Computer Modeling in Engineering & Sciences

Tech Science Press

https://doi.org/10.32604/cmes.2026.086003

ARTICLE

MALT-Drive: Moment-Aligned Language-to-Trajectory Planning for Autonomous Driving

Ziheng Lu[1], Yingfeng Cai[1,*], Wei Dong[1], Hai Wang[2,*] and Long Chen[1]

[1]Automotive Engineering Research Institute, Jiangsu University, Zhenjiang, China
[2]School of Automotive and Traffic Engineering, Jiangsu University, Zhenjiang, China

*Corresponding Authors: Yingfeng Cai. Email: 1000004192@ujs.edu.cn; Hai Wang. Email: wanghai1019@163.com

Received: 22 May 2026; Accepted: 21 July 2026; Published: 28 August 2026

ABSTRACT: Vision-language-action models have recently gained attention in autonomous driving, as language can express high-level behavioral intent beyond geometric waypoints. However, existing planners often treat language as an external command or auxiliary explanatory signal, while the hidden states of generated control instructions are rarely aligned explicitly with the latent variables that produce executable trajectories. This paper presents MALT-Drive, a moment-aligned language-to-trajectory end-to-end planning framework for autonomous driving. Given front-camera visual input, ego-state history, route priors, and a driving-oriented VQA prompt, MALT-Drive first autoregressively generates a concise control instruction from a controlled instruction space and then decodes trajectory-level actions. To make the instruction actionable, planning is decomposed into a Maneuver Branch for path geometry and a Modulation Branch for speed regulation and executability estimation. For each branch, generated control-instruction tokens and planning tokens are represented by compact moment descriptors that combine first-order pooled statistics with grouped second-order relation sketches, thereby supporting distribution-aware alignment between language semantics and latent planning states. To reduce visual imitation shortcuts, MALT-Drive further constructs counterfactual control-instruction rollouts, in which the same scene is paired with multiple executable and non-executable canonical control instructions and corresponding motion targets. Experiments on Bench2Drive and NAVSIM show that MALT-Drive improves closed-loop driving performance and provides cross-benchmark evidence of instruction-action consistency.

KEYWORDS: Autonomous driving; vision-language-action models; multimodal alignment; motion planning

1 Introduction

End-to-end autonomous driving is moving from modular pipelines toward unified multimodal systems, where perception, prediction, and planning are jointly modeled and optimized within a differentiable framework [1–4]. Recent vision-language-action (VLA) models further extend this paradigm by introducing natural language into driving policy learning. Beyond sparse route commands and geometric waypoints, language enables driving systems to express scene constraints and high-level behavioral decisions more effectively [5–8].

Compared with representative driving VLA methods, MALT-Drive focuses on the explicit coupling between generated control-instruction states and trajectory-producing planning states. Table 1 summarizes this positioning across language role, representation granularity, planning-state binding, and supervision mechanism. DriveLM organizes driving reasoning through graph-structured VQA [5], DriveMLM models decision states and explanations [9], SimLingo strengthens language-action association through alignment and Action Dreaming [10], ORION conditions trajectory generation on planning tokens [11], and AutoVLA

generates reasoning and physical action tokens autoregressively [8]. MALT-Drive uses the generated control instruction as an intermediate planning variable and aligns its control-instruction token states with branch-specific planning tokens for path geometry, speed regulation, and executability estimation.

Table 1: Methodological positioning of MALT-Drive compared with representative driving VLA methods.

Method	Input/Data	Language Role	Representation Granularity	Planning-State Binding	Supervision Mechanism
DriveLM [5]	Camera/driving annotations; DriveLM-style QA	Graph VQA reasoning	QA/graph-QA	Graph-structured reasoning for driving tasks	Driving VQA and E2E driving supervision
DriveMLM [9]	Multi-modal driving input; decision and explanation data	Decision-state explanation	Decision state/explanation	Behavioral planning-state prediction	Decision state and language explanation data
SimLingo [10]	Camera-only; Bench2Drive/CARLA	Language-action alignment	Instruction/action	Action prediction conditioned on language cues	Closed-loop driving, VQA, commentary, and Action Dreaming
ORION [11]	Multi-camera; Bench2Drive planning/VQA data	Vision-language instructed action generation	Planning-token	Generative planner conditioned on planning tokens	Unified VQA and planning optimization
AutoVLA [8]	Camera-only; driving SFT and reinforcement fine-tuning data	Autoregressive reasoning-action generation	Physical action-token	Unified reasoning and feasible action-token generation	SFT and reinforcement fine-tuning
MALT-Drive	Front camera, ego state, and route prior; Bench2Drive/NAVSIM with counterfactual VQA rollouts	Generated control instruction	Control-instruction token and planning token	Branch-wise binding to path, speed, and executability states	Same-scene counterfactual VQA-action supervision with executability labels

Throughout this paper, *control instruction* denotes the short, action-bearing textual output associated with a driving-oriented VQA prompt. The teacher-forcing target is termed the *reference control instruction*, and the autoregressive model output is termed the *generated control instruction*. A *canonical control-instruction class* is one of the 30 entries in the controlled instruction library.

The planning value of language supervision depends on whether generated control instructions can constrain downstream motion generation in a behaviorally consistent manner. Driving instructions usually affect different factors of future motion [8,10,11]. Maneuver-oriented instructions, such as lane changing and obstacle bypassing, mainly determine future path geometry. Modulation-oriented instructions, such as slowing down, stopping, and waiting, mainly affect the speed profile and executability state. Compressing these factors into a single planning representation can entangle path semantics, speed semantics, and feasibility-related semantics within the same latent space, weakening the correspondence between language states and executable actions.

Based on this observation, this paper proposes MALT-Drive, a moment-aligned VQA-supervised language-to-trajectory planning framework for end-to-end autonomous driving. Given front-view camera inputs, ego-motion history, route priors, and a driving-oriented VQA prompt, the VLA backbone first autoregressively generates a concise control instruction within a controlled instruction space. The generated control-instruction states are then reused as control-related intermediate representations for trajectory decoding. MALT-Drive further introduces a dual-branch action decoder. The Maneuver Branch predicts future path geometry, while the Modulation Branch predicts the speed profile and executability score. This design separates path-level maneuver planning from speed-executability modulation, allowing generated control instructions to constrain the corresponding planning factors through branch-specific channels.

Moment alignment is introduced to support this branch-wise coupling. Token-level alignment emphasizes local token correspondences, global embedding alignment compresses control-instruction and planning states into one vector, and standard contrastive alignment is usually applied at the sample-embedding level. In MALT-Drive, the first-order moment summarizes dominant control semantics, while the grouped second-order sketch preserves token-relation patterns within the control-instruction and planning token sets. The resulting descriptor aligns maneuver-related control-instruction states with path-generation tokens and modulation-related control-instruction states with speed/executability tokens.

At the supervision level, standard imitation learning usually provides only one expert trajectory for each visual scene, making it difficult to observe how the planner responds to language variations. MALT-Drive constructs same-scene counterfactual control-instruction rollouts. For the same scene, multiple prompt–control-instruction pairs cover executable maneuver instructions, executable modulation instructions, and non-executable instructions that should be rejected. Each reference or counterfactual control instruction is associated with corresponding path, speed, and executability targets. This supervision explicitly reveals behavior differences induced by language variations and supports the evaluation of instruction-action consistency and unsafe-instruction rejection.

The main contributions of this paper are summarized as follows:

- This paper proposes MALT-Drive, a VQA-supervised language-to-trajectory planning framework that treats generated control instructions within a controlled instruction space as control-related intermediate variables and explicitly binds their token states to branch-specific path, speed, and executability planning states.
- A dual-branch planning decoder with branch-level moment alignment is designed to separately model path-level maneuvers and speed-executability modulation, and to align branch-specific planning states with the corresponding instruction semantics through compact moment descriptors.
- Same-scene counterfactual control-instruction rollouts are introduced to provide instruction-sensitive language-action supervision and enable direct evaluation of executable action generation and unsafe-instruction rejection.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the proposed MALT-Drive framework, including dual-branch planning, moment alignment, and counterfactual control-instruction rollouts. Section 4 reports experimental results and ablation studies. Section 5 concludes the paper.

2 Related Work

2.1 End-to-End Autonomous Driving

End-to-end autonomous driving seeks to learn planning and control policies directly from sensor observations within a unified differentiable framework. Representative methods integrate perception, prediction, and planning through joint optimization, which reduces error accumulation commonly observed in modular pipelines. TransFuser fuses multi-view sensory features for navigation-oriented planning [1], while UniAD and VAD organize perception, prediction, and planning into unified task structures [2,3]. Recent generative approaches, such as GenAD, further improve behavior modeling by formulating planning as a generative prediction problem [4]. Even with this progress, most end-to-end planners are primarily supervised by expert trajectories paired with visual scenes and route priors. Such supervision is effective for imitation but provides limited evidence that a planner can respond consistently to different high-level behavioral intents under the same scene. In scenes where multiple feasible behaviors exist, a single expert trajectory may encourage visual shortcut learning rather than intent-sensitive planning. By contrast, this paper studies how generated control

instructions can be consistently associated with the latent planning states that produce path geometry, speed profiles, and executability estimates.

2.2 Language-Grounded Driving

Language has become an increasingly common component of autonomous driving to support scene understanding, explanation, instruction following, and action generation. DriveLM demonstrates the value of language supervision for driving-oriented reasoning [5]. LMDrive and EMMA condition driving policies on natural-language instructions [6,7]. Recent driving VLA models, including DriveMLM, ORION, AutoVLA, Reasoning-VLA, and OpenREAD, further explore action tokens, autoregressive generation, and unified reasoning-action modeling [8,9,11]. These studies respectively advance graph VQA, decision-state explanation, language-action learning, planning-token conditioning, and autoregressive action-token generation. MALT-Drive studies a complementary interface: branch-wise coupling between generated-control-instruction states and trajectory-producing states. Rather than introducing another global language-action objective, it uses the generated control instruction as a controlled intermediate planning variable whose control-instruction token states are aligned with path, speed, and executability planning tokens.

