

# Open-Ended Instruction Realization with LLM-Enabled Multi-Planner Scheduling in Autonomous Vehicles

## Supplementary Material

### A. Related Work

In Figure 3, this work classifies LLM-in-the-loop AD systems from a scheduling perspective. To help readers track the methodological evolution in the field, this appendix instead groups prior work into three categories: (i) LLM-based AD, (ii) VLA-based AD, and (iii) LLM-enhanced AD. For brevity, we use “LLM” as an umbrella term for both LLMs and VLMs. The key distinction between category (iii) and categories (i)/(ii) is whether the LLM can be decoupled from low-level trajectory planning or vehicle control. The difference between categories (ii) and (i) is the introduction of an additional action head or expert to handle planning or control.

#### A.1. LLM-Based AD

One intuitive way to integrate an LLM into autonomous driving is to let it generate both the “reasoning” and the “decision” within the same output space. Because the LLM produces tokens autoregressively, the decision tokens are naturally conditioned on the previously generated reasoning tokens, thus establishing a chained dependency from reasoning to decision. For instance, DiLu [68] takes textual scene descriptions and past experiences from an evolving memory as LLM inputs, and outputs CoT reasoning along with high-level discrete driving actions. AlphaDrive [27] increases the reasoning token budget by post-training the VLM, aiming to elicit more sophisticated reasoning before generating high-level discrete driving actions. This technique is also known as test-time scaling.

Nonetheless, discrete driving decisions cannot be directly used for continuous control in complex real-world traffic. Consequently, this line of work is usually evaluated in simplified simulation environments such as Highway [31] or on driving QA datasets (e.g., “Q: what’s the best action to take for now? A. Acceleration; B. Deceleration;...”).

Towards closing this gap, a majority of recent studies generate planning trajectories or control signals directly in the LLM output space. GPT-Driver [39], for instance, encodes surrounding actors’ positions and the ego vehicle’s historical trajectory in text form and uses an LLM to predict the ego vehicle’s motion over the next three seconds. Agent-Driver [40], similar to DiLu [68], incorporates a cognitive memory to fuse commonsense knowledge with past driving experience and applies an explicitly defined reasoning workflow to produce planned trajectories. Poutine [48] takes multi-view images, a high-level navigation command,

and a text-based ego-trajectory as input, and uses a VLM to generate future trajectories directly in text space, where the VLM undergoes two-stage post-training: supervised fine-tuning (SFT) followed by group relative policy optimization (GRPO). To incorporate richer visual cues, OmniDrive [62], RDA-Driver [22], and OpenDriveVLA [79] extend the VLM input space with multi-view images. RDA-Driver and OpenDriveVLA explicitly encode these images as BEV features and project them into the LLM input space, whereas OmniDrive adopts a query-based 3D representation of the images and fine-tunes the VLM on 3D perception tasks. Similarly, TrajLLM [35] represents the vectorized structured scene graph using a combination of existing LLM vocabulary tokens and achieves competitive performance on trajectory prediction tasks. TOKEN [58] uses an end-to-end driving model (PARA-Drive) as a traffic-scene tokenizer to generate object-level tokens for both traffic agents and map elements, and aligns these tokens with the LLM’s text embedding space using MLP-based adapters. EMMA [23] performs end-to-end motion planning by feeding surround-view camera videos, high-level navigation commands, and text-encoded ego-state histories into a Gemini-based multi-modal LLM. By expressing all non-sensor inputs (e.g., navigation, ego status) and outputs (e.g., trajectories, 3D boxes, road graphs) as natural language, a single model can jointly handle planning and perception tasks, including 3D object detection, road graph estimation, and scene understanding. With a two-stage post-training (i.e., SFT followed by RL), AdaThinkDrive [36] adaptively determines, based on scene complexity, whether to perform reasoning before planning. FutureSightDrive [77] first expands the VLM vocabulary with MoVQGAN visual codebook, allowing the VLM to generate both images and text. At inference time, the VLM receives the current surround-view images, task description, navigation instructions, and ego-status description. It then acts as a world model, producing an imagined future frame that includes lane markings and 3D bounding boxes (denotes as spatio-temporal CoT in the FutureSightDrive). Conditioned on spatio-temporal CoT and the current observations, the VLM subsequently generates a text-based planned trajectory.

