-to-end training loop without modifying either dependency stack.

Hyperparameters. Experiments are conducted on a workstation with AMD EPYC 7313 CPUs and 4 NVIDIA A6000 48G GPUs. All the baseline methods were run using the hyper-parameters and checkpoints recommended in their original papers or project repositories to ensure fairness. The proposed framework fine-tunes Qwen2.5-0.5B-Instruct on the constructed data using a GRPO-style objective (see Equation (16)) for 7 epochs with iterative PPO-style policy updates. Input prompts are truncated to 750 tokens, and 4 trajectories are sampled per prompt, each with a maximum length of 4,000 tokens, using decoding hyper-parameters temperature = 0.7, top-p = 0.8, and top-k = 20. Optimization uses AdamW optimizer with a learning rate of 1×10^{-6} and 2 PPO epochs per update. The training batch size is 4 prompts, with PPO mini batches of size 4 and a per GPU micro batch size of 1 implemented via gradient accumulation. To limit policy drift, KL regularization toward a fixed reference policy is applied with coefficient 1×10^{-3}.

B Latency Sensitivity Analysis

As shown in Table 5, controlled delays are introduced into the outer-loop language-model decision

```
def planner_scheduling(planner):
    def _nearest_ahead(vehicles):
        ahead = [v for v in vehicles if v.is_ahead_of(planner.ego_vehicle)]
        if not ahead:
            return None
        return min(ahead, key=lambda v: v.distance_to(planner.ego_vehicle))

    def _nearest_behind(vehicles):
        behind = [v for v in vehicles if v.is_behind(planner.ego_vehicle)]
        if not behind:
            return None
        return min(behind, key=lambda v: v.distance_to(planner.ego_vehicle))

    def _gap_to(v):
        return np.inf if v is None else v.distance_to(planner.ego_vehicle)

    def _thw_to(v):
        if v is None:
            return np.inf
        ego_v = max(planner.ego_vehicle.speed, 0.1)
        return _gap_to(v) / ego_v

    def _ttc_from_behind(v):
        if v is None:
            return np.inf
        d = _gap_to(v)
        closing_speed = v.speed - planner.ego_vehicle.speed
        if closing_speed <= 0.1:
            return np.inf
        return d / closing_speed
        yield from wait_until(lambda: planner.left_lane is not None)
    lf_gc_lp5_start = None
    lf_gc_ego_start = None
    cruise(planner)
```

```
def _lf_ok():
    nonlocal lf_gc_lp5_start, lf_gc_ego_start
    lf = _nearest_ahead(planner.left_lane.vehicles)
    gap_ok = (_gap_to(lf) >= 30.0)
    thw_ok = (_thw_to(lf) >= 1.2)
    if lf is not None and lf.speed >= 1.5:
        if lf_gc_lp5_start is None:
            lf_gc_lp5_start = planner._iteration
        else:
            lf_gc_lp5_start = None
        hold_1s = (lf_gc_lp5_start is not None) and ((planner._iteration - lf_gc_lp5_start) >= 10)
    if lf is not None and lf.speed >= planner.ego_vehicle.speed:
        if lf_gc_ego_start is None:
            lf_gc_ego_start = planner._iteration
        else:
            lf_gc_ego_start = None
        hold_0p5s = ((lf_gc_ego_start is not None) and ((planner._iteration - lf_gc_ego_start) >= 5)
        return hold_1s or gap_ok or thw_ok or hold_0p5s

def _all_ok():
    front = _nearest_ahead(planner.current_lane.vehicles)
    rear = _nearest_behind(planner.current_lane.vehicles)
    left_rear = _nearest_behind(planner.left_lane.vehicles) if planner.left_lane is not None else None
    f_r_ok = (_thw_to(front) > 2.0) and (_gap_to(front) > 5.0) and (_ttc_from_behind(rear) > 1.0)
    lr_ok = (_ttc_from_behind(left_rear) >= 2.0)
    lf_ok = _lf_ok()
    return f_r_ok and lr_ok and lf_ok

yield from wait_until(_all_ok)
success_flag_func = left_lane_change(planner)
yield from wait_until(lambda: success_flag_func())
cruise(planner)
```