2.3 Instruction Consistency

Long-tail driving scenarios often call for decisions beyond ordinary trajectory imitation, such as yielding, emergency braking, obstacle bypassing, and rejecting infeasible actions. Language-action paired supervision has therefore been explored to improve behavioral consistency. SimLingo jointly optimizes closed-loop driving, vision-language understanding, and language-action alignment [10], while SteerVLA emphasizes the role of language in long-tail reasoning and high-level behavior selection [12]. Recent post-training strategies in AutoVLA and DynVLA also suggest that supervised and reinforcement fine-tuning can improve the alignment between generated language, actions, and driving rewards [8]. Still, language-action data alone does not guarantee instruction-sensitive behavior. When each scene is paired with only one instruction and one trajectory, a model may still rely on visual priors and ignore instruction changes. A more discriminative supervision signal should keep the visual context fixed while varying the instruction and the corresponding motion target. Motivated by this point, this paper introduces counterfactual control-instruction rollouts under the same scene, in which executable and non-executable instructions are paired with path, speed, and executability targets. This setting makes instruction-action consistency more identifiable and enables direct evaluation of instruction compliance and unsafe-instruction rejection.

2.4 Multimodal Alignment

Multimodal alignment has been widely studied in vision-language learning. CLIP establishes a global contrastive learning paradigm for image-text alignment [13], while EAU and ProLIP improve robustness through uncertainty-aware or probabilistic matching [14,15]. In parallel, compact high-order descriptors such as DeepBDC and moment-based representations show that second-order statistics can preserve feature interactions that are often lost by simple mean pooling [16,17]. Common alignment choices operate at different granularities: token-level alignment emphasizes local token correspondences, global embedding alignment compresses all instruction and planning states into one vector, and standard contrastive alignment usually matches sample-level embeddings. MALT-Drive targets the language-to-trajectory interface by constructing branch-level moment descriptors for control-instruction token states and planning-token states. The first-order component summarizes dominant control semantics, and the grouped second-order component preserves token-relation patterns within each branch. This provides a compact descriptor for

aligning maneuver-related language states with path-generation states and modulation-related language states with speed/executability states.

3 Method

3.1 Problem Formulation

This work studies VQA-supervised control-instruction-conditioned trajectory planning for end-to-end autonomous driving. At decision time t, the planner receives a scene context

$$\mathcal{X}_t = \left\{ \mathbf{I}^{\text{front}}_{t-H+1:t}, \mathbf{s}_{t-H+1:t}, \mathbf{g}_t \right\}, \tag{1}$$

where $\mathbf{I}^{\text{front}}_{t-H+1:t}$ denotes the front-camera image history over the most recent H frames, $\mathbf{s}_{t-H+1:t}$ denotes the ego-state history, and $\mathbf{g}_t$ denotes the route prior. Instead of assuming an externally provided language command, the policy receives a driving-oriented VQA prompt q_t, generates a control instruction, and decodes trajectory-level actions:

$$\hat{c}_t \sim p_\theta(c_t \mid \mathcal{X}_t, q_t), \quad (\hat{\mathbf{P}}_t, \hat{\mathbf{u}}_t, \hat{e}_t) = \pi_\theta(\mathcal{X}_t, q_t, \hat{c}_t), \tag{2}$$

where $\hat{\mathbf{P}}_t \in \mathbb{R}^{T_p \times 2}$ is the predicted future path in the ego coordinate system, $\hat{\mathbf{u}}_t \in \mathbb{R}^{T_u}$ is the predicted speed sequence, and $\hat{e}_t \in [0,1]$ estimates whether the generated control instruction can be safely executed in the current traffic scene. During supervised training, the autoregressive language decoder is teacher-forced with the reference control instruction $c_t^\star$; during inference, it consumes its own generated control-instruction tokens. In this setting, language generation is placed before action decoding and serves as an explicit intermediate control interface.

The controlled instruction space is implemented as a finite action-grounding library with 30 canonical control-instruction classes, including 10 maneuver classes, 8 modulation classes, 6 mixed maneuver-modulation classes, and 6 non-executable classes. The mixed classes are formal action-bearing classes rather than exceptions to the branch design. They are used when an instruction jointly changes path geometry and longitudinal behavior, such as lane changing with yielding, merging with deceleration, or bypassing with low speed. Thus, the dual-branch decoder decomposes action factors rather than forcing each generated control instruction into a single branch. Each class is expressed as a short natural-language sentence, such as "follow the route", "change to the left lane" or "stop and wait". Each class is linked to path geometry, speed profile, executability supervision, or conservative fallback behavior, and the complete library is listed in Table 2. The controlled instruction space defines the operational scope of MALT-Drive. It is designed to cover the action-bearing behaviors evaluated in Bench2Drive, including route following, turning, lane changing, merging, bypassing, speed regulation, yielding, stopping, waiting, and unsafe request rejection. During inference, a generated control instruction is used for planning only when it exactly matches or can be normalized to one canonical control-instruction class. If the generated control instruction is ambiguous or unsupported by the library, it is treated as non-executable and routed to the conservative fallback behavior before the decoded path and speed are sent to the control layer.

Different instructions constrain different factors of future motion. Maneuver-oriented instructions mainly determine path geometry, such as lane changing or bypassing, whereas modulation-oriented instructions mainly determine longitudinal behavior, such as slowing down, stopping, or accelerating. Based on this observation, MALT-Drive formulates generated-control-instruction planning as a branch-wise latent alignment problem. The *Maneuver Branch* generates path geometry, and the *Modulation Branch* predicts speed and estimates executability. Both branches are conditioned on the same multimodal context and aligned with generated control-instruction states through branch-specific planning tokens and moment descriptors.

Table 2: Canonical control-instruction library. Each class is represented by a short natural-language control instruction and mapped to path, speed, and executability supervision.

Group	No.	Canonical Control Classes	Target Construction
Maneuver	10	follow the route; turn left; turn right; change to the left lane; change to the right lane; merge left; merge right; overtake on the left; bypass the obstacle on the left; bypass the obstacle on the right	Modify route-consistent future path points and lateral offsets; keep the nominal speed profile unless an additional modulation cue is included.
Modulation	8	keep speed; accelerate gently; decelerate; stop; stop and wait; follow the leading vehicle; yield; resume motion	Preserve the geometric path and modify the speed sequence, terminal speed, or waiting duration.
Mixed	6	change lane and yield; merge with deceleration; bypass with low speed; turn with stop-line compliance; follow the route and wait; resume after yielding	Couple a maneuver template with a modulation template under the same scene state.
Non-executable	6	overtake at a red light; enter a blocked lane; cross a solid boundary; accelerate toward a pedestrian conflict; borrow a wrong-way lane; make an unsupported off-route turn	Preserve the requested control-instruction text, set $e_t^\star = 0$, and assign a conservative fallback path and speed target.

The framework consists of three stages. First, a shared backbone fuses multimodal tokens from visual observations, ego-state history, route priors, VQA prompts, and generated control-instruction tokens. Second, two sets of planning queries extract maneuver and modulation tokens from the fused context. Third, branch-wise action heads and moment-alignment losses jointly supervise path prediction, speed regulation, and executability estimation. Fig. 1 illustrates the planner architecture, including the VLA backbone, branch-specific planning tokens, moment descriptors, action heads, and alignment losses.

3.2 Branch-Specific Planning

The dual-branch decoder follows a hierarchical view of human driving decisions. In a typical driving process, the driver first determines a tactical path-level behavior, such as keeping the lane, changing lanes, merging, or bypassing an obstacle. The driver then regulates longitudinal motion through speed keeping, deceleration, braking, stopping, waiting, or resuming. A feasibility judgment is also involved in this process to prevent unsafe or traffic-rule-violating actions from being executed. MALT-Drive maps this decision hierarchy into the decoder design. The Maneuver Branch corresponds to tactical path selection and predicts future path geometry, while the Modulation Branch corresponds to longitudinal regulation and executability estimation.

Mixed instructions activate both branches through the same generated control-instruction states. For example, “change to the left lane and slow down to yield” induces a lateral path update in the Maneuver Branch and a reduced speed/yielding profile in the Modulation Branch.