#### A.2. VLA-Based AD

Rather than directly prompting the VLM to generate a text-based trajectory, some works augment it with an additional action head/expert to regress the numerical planning trajectory/control signals. For instance, LMDrive [50] pro-

cesses camera frames and navigation commands, enabling a VLM to generate steering, throttle, and braking signals. AutoVLA [80] constructs a discrete physical action codebook from real and simulated driving log using K-disk clustering, then registers each codebook entry as a new *output* token to expand the VLM’s vocabulary. Each action token encodes a 0.5-s vehicle-motion segment. This practice enables the VLM to generate not only textual reasoning but also action token sequences that can be decoded into planning trajectories. Senna [26] is a structured planning system that couples a VLM (Senna-VLM) with an end-to-end VAD-based [25] driving model (Senna-E2E). Senna-VLM ingests multi-view surround images and navigation text to generate high-level planning decisions in natural language, which are then discretized into meta-actions specifying lateral (Left/Straight/Right) and longitudinal (Accelerate/Keep/Decelerate/Stop) intents. A meta-action encoder maps these meta-actions to embedding features, which, along with the multi-view images and navigation command, serve as inputs to Senna-E2E to generate the final planning trajectory. Building on Senna, IRL-VLA [24] first derives BEV scene features from multi-view camera images and uses a Senna-VLM-based semantic reasoning module to fuse high-level driving commands with visual observations, producing semantic-guidance features. A unified diffusion-based planner, initialized with noisy anchor trajectories and conditioned on BEV scene tokens together with the fused semantic-guidance and ego-state features, then generates fine-grained planning trajectories. SimLingo [46] takes as input a front-view camera image, a high-level language navigation instruction, and a textual ego-vehicle state. It first uses a VLM to generate a natural-language commentary that explains the vehicle’s next driving action and the rationale behind it. The system then constructs a *second input prompt* containing the image, navigation instruction, ego state, and generated commentary, and appends learnable query tokens representing the geometric path and temporally sampled (speed-related) waypoints to the end of input sequence. This full token sequence is passed through the same VLM. Finally, an MLP processes the last-layer hidden states of the two query tokens to regress the future geometric path and temporal trajectory, producing numerical planning outputs suitable for control. RecogDrive [33] uses the VLM as an encoder: it ingests the images and a textual query (e.g., “What is the next trajectory for the ego car?”) and returns the final-layer hidden states of the input tokens. The diffusion planner then conditions on these hidden states—along with historical trajectories, the ego state, and the current noisy trajectory—to produce a refined motion plan. Alpamayo-R1 [64] takes as input a multi-camera, multi-timestep image sequence, historical ego-vehicle-motion, and optional user commands and navigation text. The Cosmos-Reason VLM processes these

inputs and produces structured Chain-of-Causation (CoC) reasoning text, meta-action labels, and discrete control tokens (similar to action tokens in AutoVLA [80]). Subsequently, a flow-matching action expert, conditioned on the Cosmos-Reason VLM’s KV cache and a noisy control sequence, generates a sequence of accelerations and curvatures, which is then propagated through unicycle dynamics to produce the final planned trajectory.

As noted in the main manuscript, interpretability remains an open challenge for LLM- and VLA-based AD methods. These systems are typically implemented in an end-to-end fashion, which offers clear benefits—flexible representation learning, fewer hand-crafted modules, and strong empirical performance in complex scenarios. However, this design often lacks structured intermediate states, explicit formal constraints, and comprehensive logging, making it difficult to establish a clear, verifiable causal link between internal reasoning and the resulting planning trajectory [22, 64].

### A.3. LLM-Enhanced AD

LLM-/VLA-based AD uses large models as the central decision-making and planning module, reshaping the entire closed-loop AD system. In contrast, LLM-enhanced AD integrates large models into an existing non-LLM AD system to strengthen specific functions, while remaining largely decoupled from the original stack.

For instance, ADRD [76] uses LLMs to automatically synthesize rule-based autonomous driving policies in the form of decision trees, enabling interpretable and efficient discrete decision-making in a simplified Highway environment [31]. VLM-AD [72] employs GPT-4o as a teacher model to generate both free-form reasoning text and structured action labels. These VLM-generated annotations are converted into auxiliary text-alignment and action-classification objectives that supervise various non-LLM AD models’ training. DiMA [21] also distills knowledge from a multi-modal LLM into a vision-based end-to-end planner by adding a KL-divergence distillation loss between the penultimate-layer features of the planner and the LLM’s ego-token, and jointly training the model with surrogate tasks such as masked token reconstruction, future BEV prediction, and scene editing. LanguageMPC [49] encodes real-time traffic scenarios and traffic rules as textual inputs to an LLM. The LLM performs scene understanding and high-level decision-making and, based on its outputs, dynamically adjusts the MPC’s observation matrix, action bias, and cost-function weights (selected from a predefined weight pool). The low-level MPC then computes and applies concrete control actions within a dual-loop architecture, yielding LLM-augmented autonomous driving control.