(a) GPL-based Plan, python, 622 tokens

```
lane_ok = is_lane_exist("left")
f_r_ok = (thw("front")>2*s) & (gap("front")>5*m) & (ttc("rear")>1.0*s)
lr_ok = (ttc("left_rear")>=2*s)
lf_ok = ( (speed("left_front")>=1.5*m_s).for_(1*s)
|(gap("left_front")>=30*m)
|(thw("left_front")>=1.5*s)
|((speed("left_front")>= speed("ego")).for_(0.5*s)) )
llc_ok = lane_ok & f_r_ok & lr_ok & lf_ok
llc_ok.then("Left Lane Change")
cru_ok = is_lane_change_completed()
cru_ok.then("Cruise")
```

(b) DSL-based Plan, SchedulingDSL, 143 tokens

Figure 5: Comparison between GPL (i.e, Python) and DSL (i.e., SchedulingDSL) when presenting the same scheduling logic.

and their impact on instruction realization rate and safety metrics is measured. It can be observed that increasing the injected latency decreases the success rate of instruction tasks. This is because most evaluation instructions correspond to lane-change maneuvers that require timely lateral motion, so additional delay causes the ego vehicle to miss feasible lane-change opportunities. Besides, because the language model is invoked multiple times during execution, the effective end-to-end latency is approximately proportional to the per-call delay and the number of model invocations. Nevertheless, the decoupled dual loop architecture, in which a feedback-driven inner loop continuously operates, and enforces emergency behaviors, keeps the safety metrics almost unchanged under added inference delays. This result demonstrates the safety robustness of the proposed framework to language model inference latency.

C Visualization Results and Qualitative Analysis

Comparison between GPL and DSL. Figure 5 compares a Python implementation with a SchedulingDSL specification that encodes approximately equivalent scheduling logic for the same driving intent. SchedulingDSL encapsulates safety metrics (`gap`, `thw`, `ttc`) as first-class domain primitives and supports built-in temporal constraint operators

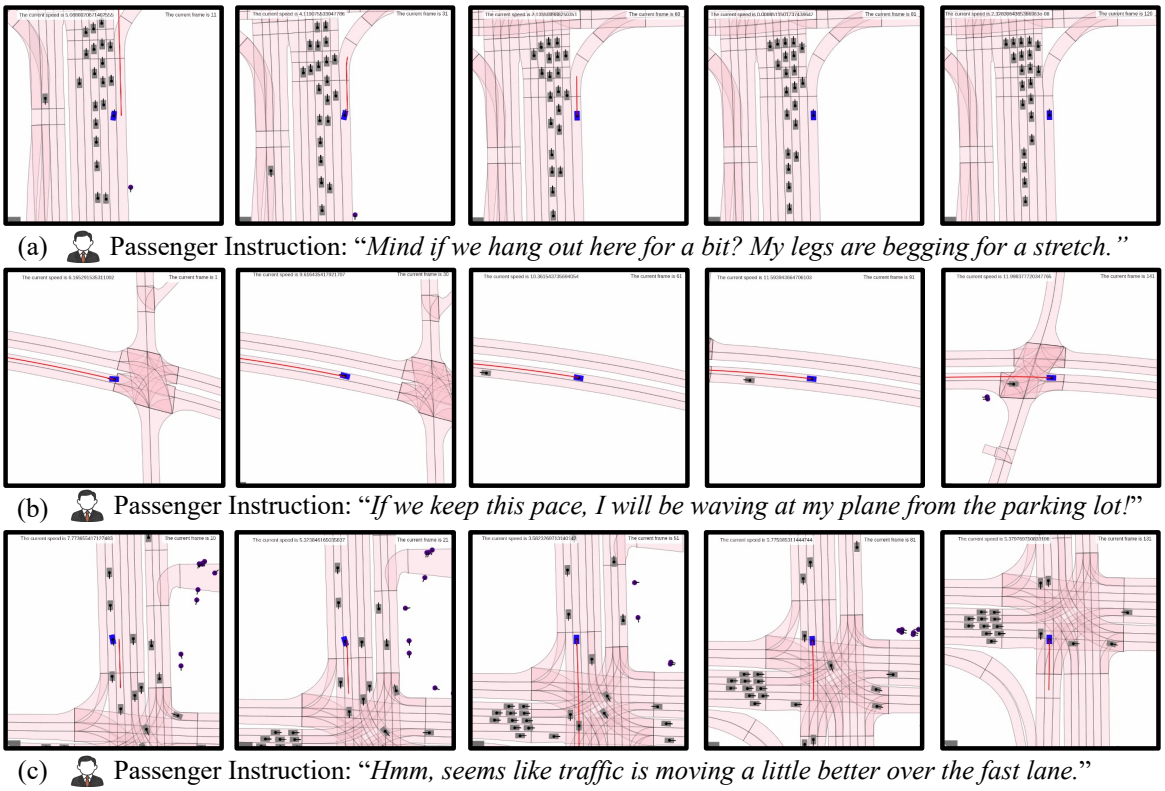

Figure 6: Open-ended passenger instruction realization. Animations are provided in the supplementary material.