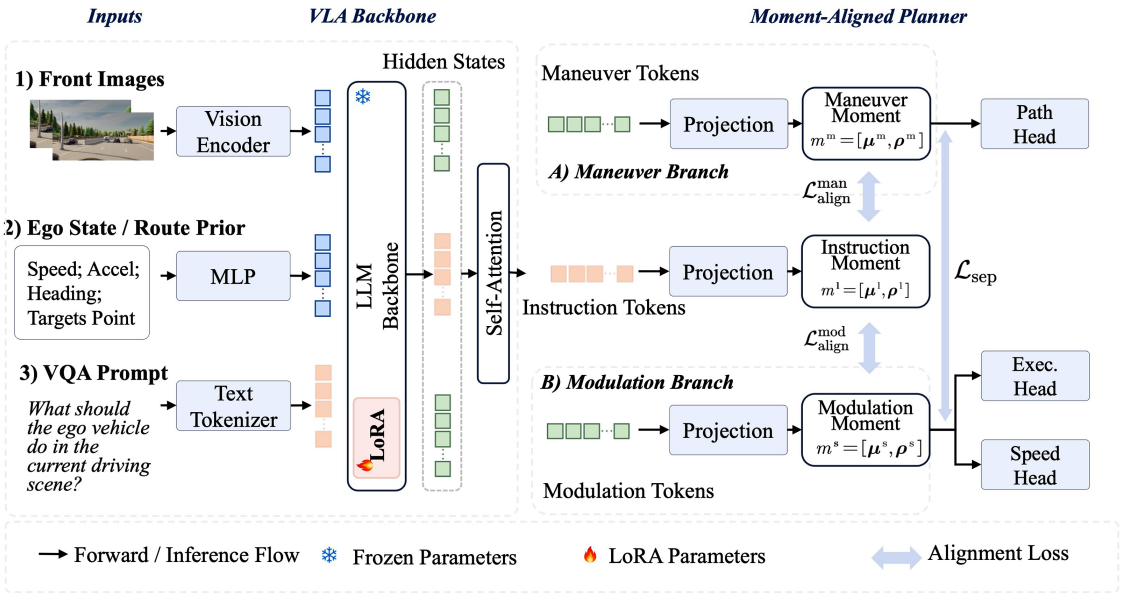

Figure 1: Planner architecture of MALT-Drive. Front images, ego-state/route-prior tokens, and the VQA prompt are encoded by the LoRA-adapted VLA backbone. Self-attention exposes generated-control-instruction states to the Moment-Aligned Planner, where maneuver, control-instruction, and modulation tokens are projected into branch-specific latent spaces. The Maneuver Branch aligns maneuver and control-instruction moments for path prediction, while the Modulation Branch aligns modulation and control-instruction moments for speed prediction and executability estimation. Solid arrows denote forward/inference flow, the snowflake symbol marks frozen parameters, the flame symbol marks trainable LoRA parameters, and bidirectional pale arrows denote alignment losses.

The backbone uses a token-level multimodal fusion paradigm. The visual encoder maps the images into visual tokens $\mathbf{V}_t \in \mathbb{R}^{N_v \times d}$, while the ego-state history and route prior are embedded as structured state tokens $\mathbf{S}_t \in \mathbb{R}^{N_s \times d}$. The VQA prompt is represented by prompt tokens $\mathbf{Q}_t \in \mathbb{R}^{N_q \times d}$, and the control instruction is represented by contextualized control-instruction token states $\mathbf{L}_t \in \mathbb{R}^{N_l \times d}$. These token groups are concatenated and fed into a multimodal Transformer [5,10].

$$\mathbf{H}_t = f_\theta\left([\mathbf{V}_t; \mathbf{S}_t; \mathbf{Q}_t; \mathbf{L}_t]\right) \in \mathbb{R}^{N_h \times d} \tag{3}$$

where $[\cdot;\cdot]$ denotes token concatenation and d is the hidden dimension.

Rather than decoding all future motion from a single pooled context vector, MALT-Drive introduces two sets of learnable planning queries, $\mathbf{U}_{\text{man}} \in \mathbb{R}^{K_{\text{man}} \times d}$ and $\mathbf{U}_{\text{mod}} \in \mathbb{R}^{K_{\text{mod}} \times d}$. They attend to the fused context and produce branch-specific planning tokens:

$$\mathbf{M}_t = D_{\text{man}}(\mathbf{U}_{\text{man}}, \mathbf{H}_t) \in \mathbb{R}^{K_{\text{man}} \times d}, \quad \mathbf{R}_t = D_{\text{mod}}(\mathbf{U}_{\text{mod}}, \mathbf{H}_t) \in \mathbb{R}^{K_{\text{mod}} \times d} \tag{4}$$

where D_{man} and D_{mod} are lightweight Transformer query decoders with independent parameters. The Maneuver Branch tokens $\mathbf{M}_t$ are used for path prediction, whereas the Modulation Branch tokens $\mathbf{R}_t$ are used for speed prediction and executability estimation. In practice, $K_{\text{man}} = T_p$ and $K_{\text{mod}} = T_u$, so that token indices correspond to the prediction horizon. This design allows the two branches to extract different types of control evidence from the same generated control instruction and scene context: maneuver tokens focus on route geometry and lateral deviation, while modulation tokens focus on speed, temporal behavior, and safety feasibility.

Maneuver branch. The Maneuver Branch predicts local path offsets in ego coordinates. For each future step $\tau \in \{1, \ldots, T_p\}$, the corresponding maneuver token is mapped to a displacement vector, and the final trajectory is obtained by cumulative integration.

$$\Delta\hat{\mathbf{p}}_{t,\tau} = W_p \mathbf{M}_{t,\tau}, \quad \hat{\mathbf{p}}_{t,\tau} = \sum_{j=1}^{\tau} \Delta\hat{\mathbf{p}}_{t,j}, \quad \hat{\mathbf{P}}_t = [\hat{\mathbf{p}}_{t,1}, \ldots, \hat{\mathbf{p}}_{t,T_p}]^\top \tag{5}$$

where $W_p \in \mathbb{R}^{2\times d}$. This local-offset formulation preserves path continuity and makes instruction-induced geometric changes easier to represent. For example, a generated lane-change or obstacle-bypass instruction can affect a coherent sequence of maneuver tokens rather than weakly perturbing a single global trajectory embedding.

Modulation branch. The Modulation Branch predicts the longitudinal speed curve and the executability score. Each modulation token is mapped to a non-negative speed value, while the pooled branch state estimates whether the instruction is safely executable:

$$\hat{u}_{t,\tau} = \mathrm{softplus}\left(\mathbf{w}_u^\top \mathbf{R}_{t,\tau}\right), \quad \hat{e}_t = \mathrm{sigmoid}\left(\mathbf{w}_e^\top \mathrm{mean}(\mathbf{R}_t)\right). \tag{6}$$

The predicted speed sequence $\hat{\mathbf{u}}_t$ describes longitudinal behaviors such as maintaining speed, braking, stopping, waiting, or resuming motion. The executability score $\hat{e}_t$ provides an explicit rejection signal that helps reduce the risk of unsafe or infeasible generated control instructions being directly converted into vehicle motion.

The executability head is attached to the Modulation Branch because unsafe-instruction rejection is mainly determined by temporal motion feasibility. Red-light compliance, stop-line holding, gap acceptance, car-following, and pedestrian yielding are all reflected in the predicted speed evolution and waiting behavior. The pooled modulation state therefore provides a compact summary for deciding whether the generated control instruction can be safely executed. A separate classifier would introduce an additional feasibility representation that is less directly tied to the decoded speed profile. The rollout checker is used to construct executability labels for counterfactual supervision, while the learned sigmoid head provides a lightweight inference-time rejection signal.

3.3 Branch-Wise Moment Alignment

The dual-branch decoder separates maneuver and modulation states, but this separation alone does not define how control-instruction token states should be coupled with trajectory-producing planning states. MALT-Drive therefore introduces branch-wise moment alignment at the token-set descriptor level. Token-level alignment emphasizes local token correspondences, global embedding alignment compresses instruction and planning states into one vector, and standard contrastive alignment is usually applied at the sample-embedding level. MALT-Drive instead computes branch-level descriptors for instruction-token states and planning-token states, so the Maneuver Branch is aligned with path-generation tokens and the Modulation Branch is aligned with speed/executability tokens.

For branch $b \in \{\mathrm{man}, \mathrm{mod}\}$, the projected language and planning tokens are defined as

$$\mathbf{L}_t^b = \Pi_L^b(\mathbf{L}_t), \quad \mathbf{A}_t^b = \begin{cases} \Pi_A^{\mathrm{man}}(\mathbf{M}_t), & b = \mathrm{man}, \\ \Pi_A^{\mathrm{mod}}(\mathbf{R}_t), & b = \mathrm{mod}. \end{cases} \tag{7}$$

where Π_L^b and Π_A^b are learnable projection layers that map language and planning tokens into the same branch-specific latent space. They are applied token-wise and output d-dimensional branch tokens, so $\mathbf{L}_t^b \in$

$\mathbb{R}^{N_l \times d}$ and $\mathbf{A}_t^b \in \mathbb{R}^{K_b \times d}$, where $K_b = K_{\text{man}}$ for b = man and $K_b = K_{\text{mod}}$ for b = mod. Thus, $\mathbf{L}_t^{\text{man}}$ and $\mathbf{A}_t^{\text{man}}$ form the language-planning pair for path geometry, while $\mathbf{L}_t^{\text{mod}}$ and $\mathbf{A}_t^{\text{mod}}$ form the language-planning pair for speed regulation and executability. The following moment descriptor is computed on these paired token sets rather than on a single global pooled feature.