Inspired by Code as Policy (CaP) [34] in robotics, recent work has begun to control vehicles using executable scripts generated by LLMs. Co-Pilot [61] uses an LLM

to infer human intent from standard passenger commands and, based on the road state and its memory, selects a single motion controller for each path-tracking task. Talk2Drive [9] is designed to translate passenger commands and contextual information into executable control code that adjusts the parameters of the low-level planning and control modules. Words2Wheel [19] uses LLMs, together with a Driving Style Database and a Statistical Evaluation module, to automatically translate user commands into style-specific reward functions, and then applies reinforcement learning in a car-following scenario to learn and refine the corresponding style-specific driving policies, aiming to enable automated customization and generalization of personalized autonomous driving behavior. DriveAsSay [8] can be seen as an open-loop driving method, which takes text-formatted observations from the Highway simulator [31] and uses GPT-4 to generate a script that specifies which discrete actions to take at pre-determined time points (with “if-the” logic in script). LaMPilot [37] takes as input the user’s natural-language instructions, textual scene descriptions from the game-engine-based Highway simulator [31], and API documentation. Based on these inputs, the LLM issues calls to predefined functional primitives (Ego, Perception, Navigation, Control). Diffusion-ES [74] uses an unconditional trajectory diffusion model to sample and mutate candidate trajectories, performing evolutionary search at test time to maximize arbitrary non-differentiable black-box rewards and thereby obtain high-return trajectories. For instruction-realization tasks, the authors prompt an LLM to generate Python reward-shaping programs that translate language instructions into additional reward terms, and then apply Diffusion-ES to optimize trajectories under these language-shaped rewards.

Unlike prior methods, our framework targets open-ended Robotaxi instructions and casts instruction realization as a multi-motion-planner scheduling problem. A coroutine-based scheduler with generator delegation, non-blocking conditional primitives, and asynchronous triggers enables seamless planner switching under highly concurrent AD workloads. This scheduling-centric design [51] also enforces hierarchical decoupling [18] and timescale separation [30]: the LLM is restricted to high-level, slow-loop textual reasoning, while low-level, fast-loop numerical control is delegated to verifiable MPC-based planners.

#### A.4. Language-Enhanced Benchmarks

Leveraging driving datasets for open-loop evaluation such as nuScenes [4], recent work such as NuScenesQA [44] and NuScenesSpatialQA [57] primarily focuses on object-centric perception, enabling VLMs to acquire general perception knowledge and improve object grounding in driving scenes. Furthermore, datasets such as WOMDReasoning [32] and DriveQA [66] extend vision-language annota-

tions to large-scale motion datasets, including the Waymo Open Motion Dataset and the CARLA simulator [13], with an emphasis on agent interactions, traffic regulations, and right-of-way. Nonetheless, these datasets/benchmarks provide only limited supervision for planning-oriented reasoning, a key capability for LLM-/VLA-based AD systems.

To bridge this gap, recent work has introduced language-augmented driving benchmarks for language-driven motion planning. DriveAction [20], DriveBench [70], and OmniDrive-nuScenes [62] provide comprehensive QA pairs for VLA training, covering not only planning-critical object identification but also motion prediction, traffic sign and road marking understanding, navigation following, and other related tasks. Notably, to address reasoning-action inconsistency, Alpamayo-R1 [64] constructs a CoC dataset by mining real-world driving data for critical decision moments and, for each moment, explicitly annotating the driving decision and its rationale. These annotations are paired with the corresponding historical and planned trajectories, and a hybrid labeling pipeline is used that combines human annotators with LLM-based auto-labeling.

Despite this momentum, most existing benchmarks rely on standard, well-specified language instructions rather than genuinely open-ended ones. They also heavily depend on open-loop evaluation [4, 5] or game-style simulators [13, 31], while closed-loop evaluation in high-fidelity urban traffic simulations remains comparatively rare. To address the absence of a testbed for open-ended Robotaxi passenger instructions, this work introduces the POINT benchmark. A detailed comparison between POINT and Lampilot [37], a related but simplified benchmark in the Highway environment, is provided in Appendix E.