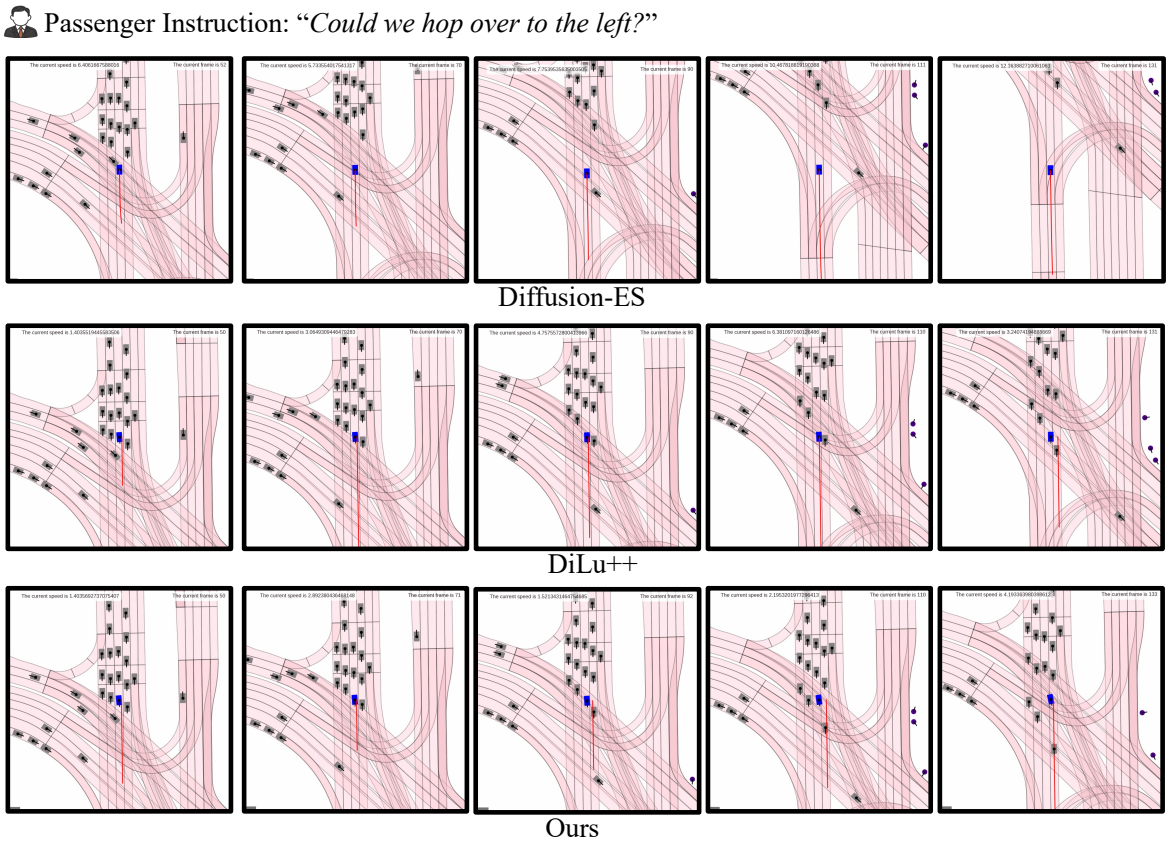

Figure 7: Comparison with Diffusion-ES and DiLu++ on the left-lane-change scenario. Animations are provided in the supplementary material.