Moment descriptor. For each branch b, the descriptor operator is applied to either the projected language tokens $\mathbf{L}_t^b$ or the projected planning tokens $\mathbf{A}_t^b$. Thus, $\mathbf{X} \in \{\mathbf{L}_t^b, \mathbf{A}_t^b\}$, where $n = N_l$ for control-instruction token states and $n = K_b$ for branch planning tokens. Let $\mathbf{1} \in \mathbb{R}^n$ denote an all-one column vector. For a token matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, both first-order and grouped second-order statistics are computed. The first-order component is

$$\boldsymbol{\mu}(\mathbf{X}) = \frac{1}{n}\mathbf{1}^\top \mathbf{X}, \quad \mathbf{X} = [\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(G)}], \quad \mathbf{X}^{(g)} \in \mathbb{R}^{n \times d_g} \tag{8}$$

where $d_g = d/G$ and G is the number of channel groups, with d divisible by G. For each group, the mean is removed and a covariance-like relation sketch is computed, assuming $n > 1$:

$$\bar{\mathbf{X}}^{(g)} = \mathbf{X}^{(g)} - \mathbf{1}\mu\left(\mathbf{X}^{(g)}\right), \qquad \mathbf{C}\left(\mathbf{X}^{(g)}\right) = \frac{1}{n-1}\left(\bar{\mathbf{X}}^{(g)}\right)^\top \bar{\mathbf{X}}^{(g)}. \tag{9}$$

Since $\mathbf{C}(\mathbf{X}^{(g)})$ is symmetric, only its upper-triangular entries are retained. The grouped second-order descriptor is then written as

$$\boldsymbol{\rho}(\mathbf{X}) = W_\rho \underset{g=1}{\overset{G}{\text{concat}}} \text{vec}_\triangle\left(\mathbf{C}(\mathbf{X}^{(g)})\right) \tag{10}$$

where $\text{vec}_\triangle(\cdot)$ denotes upper-triangular vectorization and W_ρ projects the concatenated sketch into a compact descriptor space. The final moment descriptor combines the two components:

$$\mathbf{z}(\mathbf{X}) = \text{Norm}\left(W_z[\boldsymbol{\mu}(\mathbf{X}); \boldsymbol{\rho}(\mathbf{X})]\right) \tag{11}$$

where $\text{Norm}(\cdot)$ denotes ℓ_2 normalization, and $\mathbf{z}(\cdot)$ is the moment-descriptor operator that maps a projected token set to a unit-norm vector in a compact alignment space. In this descriptor, the first-order term summarizes the dominant control semantics of a branch-specific token set, while the grouped second-order term preserves token-relation patterns that are often lost by simple pooling. This descriptor-level design is suitable for the language-to-trajectory interface considered here because it aligns branch-level token sets rather than isolated token pairs or a single global state.

Computationally, the grouped second-order sketch avoids forming a full $d \times d$ covariance matrix. For a token set $\mathbf{X} \in \mathbb{R}^{n \times d}$, mean pooling costs $O(nd)$. The grouped covariance term computes G matrices of size $d_g \times d_g$, where $d_g = d/G$, giving a second-order cost of $O(Gnd_g^2) = O(nd^2/G)$. The number of retained upper-triangular entries is $Gd_g(d_g+1)/2$. Thus, increasing G reduces the quadratic group dimension, while very small d_g may weaken relation modeling. The default setting uses $G = 8$ to balance latency and alignment quality.

The moment descriptor gives branch-specific language-planning alignment a concrete action meaning. For example, for the instruction "change to the left lane", the Maneuver Branch links the "left lane" cue to path tokens that generate lateral displacement, while the Modulation Branch maintains the nominal speed when no braking or yielding cue appears. The first-order moment captures the dominant control semantics, and the grouped second-order sketch preserves relations between instruction cues and action-producing planning tokens.

Alignment objective. In the mini-batch objective, the decision-time subscript t is omitted and i, j index training tuples. For a mini-batch of size B, the positive pair in branch b is the matched language-planning descriptor pair from the same training tuple, i.e., $\left(\mathbf{z}(\mathbf{L}_i^b), \mathbf{z}(\mathbf{A}_i^b)\right)$. Mismatched pairs $\left(\mathbf{z}(\mathbf{L}_i^b), \mathbf{z}(\mathbf{A}_j^b)\right)$ with $j \neq i$ and $\left(\mathbf{z}(\mathbf{L}_k^b), \mathbf{z}(\mathbf{A}_i^b)\right)$ with $k \neq i$ serve as sample-level in-batch negatives. This contrastive definition of negative pairs is independent of the executability label. The similarity between the language descriptor of sample i and the planning descriptor of sample j in branch b is

$$s_{ij}^b = \mathbf{z}\left(\mathbf{L}_i^b\right)^{\mathrm{T}} \mathbf{z}\left(\mathbf{A}_j^b\right). \tag{12}$$

Since both descriptors are ℓ_2-normalized, s_{ij}^b is equivalent to cosine similarity. The diagonal entries s_{ii}^b correspond to positives, whereas the off-diagonal entries are used as negatives in the corresponding row-wise or column-wise softmax. Based on this similarity matrix, the language-to-planning and planning-to-language matching probabilities are

$$p_{A_j|L_i}^b = \frac{\exp(s_{ij}^b/\tau)}{\sum_{k=1}^B \exp(s_{ik}^b/\tau)}, \quad p_{L_i|A_j}^b = \frac{\exp(s_{ij}^b/\tau)}{\sum_{k=1}^B \exp(s_{kj}^b/\tau)} \tag{13}$$

where τ is the temperature. The branch-wise alignment loss is defined as

$$\mathcal{L}_{\text{align}}^b = -\frac{1}{2B} \sum_{i=1}^B \left[\log p_{A_i|L_i}^b + \log p_{L_i|A_i}^b\right] \tag{14}$$

This symmetric form encourages each generated control instruction to retrieve its corresponding branch-specific planning state and, conversely, encourages each planning state to remain identifiable in the language space. In the maneuver branch, this constraint directly couples instruction semantics with the path-producing tokens $\mathbf{M}_t$; in the modulation branch, it couples instruction semantics with the speed- and executability-producing tokens $\mathbf{R}_t$.

To reduce collapse between the two branches, a weak branch-separation regularizer is further introduced. For compact notation, let $\mathbf{z}_{L,i}^b = \mathbf{z}(\mathbf{L}_i^b)$ and $\mathbf{z}_{A,i}^b = \mathbf{z}(\mathbf{A}_i^b)$:

$$\mathcal{L}_{\text{sep}} = \frac{1}{B} \sum_{i=1}^B \left[\cos^2\left(\mathbf{z}_{L,i}^{\text{man}}, \mathbf{z}_{L,i}^{\text{mod}}\right) + \cos^2\left(\mathbf{z}_{A,i}^{\text{man}}, \mathbf{z}_{A,i}^{\text{mod}}\right)\right] \tag{15}$$

The separation term is applied as a weak regularizer. It encourages the maneuver and modulation descriptors to specialize in different control evidence while keeping both branches conditioned on the same generated control-instruction tokens. Mixed instructions are represented by paired labels across the two branches. For example, "change to the left lane and slow down to yield" provides a left-lane-change target for the Maneuver Branch and a deceleration/yield target for the Modulation Branch. The regularizer separates branch channels; it does not suppress shared instruction conditioning.

3.4 Counterfactual Control-Instruction Supervision

Imitation learning usually provides only one expert trajectory for a scene, making it difficult to determine whether the model follows language instructions in the intended way. To make this dependency observable, MALT-Drive constructs counterfactual control-instruction rollouts. The symbol $c_t^{\star(m)}$ denotes the reference control instruction used as the teacher-forcing target during training, whereas $\hat{c}_t$ denotes

the generated control instruction autoregressively produced by the model during inference. For each scene context $\mathcal{X}_t$, a VQA prompt–control-instruction set is constructed:

$$\mathcal{D}_t^{\mathrm{vqa}} = \{(q_t^{(m)}, c_t^{\star(m)})\}_{m=1}^{N_t^{\mathrm{vqa}}} \tag{16}$$

where N_t^{vqa} denotes the number of VQA prompt–control-instruction pairs attached to the same scene. Each control instruction is associated with a path target $\mathbf{P}_t^{\star(m)} \in \mathbb{R}^{T_p \times 2}$, a speed target $\mathbf{u}_t^{\star(m)} \in \mathbb{R}^{T_u}$, and an executability label $e_t^{\star(m)} \in \{0,1\}$. The reference control instruction is paired with the expert path and speed, whereas additional counterfactual control instructions are generated according to local topology and interaction-target constraints. For all counterfactual tuples, the scene context $\mathcal{X}_t$ remains fixed; only the prompt–control-instruction pair and its corresponding motion-executability target vary.

Fig. 2 summarizes the counterfactual rollout mechanism. It shows how the same scene and VQA prompt are paired with reference, alternative executable, and non-executable candidate control instructions, how rollout evaluation assigns executable/non-executable labels, and how the training objective backpropagates through the trainable planner components while remaining distinct from the data-forward flow used to generate candidate supervision.

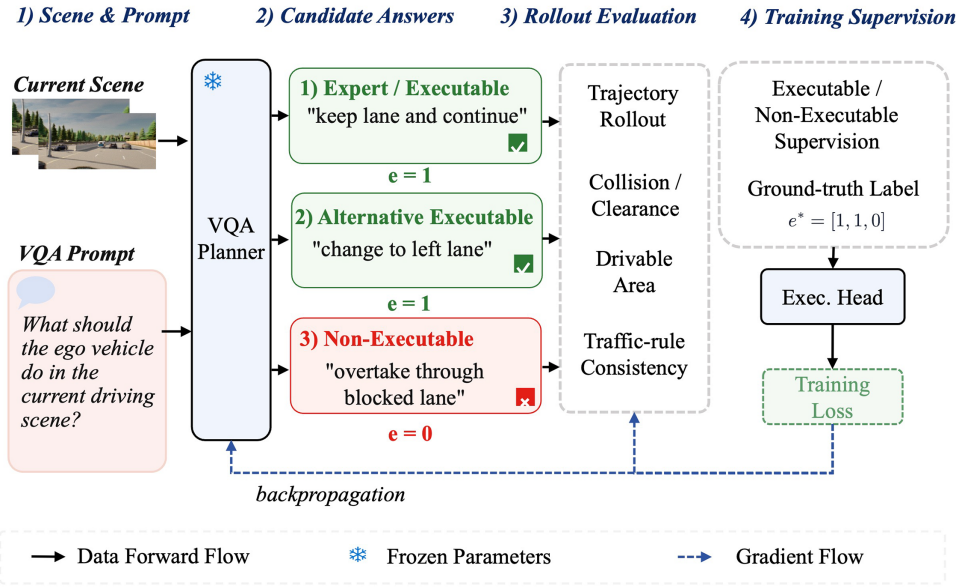

Figure 2: Counterfactual control-instruction rollout and training supervision. Given the same scene and VQA prompt, the VQA planner instantiates reference, alternative executable, and non-executable candidate control instructions. Candidate path-speed rollouts are checked for trajectory feasibility, collision/clearance, drivable-area containment, and traffic-rule consistency, producing executability labels for the training target. The executability head is supervised by these labels, and the dashed blue path indicates the training-time backpropagation flow, separated from the solid data-forward flow.