## B. Prompt Template

Figure 8 illustrates the input templates used for open-ended instruction intent recognition and realization. To highlight the challenges of open-ended instruction understanding and to enable comparison with baseline methods that do not support open-ended instructions [68, 74], the tasks of instruction understanding and instruction execution are decoupled during evaluation.

## C. Visual Context Understanding

Figure 9 illustrates inference results from the advanced commercial vision-language model o4-mini-high [45] during instruction interpretation, using bird’s-eye-view (BEV) maps from the nuPlan [29] and Carla [13] simulations as contextual traffic cues. In the left panel, the model correctly infers a lane change intent but generates an incorrect driving behavior sequence due to hallucinated traffic elements. In the right panel, while the traffic sign is correctly detected, the model misinterprets its spatial relation-

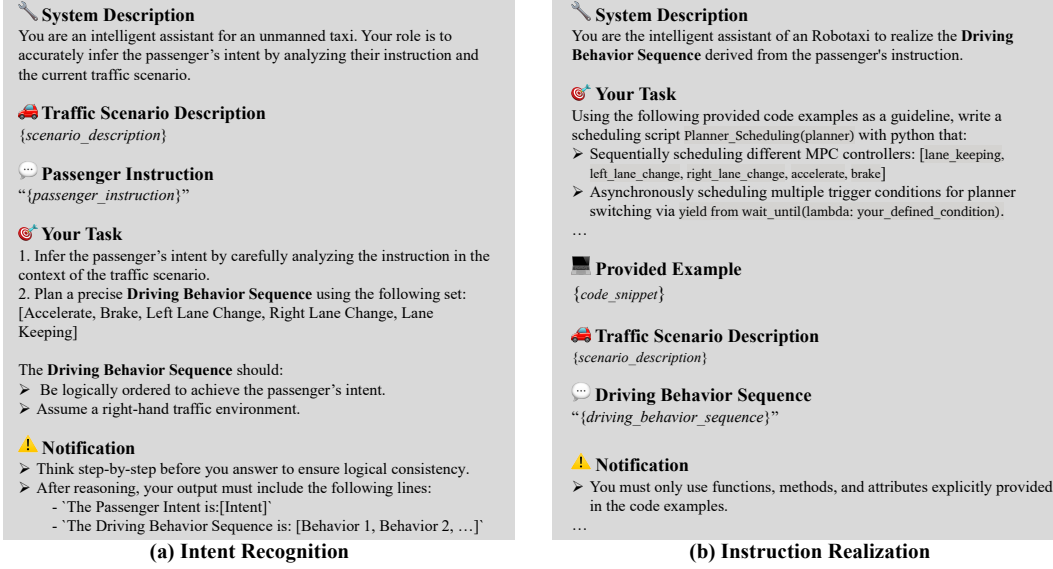


Figure 8. LLM Input Prompt Template.