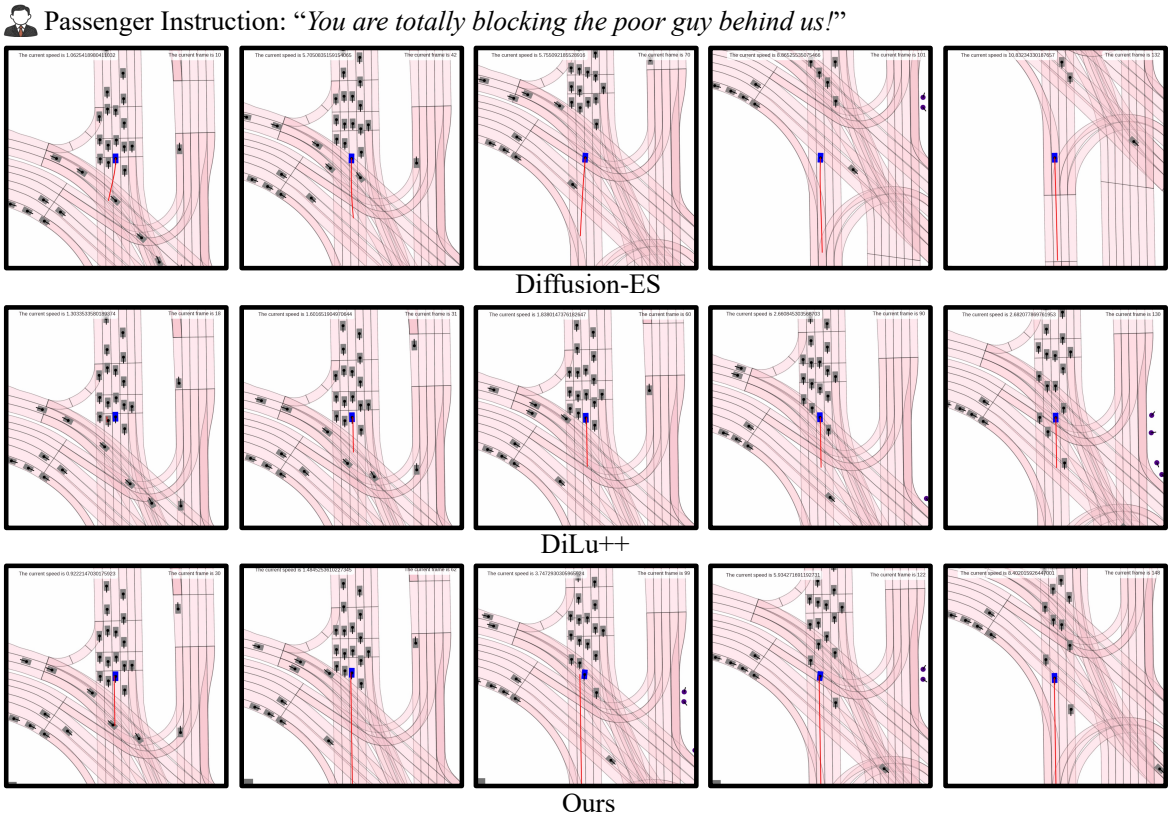

Figure 8: Comparison with Diffusion-ES and DiLu++ on the right-lane-change scenario. Animations are provided in the supplementary material.

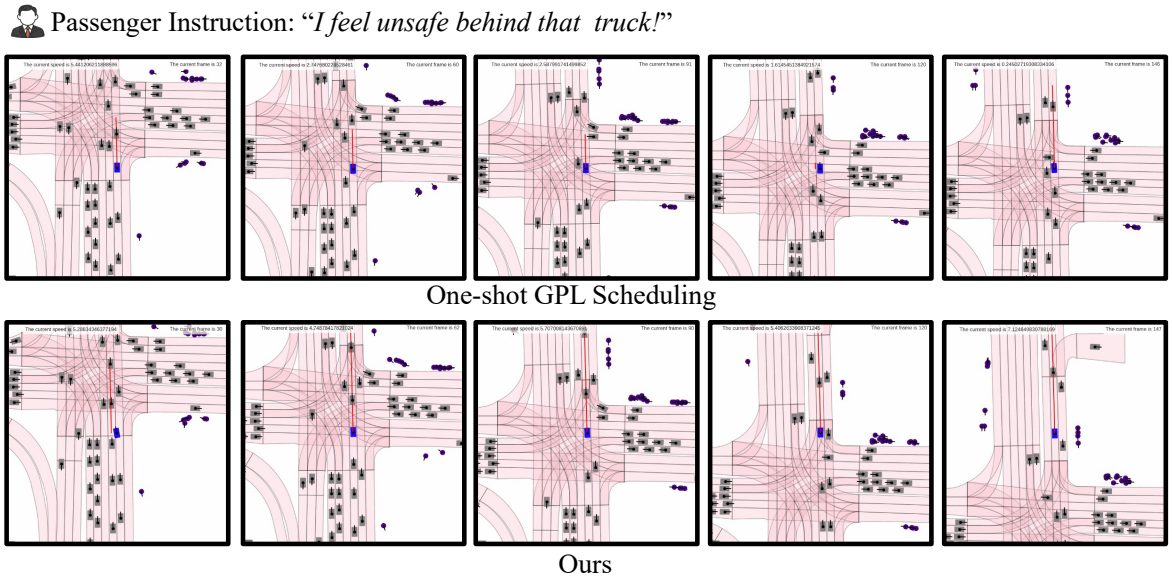

Figure 9: Comparison with One-Shot GPL Scheduling (OSGS) on the left-lane-change scenario. Animations are provided in the supplementary material.