VQA library: Table 2 lists the complete canonical control-instruction library used in MALT-Drive. The library contains 30 canonical classes. Maneuver classes mainly change future path geometry. Modulation classes mainly change the longitudinal speed profile or waiting behavior. Mixed classes jointly affect path and speed targets. Non-executable classes describe unsafe or rule-violating requests and are supervised with a conservative fallback target and an executability label of zero. The coverage of the library is checked using Bench2Drive metadata and expert trajectories, including route direction, lane availability, traffic-light state, obstacle layout, leading-object distance, conflict zones, and drivable-area constraints. These cues activate

canonical classes such as lane changing, stopping and waiting, obstacle bypassing, or lane borrowing. During inference, generated control instructions that match or can be normalized to a canonical class are passed to the executability check, whereas ambiguous or unsupported generated control instructions are treated as out of scope and routed directly to the conservative fallback behavior.

Counterfactual rollout: For each scene state, one reference control instruction consistent with the expert behavior is retained, and additional maneuver, modulation, and non-executable variants are instantiated. Maneuver counterfactuals are generated according to local topology and obstacle conditions. A lane-change instruction is activated when the adjacent lane is reachable and the transition corridor is free. An obstacle-bypass or lane-borrowing instruction is activated when a parked vehicle, construction area, or blocked lane affects the nominal route. Modulation counterfactuals are generated according to longitudinal constraints. Speed keeping or acceleration is used when the front gap, speed limit, and route curvature allow forward motion. Deceleration, stopping, or waiting is used near red lights, stop lines, crossing pedestrians, slow leading vehicles, or yield-required gaps. Non-executable counterfactuals are generated for unsafe-instruction rejection, including red-light running, collision-seeking motion, blocked-lane traversal, and non-drivable-area entry. For these samples, the control-instruction text is preserved, the executability label is set to $e_t^{\star(m)} = 0$, and the motion target is replaced by a conservative fallback trajectory.

A requested instruction is first interpreted by its intended maneuver and longitudinal effect, and the candidate rollout is then checked against the current scene constraints. If the intended action violates traffic rules, route topology, drivable-area boundaries, or short-horizon collision constraints, the instruction is labeled as non-executable. For example, an instruction such as "overtake the front vehicle" is rejected at a red light when the rollout would cross the stop line or enter the conflict area before passage is allowed; the fallback target stops or waits before the stop line. In these non-executable tuples, the original control-instruction text is retained, $e_t^{\star(m)} = 0$, and the path/speed target is replaced by the conservative fallback target.

All counterfactual tuples are constructed using the same deterministic procedure summarized in Algorithm 1. For each scene, the reference control instruction is retained, and additional canonical control instructions are activated according to lane topology, obstacle layout, longitudinal constraints, and traffic-rule states. Maneuver classes modify the route-consistent path, modulation classes modify the speed profile, and mixed classes modify both targets. Each candidate is then evaluated through a 4.0 s rollout with a 0.5 s interval. A candidate that satisfies all safety and topology checks receives its candidate path-speed target and an executability label of one. A failed candidate retains its requested control-instruction text but is assigned the conservative fallback target and an executability label of zero. The same rules and thresholds are used for all counterfactual samples.

Algorithm 1: Counterfactual control-instruction rollout construction

Require: scene context $\mathcal{X}_t$, reference control instruction and target $(c_t^\star, \mathbf{P}_t^\star, \mathbf{u}_t^\star)$, canonical control-instruction library $\mathcal{C}$, fallback target $(\mathbf{P}_t^{\mathrm{fb}}, \mathbf{u}_t^{\mathrm{fb}})$

Ensure: counterfactual tuples $\{(c_t^{\star(m)}, \mathbf{P}_t^{\star(m)}, \mathbf{u}_t^{\star(m)}, e_t^{\star(m)})\}_m$

1: Retain the expert tuple with $e_t^\star = 1$.
2: Activate canonical control instructions using route direction, adjacent-lane reachability, obstacle layout, front gap, speed limit, traffic-light state, stop lines, and pedestrian/cyclist conflicts.
3: **for** each activated control instruction $c_t^{\star(m)}$ **do**
4: Generate $(\mathbf{P}_t^{\mathrm{cand}(m)}, \mathbf{u}_t^{\mathrm{cand}(m)})$:
5: maneuver classes modify the path and preserve the nominal speed;
6: modulation classes preserve the path and modify the speed;

(Continued)

Algorithm 1 (continued)

mixed classes modify both the path and speed.
Roll out the ego footprint for $T = 4.0$ s with $\Delta t = 0.5$ s.
Set $\chi_t^{(m)} \leftarrow 1$.
if dynamic-object clearance < 0.50 m or TTC < 1.50 s **then**
    $\chi_t^{(m)} \leftarrow 0$
end if
if any ego footprint leaves the drivable polygon with a 0.20 m margin **then**
    $\chi_t^{(m)} \leftarrow 0$
end if
if the rollout crosses a solid boundary or travels in the wrong direction for more than 5.0 m **then**
    $\chi_t^{(m)} \leftarrow 0$
end if
if the rollout crosses a red-light or stop-line boundary without stopping at least 1.0 m before the line **then**
    $\chi_t^{(m)} \leftarrow 0$
end if
if the target lane is blocked within 15.0 m and lateral clearance is below 0.60 m **then**
$\chi_t^{(m)} \leftarrow 0$
end if
if a pedestrian/cyclist conflict zone intersects the rollout and ego speed exceeds 0.50 m/s **then**
    $\chi_t^{(m)} \leftarrow 0$
end if
if $\chi_t^{(m)} = 1$ **then**
Assign the candidate target:
    $(\mathbf{P}_t^{\star(m)}, \mathbf{u}_t^{\star(m)}, e_t^{\star(m)}) \leftarrow (\mathbf{P}_t^{\mathrm{cand}(m)}, \mathbf{u}_t^{\mathrm{cand}(m)}, 1)$.
else
    Assign the fallback target:
        $(\mathbf{P}_t^{\star(m)}, \mathbf{u}_t^{\star(m)}, e_t^{\star(m)}) \leftarrow (\mathbf{P}_t^{\mathrm{fb}}, \mathbf{u}_t^{\mathrm{fb}}, 0)$.
end if
end for
return all nominal and counterfactual tuples

The fallback target follows the valid route-aligned lane and applies a monotonically decreasing speed profile that stops the ego vehicle at the nearest safe stopping position within the prediction horizon.

3.5 Training and Inference

All losses below are written for one training tuple and then averaged over the mini-batch. To keep notation compact, the counterfactual index m is omitted unless necessary. The VQA-based control-instruction generator is trained with the autoregressive negative log-likelihood. For a reference control instruction $c_t^\star = \{w_{t,\ell}^\star\}_{\ell=1}^{N_c}$, where $w_{t,\ell}^\star$ denotes the ℓ-th control-instruction token,

$$\mathcal{L}_{\mathrm{vqa}} = -\frac{1}{N_c}\sum_{\ell=1}^{N_c}\log p_\theta\left(w_{t,\ell}^\star \mid w_{t,<\ell}^\star, \mathcal{X}_t, q_t\right) \tag{17}$$

During SFT, the control-instruction token states used by the planning decoder are obtained from teacher-forced reference control instructions, while during inference they are obtained from the generated control instruction. This train-inference difference is an exposure-bias source for the language-to-trajectory interface: an incorrect or unstable generated control instruction can shift the control-instruction token states received by the path, speed, and executability heads. Conditioned on the control-instruction token states, the Maneuver Branch is supervised by path regression, the Modulation Branch is supervised by speed regression, and the executability head is supervised by binary cross-entropy:

$$\begin{aligned}
\mathcal{L}_{\text{path}} &= \frac{1}{T_p}\sum_{\tau=1}^{T_p}\ell_{\text{sl1}}\left(\hat{\mathbf{p}}_{t,\tau},\mathbf{p}^{\star}_{t,\tau}\right),\\
\mathcal{L}_{\text{speed}} &= \frac{1}{T_u}\sum_{\tau=1}^{T_u}\ell_{\text{sl1}}\left(\hat{u}_{t,\tau},u^{\star}_{t,\tau}\right),\\
\mathcal{L}_{\text{exe}} &= -e^{\star}_t\log\hat{e}_t-(1-e^{\star}_t)\log(1-\hat{e}_t)
\end{aligned} \tag{18}$$

where ℓ_{sl1} denotes the smooth-ℓ_1 loss. For non-executable tuples, $\mathbf{P}^{\star}_t$ and $\mathbf{u}^{\star}_t$ correspond to the fallback target, while $e^{\star}_t = 0$ trains the executability head to reject the requested instruction. The full supervised objective is

$$\mathcal{L}_{\text{SFT}} = \lambda_c\mathcal{L}_{\text{vqa}}+\lambda_p\mathcal{L}_{\text{path}}+\lambda_u\mathcal{L}_{\text{speed}}+\lambda_e\mathcal{L}_{\text{exe}}+\lambda_a\left(\mathcal{L}^{\text{man}}_{\text{align}}+\mathcal{L}^{\text{mod}}_{\text{align}}\right)+\lambda_s\mathcal{L}_{\text{sep}} \tag{19}$$

This objective is applied to both nominal and counterfactual tuples; therefore, multiple samples may share the same scene but have different control instructions and different motion-executability targets. Training begins by warm-starting the VQA generator and planning heads using nominal expert prompt–control-instruction pairs. Counterfactual control-instruction rollouts are then introduced so that the same scene is associated with multiple instruction-conditioned outcomes. Finally, non-executable samples are upweighted to improve executability calibration and reduce unsafe-instruction acceptance. Throughout these SFT stages, the alignment losses use the matched language-planning descriptors from the same tuple as positives, including counterfactual tuples derived from the same scene.