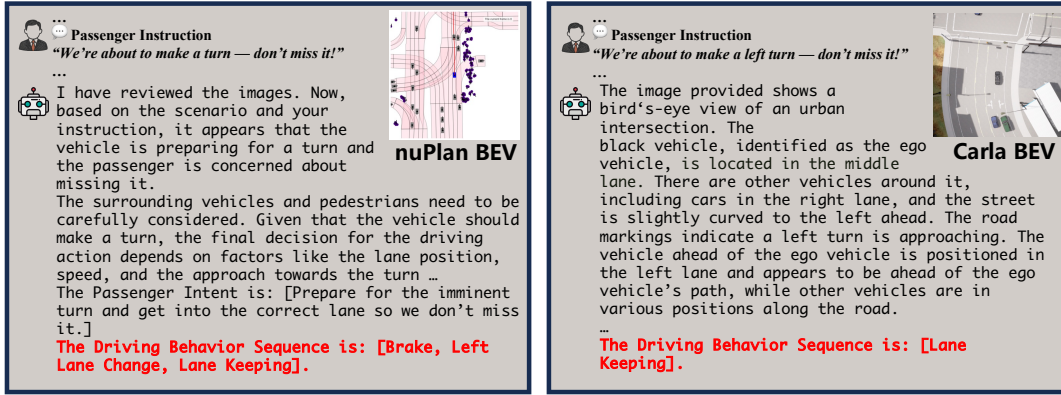


Figure 9. LLM Hallucination When Parsing Fine-Grained Visual Traffic Context.

```
... so, the scheduling script is:
'''python
def Planner_Scheduling(planner):
    # 1. start with the lane keeping.
    lane_keeping(planner)
    # 2. Wait until we are on a straight-going lane.
    yield from wait_until(lambda: lane_type_check(planner.current_lane) == 'straight-going lane')
    # 3. Accelerate to improve gap creation.
    accelerate(planner)
    speed_cap = min(1.3 * planner.ego_vehicle.speed, planner.current_lane.speed_limit)
    yield from wait_until(lambda: planner.ego_vehicle.speed > (speed_cap - 0.2))
    lane_keeping(planner)
    # 4. Wait until the target lane exists, then wait for a basic safe gap.
    yield from wait_until(lambda: planner.left_lane is not None)
    yield from wait_until(
        lambda: all(
            (v.distance_to(planner.ego_vehicle) > 10) or (not v.is_behind(planner.ego_vehicle))
            for v in planner.left_lane_vehicles)
    )
    # 5. Switch to left lane change planner.
    success_flag = left_lane_change(planner)
    # 6. Wait until the lane change completes.
    yield from wait_until(lambda: success_flag())
    # 7. Switch back to lane keeping.
    lane_keeping(planner)
...
'''
```

Figure 10. A code snippet in LLM response. It can be seen as a scene-grounded ad hoc task compiled by the LLM from an instruction. LLM reasoning text is omitted.

ship to the road and ego vehicle, resulting in a flawed decision to remain in the current lane. To mitigate such per-

ceptual hallucinations, the proposed framework adopts textual representations of traffic scenes to improve contextual grounding. Nonetheless, this does not mean the framework operates solely on text-encoded representations of traffic. Its inner high-frequency loop runs directly on raw perception outputs, such as online 3D detections and HD maps, enabling real-time, feedback-driven planner scheduling and trajectory generation.

## D. Motion Planner

Instruction realization entails managing multiple driving behaviors with distinct, sometimes conflicting, objectives—surpassing the scope of a single motion planner. To address this, five specialized planners are employed: Lane Keeping, Left Lane Change, Right Lane Change, Acceleration, and Braking. Lane Keeping relies on PDMClosed