(e.g., `for`, `until`), making rule specification closer to declarative constraint composition. In contrast, the Python script must explicitly implement metric computation, actor binding, and temporal state maintenance, expanding high-level semantics into substantially more procedural detail. Moreover, the DSL adopts a reactive semantics of "*trigger the action when the condition holds,*" whereas Python relies on `yield from` and `wait_until` to explicitly encode trigger conditions as control-flow wait

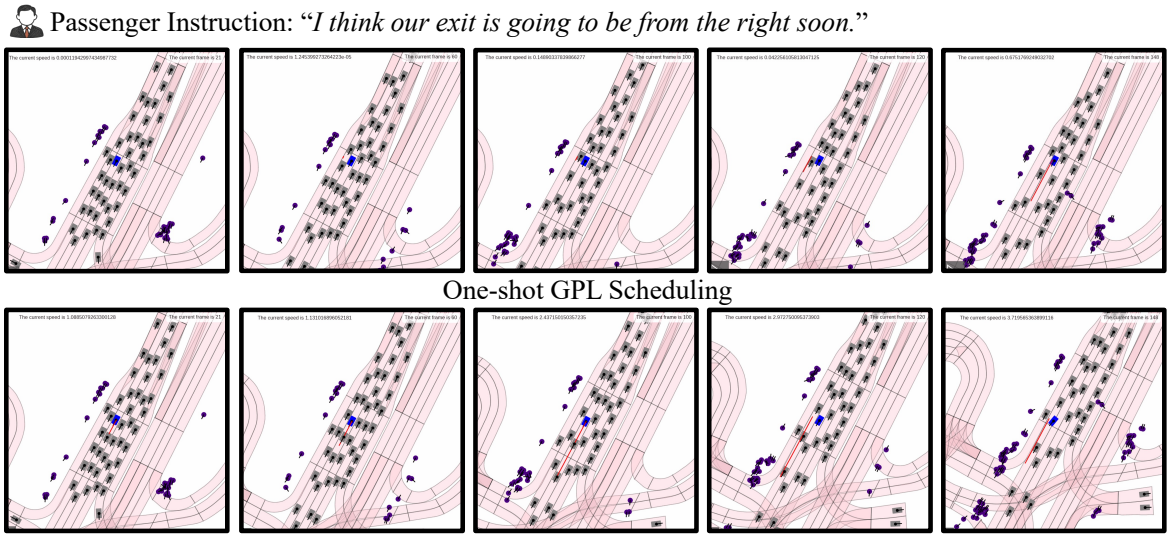

Figure 10: Comparison with One-Shot GPL Scheduling (OSGS) on the right-lane-change scenario. Animations are provided in the supplementary material.

points, which tends to introduce additional boundary handling and state management in implementation.

Although SchedulingDSL is implemented on top of Python, it is more than a thin syntactic wrapper. The SchedulingDSL introduces an explicit LTL-based domain model (defined in Section 3.3.2) for scheduling—safety metrics, actor roles, and temporal guards—as named primitives with a fixed semantics, rather than leaving these concerns to ad hoc helper functions and state variables. As a result, different driving intents can be expressed using a common vocabulary and structure, yielding specifications that are shorter, more uniform, and easier to inspect for correctness than their procedural counterparts.

Passenger instruction realization. Figure 6 illustrates how passenger instructions are realized: (a) pull over, with a right lane change and stop; (b) accelerate, with the ego vehicle speeding up; (c) left lane change, with the ego vehicle moving into the left lane.

Because longitudinal maneuver instructions are lower risk and relatively easy to execute, most of instruction realization methods can achieve considerable success rates on these tasks. The following qualitative analysis therefore focuses on lateral maneuver instructions, which are higher risk, yield lower success rates, and constitute the majority of the dataset.

Comparison with Diffusion-ES and DiLu++. Figure 7 illustrates execution of a left-lane-change command on a congested road segment. In Diffusion-ES, encoding the passenger instruction as an objective function does not adequately guide the diffusion model to generate a left-lane-change trajectory; the ego vehicle instead continues to accelerate in its original lane. DiLu++ completes the lane change but merges into the target lane with an insufficient longitudinal gap to the preceding vehicle, substantially increasing safety risk. With the SchedulingDSL mechanism, the proposed method initiates the lane change only after verifying sufficient gaps to both the leading and following vehicles in the target lane, thereby satisfying the command while reducing safety risk.