After supervised fine-tuning, the policy can be further refined by reinforcement fine-tuning [18]. For each scenario prompt, a group of G_r candidate outputs $o_i = (\hat{c}_{t,i},\hat{\mathbf{P}}_{t,i},\hat{\mathbf{u}}_{t,i},\hat{e}_{t,i})$ is sampled from the current policy. Each output receives a scalar reward and a group-relative advantage:

$$\begin{aligned}
r_i &= r_{\text{drive},i}+\eta_{\text{align}}r_{\text{align},i}+\eta_{\text{fmt}}r_{\text{fmt},i}-\eta_{\text{unsafe}}r_{\text{unsafe},i},\\
A_i &= \frac{r_i-\text{mean}(\{r_j\}_{j=1}^{G_r})}{\text{std}(\{r_j\}_{j=1}^{G_r})+\varepsilon_{\text{std}}}
\end{aligned} \tag{20}$$

where r_{drive} summarizes route progress, collision avoidance, comfort, and traffic-rule compliance; r_{align} measures consistency between the generated control instruction and decoded action; r_{fmt} preserves the canonical control-instruction format; and r_{unsafe} penalizes acceptance of non-executable instructions. The KL-regularized optimization objective is

$$\mathcal{J}_{\text{RFT}}(\theta) = \frac{1}{G_r}\sum_{i=1}^{G_r}\min\Big(\rho_i A_i,\text{clip}(\rho_i,1-\varepsilon,1+\varepsilon)A_i\Big)-\beta D_{\text{KL}}\left(\pi_\theta\,\|\,\pi_{\text{SFT}}\right) \tag{21}$$

where

$$\rho_i = \exp\left[\log\pi_\theta(o_i\mid\mathcal{X}_t,q_t)-\log\pi_{\theta_{\text{old}}}(o_i\mid\mathcal{X}_t,q_t)\right] \tag{22}$$

$\pi_\theta(o_i \mid \mathcal{X}_t, q_t)$ denotes the policy probability or density of the sampled output, and π_{SFT} is the frozen supervised reference policy. The KL term prevents the refined policy from drifting away from the controlled instruction space and the branch-wise alignment learned during supervised fine-tuning. During RFT, the optimization uses sampled outputs from the current policy, including the generated control instruction, path sequence, speed sequence, and executability score. This exposes the planner to inference-style control-instruction token states after SFT and serves as a partial mitigation for the teacher-forcing mismatch. The remaining gap is measured by the control-instruction-source diagnostic in Section 4.4.

During inference, the model first autoregressively generates the control instruction and then decodes the path, speed sequence, and executability score from the generated control-instruction states. No teacher forcing is used at inference:

$$\hat{c}_t \sim p_\theta(c_t \mid \mathcal{X}_t, q_t),\ (\hat{\mathbf{P}}_t, \hat{\mathbf{u}}_t, \hat{e}_t) = \pi_\theta(\mathcal{X}_t, q_t, \hat{c}_t) \tag{23}$$

where γ is the executability threshold used to convert the predicted executability score into an execute-or-reject decision. The canonical-class check is applied before the final control command is issued. If a generated control instruction cannot be exactly matched or normalized to the 30-class library, it is marked as unsupported and routed to the conservative fallback branch. For supported classes, the predicted executability score then determines whether the decoded motion can be executed. If $\hat{e}_t \geq \gamma$, the planner executes the generated-control-instruction-conditioned path and speed profile. If $\hat{e}_t < \gamma$, the decoded pair $(\hat{\mathbf{P}}_t, \hat{\mathbf{u}}_t)$ is replaced by the fallback pair $(\mathbf{P}_t^{\mathrm{fb}}, \mathbf{u}_t^{\mathrm{fb}})$, while the generated control instruction is treated as rejected rather than executed. This two-stage rule preserves sensitivity to supported valid instructions while reducing the likelihood that unsupported or unsafe generated control instructions directly influence the control loop.

4 Experiments

4.1 Experimental Settings

Dataset. Closed-loop driving performance is evaluated on Bench2Drive, a CARLA-based benchmark designed to assess multiple abilities in end-to-end autonomous driving covering dense urban interaction, adverse weather, and long-tail traffic events [19]. Following the official protocol, each method is evaluated on the complete set of 220 routes. To further evaluate cross-benchmark generalization beyond the CARLA-based setting, an additional experiment is conducted on the NAVSIM benchmark [20]. NAVSIM is built from real-world driving logs from nuScenes and provides a different planning evaluation setting with different traffic distributions, sensor characteristics, and behavior patterns. Following the official NAVSIM protocol, the reported metrics include no at-fault collision (NC), drivable-area compliance (DAC), ego progress (EP), time-to-collision margin (TTC), comfort (Comf.), and the Planning Driving Metric Score (PDMS).

VQA Generation. The VQA supervision follows the driving-oriented VQA design of DriveLM [5]. Using simulator metadata, route priors, and expert trajectories [21], this QA taxonomy is instantiated on Bench2Drive anchor frames while keeping the prompts short and planning-oriented. To make the language-action correspondence easier to identify, counterfactual examples are further constructed from the Dreamer data in SimLingo [10]. For each scene, the nominal VQA prompt–control-instruction pair is replaced with alternative control instructions that specify a different maneuver or speed regulation, the corresponding candidate path and speed targets are generated, and an executability label is assigned through rollout-based checks for collisions, drivable-area violations, and traffic-rule compliance.

Evaluation Metrics. MALT-Drive is evaluated along three axes: *Closed-loop Driving*, *VQA Generation*, and *Instruction-Action Consistency*.

Closed-loop Driving: Closed-loop driving performance follows the official Bench2Drive protocol. The reported metrics include Driving Score (DS), Success Rate (SR), and the success rates of five ability groups:

Merging, Overtaking, Emergency Brake, Give Way, and Traffic Sign [19]. DS summarizes route completion under infraction penalties, while SR measures the percentage of successfully completed routes over the 220-route closed-loop benchmark. The ability-group success rates further evaluate scenario-specific driving competence under interactive and rule-constrained traffic conditions.

VQA Generation: The autoregressive VQA-based control-instruction generator is evaluated with GPT Score and SPICE, following the driving VQA evaluation protocols used in DriveLM and SimLingo [5,10]. GPT Score evaluates semantic correctness by providing a language-model evaluator with the question, the reference control instruction, and the generated control instruction, and asking it to assign a score from 0 to 100. SPICE parses the generated and reference control instructions into semantic proposition graphs and computes the tuple-level F-score [22]. These metrics assess whether the generated control instruction preserves the intended driving semantics.

Instruction-Action Consistency: Instruction-Action Consistency F1 (IAC-F1) is reported as a trajectory-grounded instruction-action consistency metric that evaluates whether the planned motion realizes the intended control semantics of each reference control instruction. Following decision-planning consistency evaluation in VLM-based driving [23], the reference control instruction and executability annotation define the instruction-side label, while the predicted path, speed, and executability score define the action-side label through deterministic kinematic rules. Concretely, signed lateral displacement assigns maneuver labels, speed trend and terminal speed assign speed-regulation labels, and $\hat{e} < \gamma$ assigns the non-executable label. Let $\mathcal{Y} = \mathcal{Y}_{\text{man}} \cup \mathcal{Y}_{\text{spd}} \cup \mathcal{Y}_{\text{exe}}$ denote the structured label space for maneuver, speed-regulation, and executability classes. For each class $y \in \mathcal{Y}$, precision and recall are computed as

$$\text{Prec}_y = \frac{\text{TP}_y}{\text{TP}_y + \text{FP}_y}, \ \text{Rec}_y = \frac{\text{TP}_y}{\text{TP}_y + \text{FN}_y}, \tag{24}$$

where TP_y denotes matched instruction-side and action-side assignments to class y, FP_y denotes action-side assignments to class y without the corresponding instruction-side label, and FN_y denotes instruction-side assignments to class y not realized by the action-side label. IAC-F1 is the macro-F1 over all structured control classes:

$$\text{IAC-F1} = \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} \frac{2\,\text{Prec}_y\,\text{Rec}_y}{\text{Prec}_y + \text{Rec}_y}. \tag{25}$$

To make the metric implementation explicit, fixed kinematic thresholds are used for action-label assignment. Let $\Delta\hat{y}_{T_p}$ denote the signed terminal lateral displacement of the predicted path, $\Delta\hat{u} = \hat{u}_{T_u} - \hat{u}_1$ denote the speed change over the prediction horizon, and $\hat{u}_{T_u}$ denote the terminal speed. The maneuver label is assigned as left-change if $\Delta\hat{y}_{T_p} > \delta_y$, right-change if $\Delta\hat{y}_{T_p} < -\delta_y$, and route-following otherwise. The speed-regulation label is assigned as stopping/waiting if $\hat{u}_{T_u} < v_{\text{stop}}$, acceleration if $\Delta\hat{u} > \delta_u$, deceleration if $\Delta\hat{u} < -\delta_u$, and speed-keeping otherwise. The non-executable label is assigned when $\hat{e} < \gamma$. In all experiments, $\delta_y = 0.6\,\text{m}$, $\delta_u = 0.5\,\text{m/s}$, $v_{\text{stop}} = 0.3\,\text{m/s}$, and $\gamma = 0.5$ are fixed after validation and kept unchanged for all evaluated variants.