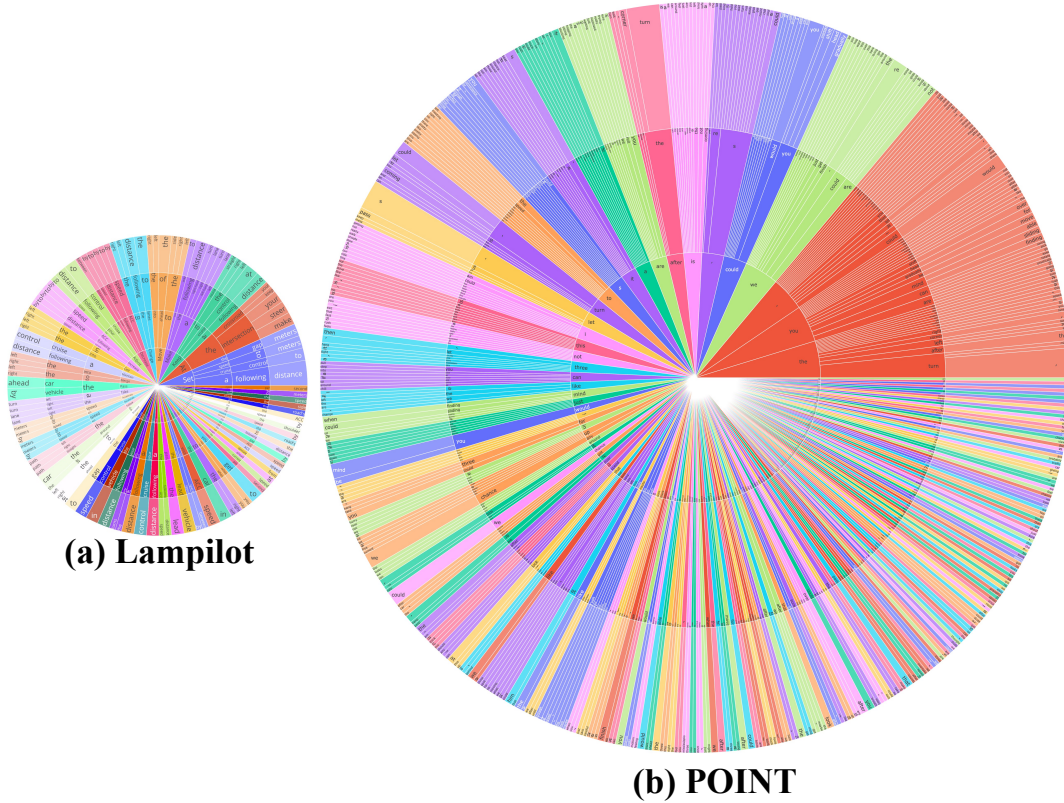


Figure 11. Word Distribution of Lampilot and POINT.

[11], while the more complex, time-critical lane changes are handled by a sampling-based MPC (SBMPC) [14, 69] that evaluates trajectory candidates and selects the top-ranked trajectory as reference. Acceleration and Braking are governed by a planner that models longitudinal motion along the lane centerline with a constant-acceleration kinematic model. Additionally, in emergency scenarios, braking is triggered when Time-To-Collision (TTC) is below a predefined threshold.

## E. POINT Benchmark

This section provides a detailed comparison with the benchmark Lampilot [37], elaborating on aspects of POINT not covered in the main manuscript.

### E.1. Simulator

The POINT benchmark is developed on the high-fidelity nuPlan simulation platform, representing a significant advancement over the earlier Lampilot benchmark based on Highway-Env [31]. Key distinctions include: (i) Highway-Env reduces complex driving behavior to discrete, instantaneous actions—e.g., lane changes are completed instantly without accounting for inertia or physical constraints—compromising realism. In contrast, nuPlan cap-

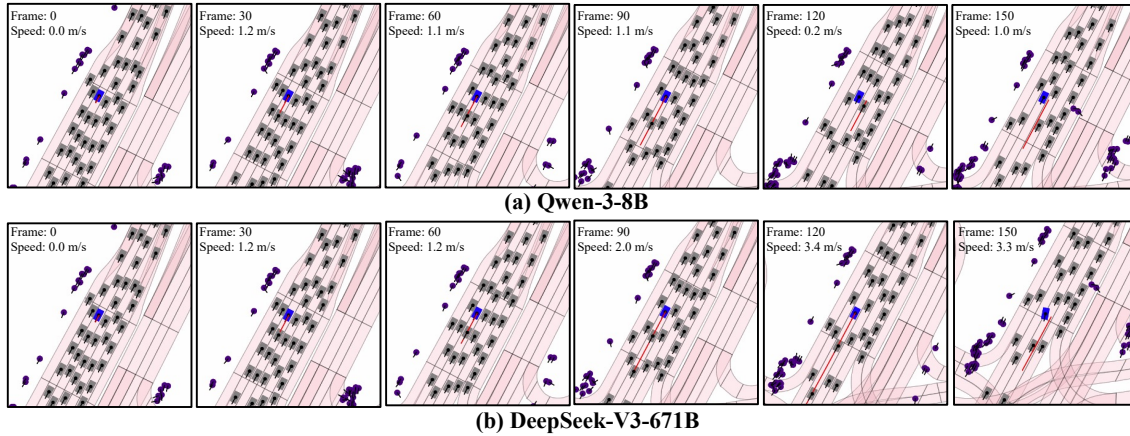
tures more nuanced behavior. (ii) nuPlan enables continuous low-level control, requiring agents to output acceleration and steering commands executed through explicit vehicle dynamics. This closely mirrors real-world control systems, unlike the simplified models in Highway-Env. (iii) While Highway-Env focuses on long-horizon planning in simplified highway settings, nuPlan features rich urban environments with pedestrians, traffic signals, and complex intersections, better reflecting the challenges of real-world Robotaxi deployment.

### E.2. Instruction

Beyond platform differences, POINT and Lampilot also differ markedly in instruction dataset size and linguistic characteristics. (i) POINT contains 1,050 test instructions—approximately twice as many as Lampilot. (ii) As shown in Figure 11, Lampilot predominantly uses templated, imperative phrasing with limited lexical variety, centered around high-frequency verbs such as “set” and “keep.” In contrast, POINT exhibits more natural, conversational language with greater structural diversity, frequently incorporating personal pronouns and modal verbs like “you,” “we,” and “could,” indicative of a more human-like interaction model. (iii) Around 70% of POINT instructions involve high-risk lateral maneuvers in urban settings (e.g.,

Table 3. Representative Instructions in POINT.