In the right-lane-change scenario in Figure 8, Diffusion-ES first accelerates, then performs a stable lane change along a smooth trajectory during the subsequent high speed phase and ultimately merges successfully. In contrast, DiLu++ makes several unsuccessful lane change attempts in the initial stage, then gradually adopts a more conservative behavior and eventually abandons the lane change. The proposed method correctly identifies the temporal gap created by the deceleration of the leading vehicle in the right lane and responds with timely acceleration, completing the lane change quickly while maintaining trajectory continuity and stability.

For Diffusion-ES, incorporating expert demonstration priors during training substantially improves performance in closed-loop autonomous driving (Yang et al., 2024). Nonetheless, this prior could also lead to ineffectiveness for instruction-

realization tasks. The main limitation arises during test-time optimization. When initial candidate trajectories are strongly shaped by the expert prior and thus far from instruction-aligned behavior, even multi-step truncated diffusion inversion with iterative refinement rarely returns them to a feasible region that satisfies the instruction. Once the initialization bias becomes too strong, inversion and correction primarily yield local adjustments around the original prior rather than the global shift required to match the intended instruction. In addition, using an objective function to guide trajectory generation during test-time optimization may introduce instability and reduce controllability. In several scenarios, the guidance signal drives the policy toward overly conservative local optima, such as "safe-but-no-progress" behavior in which the vehicle remains stationary for extended periods and fails to execute the required actions and achieve the instructed goals.

In DiLu++, the main bottleneck is context inflation from long horizon interactions. As the interaction number between the LLM and the environment grows, accumulated observations and action histories enlarge the context at each decision step and complicate reasoning and planning. In this regime, the LLM is more likely to misjudge the instruction execution state. For example, when instructed to perform a single lane change, the LLM may repeatedly trigger lane change maneuvers because of unstable state tracking. Such inconsistent decisions cause frequent switching among driving strategies and degrade instruction realization and passenger comfort. Nonetheless, in some cases, frequent interaction between the LLM and the environment can also be beneficial. DiLu+ does not use historical information and instead acts as a purely reactive agent. Its decision paradigm, driven by frequent environmental feedback at 1 Hz, improves online adaptability and collision avoidance in autonomous driving.

Comparison with OSGS. This experiment further compares the proposed method with the One-Shot GPL Scheduling (OSGS) baseline. In Figure 9, the OSGS baseline adopts a conservative lane-change plan, initiating the maneuver only when the lateral gap is sufficiently large. During the lane change, interference from a cross-traffic vehicle forces the ego vehicle to decelerate, and the maneuver cannot be completed within the finite simulation horizon. In contrast, the proposed method initiates the lane change immediately after accelerating, allowing the ego vehicle to enter the target lane earlier and satisfy the passenger instruction more promptly.

Nonetheless, a more aggressive strategy does not necessarily yield consistent benefits. As shown in Figure 10, in a congested segment the OSGS baseline changes lanes too early, limiting its ability to increase speed while merging and preventing a smooth integration into the faster traffic flow on the right. The proposed method instead maintains a cruising phase to build a speed advantage and executes the lane change only once its speed has increased and sufficient space is available in the target lane, thereby improving merge success and overall traffic efficiency.

Overall, the proposed method offers three key advantages over the OSGS baseline: i) Although the DeepSeekV3 model in OSGS has stronger general intelligence, it often violates the required response template during instruction interpretation and schedule generation. Consequently, its outputs cannot be reliably parsed by downstream modules, reducing the instruction execution success rate. The proposed framework alleviates this problem via RL–based post-training. ii) The GPL-based scheduling plans in OSGS are substantially longer and more complex than SchedulingDSL-based plans (see Figure 5). Even large models such as DeepSeekV3 struggle to produce fully correct long-horizon schedules with high probability. The proposed framework instead uses SchedulingDSL to constrain plan complexity and adopts a receding-horizon scheduling mechanism that decomposes long-horizon tasks into shorter sub-tasks, thereby improving the overall success rate. iii) In OSGS, the LLM functions as a purely open-loop semantic parser, generating a schedule only once for a given instruction. In contrast, the proposed method, through receding-horizon scheduling, allows the LLM to iteratively update the scheduling policy using environmental feedback, further increasing the completion rate of instruction-following tasks.