Implementation Details. The input consists of front-camera RGB images, ego-state history, an assigned global route, and a driving-oriented VQA prompt. Four frames are sampled at 2 Hz from the Bench2Drive annotations. Each front image is resized to 1024 × 512 and split into two 512 × 512 horizontal tiles. Each tile is then resized to 448 × 448 and encoded by the InternVL2 visual encoder. The ego-state vector contains velocity, acceleration, and heading. The route prior is obtained from the assigned CARLA global route, transformed into the ego coordinate system, and projected into route tokens through a two-layer MLP.

VLA Backbone: The VLA backbone is instantiated as InternVL2-1B from the Mini-InternVL family [24], which consists of InternViT-300M-448px as the visual encoder and Qwen2-0.5B-Instruct as the language model. The language model is adapted with LoRA inserted into all linear layers, with rank $r = 32$, scaling factor $\alpha = 64$, and dropout 0.1. The model first autoregressively generates a control instruction constrained by the canonical control-instruction library. The hidden states of the generated control-instruction tokens are then reused by the dual-branch action decoder, allowing the generated control instruction to serve as the intermediate control interface for trajectory decoding.

Action Targets: The action representation follows a disentangled path-and-speed design. The Maneuver Branch predicts $T_p = 8$ geometric path points in the ego coordinate system. The Modulation Branch predicts $T_u = 8$ speed targets at future time offsets $\{0.5, 1.0, \ldots, 4.0\}$ s, obtained from the expert trajectory and speed annotations. Both branches use 8 learnable planning queries, and the grouped moment descriptor uses $G = 8$ channel groups.

Supervised Fine-Tuning: SFT contains three stages and lasts 12 epochs. In the first stage, the model is trained on nominal expert VQA-action tuples for 3 epochs. In the second stage, it is trained on a balanced 1:1 mixture of nominal and counterfactual VQA tuples for 7 epochs. In the third stage, non-executable VQA tuples are upweighted to 40% of each mini-batch for 2 epochs to improve unsafe-instruction rejection. Training uses AdamW with weight decay 0.1, one-cycle cosine scheduling, and a linear warm-up over the first 4% of iterations. The learning rate is 3×10^{-5} for backbone adaptation parameters and 2×10^{-4} for newly initialized modules. Mixed precision and gradient clipping at 1.0 are used. The supervised loss weights are fixed as $\lambda_c = 1.0$, $\lambda_p = 1.0$, $\lambda_u = 0.5$, $\lambda_e = 0.5$, $\lambda_a = 0.2$, and $\lambda_s = 0.05$. Models are trained on 8 NVIDIA A100-80GB GPUs with a global batch size of 64. The default 3/7/2 schedule and the 40% non-executable ratio are used in all experiments.

Reinforcement Fine-Tuning: After SFT, reinforcement fine-tuning is performed for one epoch from the best SFT checkpoint, with the SFT model retained as the frozen KL reference policy. For each VQA prompt, the policy samples grouped candidate outputs consisting of a generated control instruction, a path sequence, a speed sequence, and an executability score. The reward combines route progress, collision avoidance, traffic-rule compliance, comfort, control-instruction-format validity, instruction-action consistency, and unsafe-acceptance penalty. The clipped policy objective uses $\varepsilon = 0.2$ and KL coefficient $\beta = 0.02$. The executability threshold is fixed to $\gamma = 0.5$ after validation and used unchanged in all closed-loop and counterfactual VQA evaluations.

4.2 Main Results

The Bench2Drive comparison is organized according to the sensing and planning formulation of the evaluated methods. The non-privileged baselines include conventional end-to-end planners, such as TCP/TCP-traj, UniAD, and VAD [2,3,25–27]; recent generation or policy-refinement planners, including DriveTransformer and DiffAD [28–33]; and vision-language-action planners, including Driveπ_0, DriveMoE, and SimLingo [8,10,11,34–36]. All non-privileged entries are evaluated under the 220-route Bench2Drive closed-loop protocol without map-level privileged planning input. Think2Drive and PDM-Lite are listed separately as privileged references because they use map-level or expert-planner information unavailable to sensor-only closed-loop planners [21,37].

The results in Table 3 provide benchmark-level context under the reported settings. The evaluated methods differ in sensor inputs, VLM/LLM backbones, action representations, data construction procedures, and training recipes. These results consequently do not support direct component-level attribution across methods. To make the comparison transparent, the table reports the backbone, input configuration, training/action supervision, and language supervision of each method.

Table 3: Main closed-loop and multi-ability comparison on Bench2Drive. Input, training/action supervision, and language supervision are clarified with compact tags. B2D denotes Bench2Drive, traj. denotes trajectory supervision, IL denotes imitation learning, RL denotes reinforcement learning, AD denotes Action Dreaming, and CF denotes counterfactual supervision. The MALT-Drive row reports the selected checkpoint used for the benchmark comparison; controlled ablations report mean ± std over three runs. *n*C/*n*L/*n*M count distinct camera views, LiDAR streams, and map inputs. Higher values indicate better performance. "-" denotes not reported, † denotes privileged reference results, and the best/second-best non-privileged results are marked in bold/underline. ↑ indicates that higher values are better. The compared methods differ in sensor configurations, backbone architectures, action representations, training data, and optimization strategies. The comparison thus provides benchmark-level evidence rather than a controlled component-level evaluation.

Method	LLM	Input	Train./Act. Sup.	Lang. Sup.	Closed-Loop		Ability (%) ↑					
					DS ↑	SR ↑	Merging	Overtaking	Emerg. Brake	Give Way	Traffic Sign	Mean
E2E planners												
TCP [25]	–	1C	B2D traj.; IL	–	40.70	15.00	16.18	20.00	20.00	10.00	6.99	14.63
TCP-traj [25]	–	1C	B2D traj.; IL	–	59.90	30.00	8.89	24.29	51.67	40.00	46.28	34.22
ThinkTwice [26]	–	1C	B2D traj.; IL	–	62.44	31.23	27.38	18.42	35.82	50.00	54.23	37.17
DriveAdapter [27]	–	1C + 1L	B2D traj.; IL	–	64.22	33.08	28.82	26.38	48.76	50.00	56.43	42.08
UniAD [2]	–	6C	B2D traj.; IL	–	45.81	16.36	14.10	17.78	21.67	10.00	14.21	15.55
VAD [3]	–	6C	B2D traj.; IL	–	42.35	15.00	8.11	24.44	18.64	20.00	19.15	18.07
DriveTransformer [28]	–	6C	B2D traj.; IL	–	63.46	35.01	17.57	35.00	48.36	40.00	52.10	38.60
DiffAD [29]	–	6C	B2D traj.; diffusion IL	–	67.92	38.64	30.00	35.55	46.66	40.00	46.32	38.79
DriveDPO [30]	–	6C	B2D traj.; DPO	–	62.02	30.62	–	–	–	–	–	–
RAP [31]	–	6C	B2D traj.; RL	–	66.42	37.27	–	–	–	–	–	–
Raw2Drive [32]	–	6C	B2D traj.; RL	–	71.36	50.24	43.35	51.11	60.00	50.00	62.26	53.34
TF++ [33]	–	3C + 1L	B2D traj.; IL	–	84.21	67.27	–	–	–	–	–	64.39
VLA planners												
Driveπ_0 [34]	PaliGemma-3B	1C	B2D traj.; action pred.	prompt	60.45	30.00	29.35	36.58	48.83	40.00	54.45	41.84
ReCogDrive [35]	InternVL3/Qwen2.5	6C	B2D/NAVSIM; SFT/RL	QA	71.36	45.45	29.73	20.00	69.09	20.00	71.34	42.03
DriveMoE [34]	PaliGemma-3B	2C	B2D traj.; MoE IL	prompt	74.22	48.64	34.67	40.00	65.45	40.00	59.44	47.91
ORION [11]	Vicuna-7B	6C	B2D + Chat-B2D; IL	VQA + plan tokens	77.74	54.62	25.00	71.11	78.33	30.00	69.15	54.72
MindDrive-IL [36]	Qwen2-0.5B	6C	B2D; IL	reasoning QA	75.85	49.30	30.26	66.67	60.00	33.33	56.91	49.44
MindDrive [36]	Qwen2-0.5B	6C	B2D; IL+RL	reasoning QA	78.04	55.09	32.89	75.56	68.33	50.00	57.89	56.94
AutoVLA [8]	Qwen2.5-VL-3B	1C	B2D; SFT + RFT	instr. + action tokens	78.84	57.73	–	–	–	–	–	–
SimLingo [10]	InternVL2-1B	1C	B2D; driving + AD	VQA + commentary	85.07	67.27	54.01	57.04	88.33	**53.33**	**82.45**	67.03
Privileged references												
Think2Drive† [37]	–	1M	B2D; privileged planning	–	91.85	85.41	–	–	–	–	–	86.26
PDM-Lite† [21]	–	1M	expert planner	–	97.02	92.27	–	–	–	–	–	92.82
MALT-Drive	InternVL2-1B	1C	B2D + CF; SFT + RFT	VQA + control library	**86.20**	**70.91**	**58.75**	60.00	**93.33**	50.00	82.11	**68.84**

Table 3 reports the closed-loop performance and fine-grained ability scores on Bench2Drive. Among the listed methods, SimLingo provides the closest reference because both methods use InternVL2-1B and one-camera input on Bench2Drive. Under their respective data construction and training pipelines, SimLingo reports DS/SR values of 85.07/67.27, while MALT-Drive records 86.20/70.91.