Instruction Intent	Expected Behavior	Instruction
Accelerate	Accelerate	1) "Hit the gas, Grandma!" 2) "Bro, we just got smoked by bicycles." 3) "If we keep this pace, I will be waving at my plane from the parking lot!" 4) "Pretty sure the cars behind us are plotting our assassination." ...
Decelerate	Decelerate	1) "Dude, are you trying to break the speed of sound?" 2) "This isn't a highway, it's a neighborhood street." 3) "In a hurry much? Are you late for your own wedding or something?" 4) "There's a kid in the car!" ...
Left Lane Change	Left Lane Change	1) "It is not safe to keep tailgating this car." 2) "I feel unsafe behind that truck." 3) "Looks like the left lane is moving faster." 4) "We are about to turn left, do not miss it." ...
Right Lane Change	Right Lane Change	1) "You are totally blocking the poor guy behind us!" 2) "You are practically sitting on the car behind us!" 3) "We are not supposed to be in the passing lane." 4) "Exit is just around the corner — get ready for the escape!" ...
Pull Over	Pull Over	1) "I need to get out." 2) "There's a convenience store up ahead, mind if I hop out and grab something?" 3) "Can we chill here for a bit? My butt needs a break from all this sitting." 4) "I am not feeling great, can we stop for a second?" ...
Overtake	Overtake	1) "Look at him, he is basically crawling." 2) "Can you step on it a bit? Your shoes are getting bored." 3) "Don't let him mess up our time — we've got places to be!" 4) "We're kind of stuck in slow-mo here — think we can ease by?" ...
Compositional	Compositional	1) "You can pick up the speed after the turn." 2) "Let's chill for three and then go past." 3) "Could you overtake after we get around the corner?" 4) "Could you gradually move us into the right lane after the turn?" ...
Unreasonable Lane Change	Lane Keeping	1) "Change to right lane, now." 2) "Change to left lane, now." 3) "Is the left lane looking better to you?" 4) "Could we hop over to the right?" ...

Figure 12. Key Frames in Instruction Execution Process with Various Large Language Models. The instruction intent is left lane change. Animated results are available in the [Supplementary Material](#).

lane changes, overtaking), compared to only 20% in Lampi-  
lot, enabling more targeted safety evaluation in complex ur-  
ban scenarios.

Table 3 shows representative colloquial instructions that  
lack explicit articulation of passenger intent in the POINT  
benchmark. Accurate interpretation depends on language

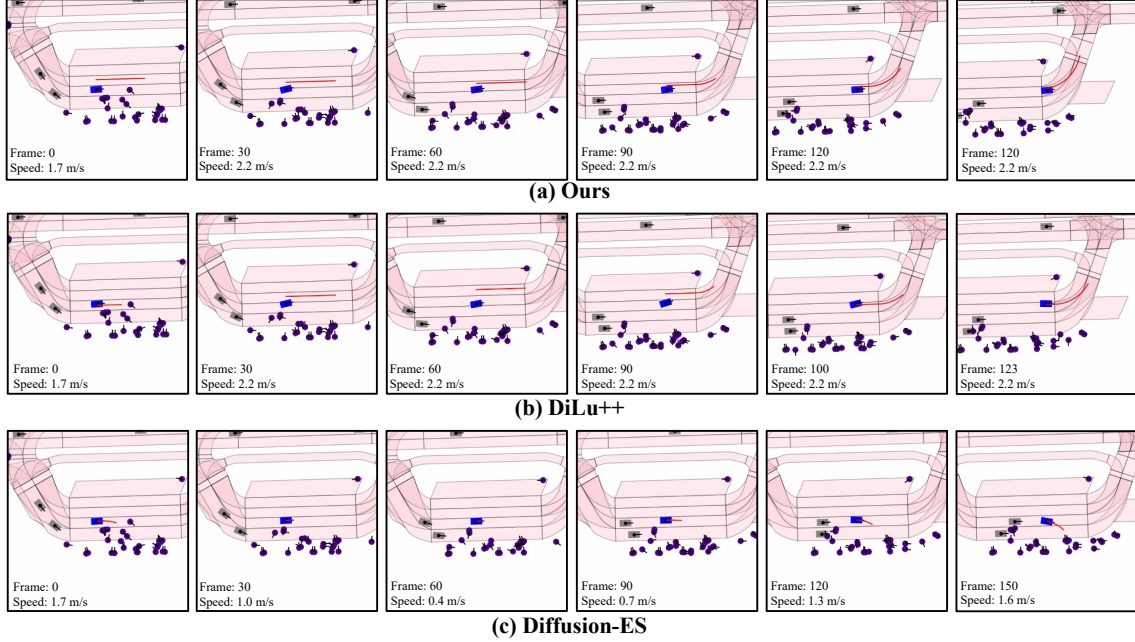


Figure 13. Key Frames in Instruction Execution Process with Various Instruction-Realization. The instruction intent is left lane change. Animated results are available in the [Supplementary Material](#).

models to infer intent from contextual traffic cues and pre-trained world knowledge. Experiment results in the main manuscript highlight the inherent difficulty of interpreting such open-ended instructions, underscoring the need for advanced, large-scale language models.

### E.3. Evaluation

Compared to Lampilot, which relies on rule-based baselines, POINT incorporates data-driven, LLM-agent, and MPC-based baselines, offering a more comprehensive evaluation framework under various metrics. The computation of safety-related and compliance-related metrics follow official nuPlan specifications [29], including: 1) Collision Avoidance, 2) TTC, 3) Drivable Area, 4) Speed Limit, 5) Direction Consistency and 6) Expert Trajectory Progress.

The 7) Instruction Recognition score quantifies the proportion of instructions whose intent is correctly identified. An instruction intent is deemed correctly recognized if the output behavior sequence contains the annotated expected behavior. For example, an instruction implying "left lane change" is satisfied by sequences such as *[accelerate, left lane change, lane keeping]*, *[left lane change]*, or *[brake, left lane change, accelerate, lane keeping]*. For an instruction implying "overtake", the output sequence must include a lane change and a subsequent acceleration. Contextual factors, such as road structure, may also be relevant. For an instruction implying "pull over", if the vehicle is not in the rightmost lane, a correct sequence must include a right lane change followed by braking. Additionally, if an in-

struction entails unsafe or structurally invalid behavior, the LLM is expected to refuse execution; such refusals are also counted as successful recognition. The 8) Instruction Realization score quantifies the proportion of instructions successfully completed under strict criteria: (i) instruction task completion within a 15-second simulation time limit; (ii) correct execution of the scheduling code without errors or exceptions; and (iii) absence of high-risk events, including collisions, red-light violations, or frequent lane changes.

### F. Case Study

Figure 12 contrasts the planner scheduling of Qwen-3-8B [73] and DeepSeek-V3 [12] under identical simulation conditions for a left lane-change command. In (a), Qwen-3-8B initiates the maneuver immediately at low speed, reducing traffic efficiency and leaving the lane change incomplete. DeepSeek-V3 delays initiation until the ego vehicle exceeds 3 m/s, enlarging the target-lane gap and shortening maneuver time, resulting in a smoother, earlier execution. This indicates that LLMs with different intelligence levels can exhibit substantial differences in behavior scheduling in dynamic traffic.

To facilitate intuitive comparison, Figure 13 illustrates how different instruction-realization methods handle the same "left lane change" instruction. In Subfigure (a), the proposed framework executes the instruction accurately. DiLu++ achieves an Instruction Realization score of 0.5, yet Subfigure (b) shows its incoherent behavior—two left

lane changes and one right lane change in a single scenario—revealing decision-making inconsistency and explaining its inferior safety performance to DiLu+. The root cause is largely that language models can occasionally overlook key information such as historical behaviors and passenger instruction in its input. This tendency becomes more pronounced in multi-turn interactions. For example, with 90% accuracy per call, the probability of completing 15 consecutive LLM calls without error drops to 20.6%, markedly raising failure risk. This observation highlights the benefit of Mode III hybrid scheduling over Mode II dynamic scheduling for instruction realization. Subfigure (c) shows that Diffusion-ES, while effective in specialized autonomous driving tasks, struggles with instruction-driven scenarios. It fails to produce instruction-aligned trajectories and can even halt the vehicle, severely degrading efficiency. A major cause is that, despite the employed test-time optimization, the diffusion process remains a black box and can yield uncontrollable plans.