AutoVLA and ORION provide additional reference points for post-trained and planning-token-based VLA planners. AutoVLA uses a Qwen2.5-VL-3B backbone and physical action tokens, while ORION uses six-camera input, Vicuna-7B, and B2D+Chat-B2D supervision. Their results are included to show the performance range of current VLA driving methods under different configurations.

At the ability level, MALT-Drive records a mean score of 68.84%, with scores of 58.75% for Merging and 93.33% for Emergency Brake. The model records 50.00% for Give Way and 82.11% for Traffic Sign, where no clear advantage over SimLingo is observed. These results indicate that short-gap yielding and fine-grained traffic-sign grounding remain challenging under the current setting. In Give Way, the remaining errors are related to short-gap yielding: the generated control instruction can indicate a reasonable yielding intent, while the Modulation Branch may still produce an overly conservative speed profile or miss a short executable gap. For Traffic Sign, the failure diagnosis suggests that the remaining errors are mainly associated with partially occluded traffic signs and perception-route ambiguity, where a sign or priority cue may be assigned to an adjacent lane or side road.

Table 4 provides additional reference points on the NAVSIM benchmark. MALT-Drive records a PDMS of 91.7 with camera-only input. It also records an ego-progress score of 89.8 while maintaining competitive safety and comfort metrics. These results provide cross-benchmark evidence under the reported method-specific settings. Differences from other methods should not be interpreted as controlled component-level improvements.

Table 4: Comparison with state-of-the-art methods on the NAVSIM benchmark. C denotes camera input, and L denotes LiDAR input.

Method	Input	NC	DAC	EP	TTC	Comf.	PDMS
UniAD [2]	C	97.8	91.9	78.8	92.9	100.0	83.4
TransFuser [1]	C + L	97.7	92.8	79.2	92.8	100.0	84.0
DrivingGPT [38]	C	98.9	90.7	79.7	94.9	95.6	82.4
DriveX [39]	C	97.5	94.0	79.7	93.0	100.0	84.5
DiffusionDrive [40]	C + L	98.2	96.2	82.2	94.7	100.0	88.1
AutoVLA [8]	C	98.4	95.6	81.9	98.0	99.9	89.1
Uni-World VLA [41]	C	98.7	96.7	83.2	96.1	100.0	89.4
Hydra-MDP++ [42]	C + L	98.6	98.6	85.7	95.1	100.0	91.0
MALT-Drive	C	98.8	98.6	89.8	96.1	100.0	91.7

4.3 Ablation Studies

All controlled ablation and sensitivity experiments use the same backbone, sensor input, prediction horizon, and evaluation protocol. The main component, descriptor, auxiliary-objective, hyperparameter, and SFT-protocol results are reported as mean ± standard deviation over three runs. These repeated runs evaluate trend stability within the tested configurations and ranges.

Counterfactual Supervision, Branch Alignment, and RFT: Table 5 directly evaluates the three main training components. Removing both counterfactual supervision and branch-wise alignment reduces DS

and IAC-F1 to 83.06 ± 0.05 and 67.28 ± 0.10, while the full model reaches 86.17 ± 0.06 and 79.22 ± 0.07. With counterfactual supervision but without alignment, GPT and SPICE increase to 79.15 ± 0.10 and 74.08 ± 0.09, while IAC-F1 remains at 68.72 ± 0.11. This result shows that improved instruction generation alone does not ensure action consistency. The SFT-only full model reaches 78.51 ± 0.06 IAC-F1, and RFT further increases it to 79.22 ± 0.07. Table 6 also shows improvements from 69.2/68.5/66.9 to 80.4/79.6/78.8 for lane changing, stopping/waiting, and obstacle avoidance. These controlled trends support the role of counterfactual supervision in exposing instruction-dependent outcomes and the role of branch-wise alignment in connecting instruction semantics with path and speed/executability states.

Table 5: Ablation of counterfactual supervision, branch-wise alignment, and reinforcement fine-tuning on Bench2Drive. CF denotes counterfactual VQA supervision. A_{man} and A_{mod} denote the branch-wise language-planning alignment losses for the Maneuver and Modulation branches, respectively. The reported values are mean ± std over three runs. Higher values are better.

Variant	CF	A_{man}	A_{mod}	RFT	DS	SR	GPT	SPICE	IAC-F1
Baseline	×	×	×	✓	83.06 ± 0.05	63.18 ± 0.37	69.48 ± 0.09	65.18 ± 0.08	67.28 ± 0.10
w/o CF	×	✓	✓	✓	83.76 ± 0.06	65.61 ± 0.21	74.06 ± 0.11	68.42 ± 0.07	76.18 ± 0.09
w/o Align	✓	×	×	✓	84.10 ± 0.07	66.36 ± 0.37	79.15 ± 0.10	74.08 ± 0.09	68.72 ± 0.11
A_{man} only	✓	✓	×	✓	85.14 ± 0.05	68.64 ± 0.37	80.42 ± 0.08	75.26 ± 0.10	75.64 ± 0.08
A_{mod} only	✓	×	✓	✓	84.93 ± 0.08	68.03 ± 0.21	80.21 ± 0.12	75.11 ± 0.09	76.43 ± 0.07
Full w/o RFT	✓	✓	✓	×	85.72 ± 0.05	69.55 ± 0.37	81.21 ± 0.07	76.29 ± 0.06	78.51 ± 0.06
Full	✓	✓	✓	✓	86.17 ± 0.06	70.91 ± 0.37	81.58 ± 0.09	76.63 ± 0.08	79.22 ± 0.07

Table 6: Case-level instruction-action consistency analysis on representative driving instructions. LC, SW, and OA denote lane changing, stopping/waiting, and obstacle avoidance, respectively. Higher values are better.

Variant	Case-Level IAC-F1 (%)		
	LC-F1	SW-F1	OA-F1
w/o Align	69.2	68.5	66.9
Full	80.4	79.6	78.8

Moment Descriptor: Table 7 shows that the combined $\mu + \rho$ descriptor reaches 86.17 ± 0.06 DS and 79.22 ± 0.07 IAC-F1, compared with 85.36 ± 0.06/76.88 ± 0.09 for μ alone and 85.03 ± 0.07/77.62 ± 0.08 for ρ alone. The results support the complementary contributions of first-order control summaries and second-order token-relation information. In Table 8, IAC-F1 remains between 78.2 and 79.2 for $G \in \{2, 4, 8, 16, 32\}$. The selected $G = 8$ setting gives the highest DS and IAC-F1 with a 3.6% latency increase over mean pooling. This result supports local stability within the evaluated grouping range.

Table 7: Ablation of the moment descriptor. μ denotes the first-order pooled statistic, and ρ denotes the grouped second-order relation sketch. The reported values are mean ± std over three runs. Higher values are better.

Variant	Descriptor	DS	SR	IAC-F1
μ only	μ	85.36 ± 0.06	69.09 ± 0.37	76.88 ± 0.09
ρ only	ρ	85.03 ± 0.07	68.79 ± 0.21	77.62 ± 0.08
Full	$\mu + \rho$	86.17 ± 0.06	70.91 ± 0.37	79.22 ± 0.07

Table 8: Sensitivity of the moment descriptor to channel grouping on Bench2Drive. The projected branch dimension is fixed to $d = 256$, and latency is measured under the same A100 inference setting with batch size 1. Bold indicates the selected setting and its best results; ↑/↓ indicate that higher/lower values are better, respectively.

Descriptor	G	d_g	Latency (s) ↓	ΔLatency	DS ↑	SR (%) ↑	IAC-F1 ↑
Mean pooling	–	–	0.948	0.0%	84.96	68.64	74.8
μ only	–	–	0.953	+0.5%	85.34	69.09	76.9
$\mu + \rho$	2	128	1.028	+8.4%	85.78	70.00	78.4
$\mu + \rho$	4	64	1.006	+6.1%	85.97	70.45	78.9
$\mu + \rho$	**8**	**32**	0.982	+3.6%	**86.20**	**70.91**	**79.2**
$\mu + \rho$	16	16	0.976	+3.0%	85.88	70.45	78.8
$\mu + \rho$	32	8	0.970	+2.3%	85.61	70.00	78.2

Auxiliary Objectives: Table 9 evaluates the two auxiliary objectives after fixing the full descriptor and branch-wise alignment. Removing $\mathcal{L}_{\text{exe}}$ reduces DS/SR to 84.72/68.48 and IAC-F1 to 75.98, corresponding to drops of 1.45 DS, 2.43 SR, and 3.24 IAC-F1 from the full model. This shows that explicit executability supervision is important for matching non-executable instructions with conservative fallback behavior. Removing $\mathcal{L}_{\text{sep}}$ produces a smaller but consistent drop, with DS/SR of 85.39/69.55 and IAC-F1 of 77.86, supporting its role as a regularizer that prevents maneuver and modulation representations from collapsing into a single shared channel.

Table 9: Ablation of auxiliary objectives. $\mathcal{L}_{\text{exe}}$ supervises instruction executability, and $\mathcal{L}_{\text{sep}}$ regularizes the separation between maneuver-related and modulation-related latent channels. The reported values are mean ± std over three runs. Higher values are better.

Variant	$\mathcal{L}_{\text{exe}}$	$\mathcal{L}_{\text{sep}}$	DS	SR	IAC-F1
w/o $\mathcal{L}_{\text{exe}}$	×	✓	84.72 ± 0.07	68.48 ± 0.21	75.98 ± 0.10
w/o $\mathcal{L}_{\text{sep}}$	✓	×	85.39 ± 0.06	69.55 ± 0.37	77.86 ± 0.08
Full	✓	✓	86.17 ± 0.06	70.91 ± 0.37	79.22 ± 0.07

Hyperparameter Sensitivity: Table 10 shows that moderate changes around the default configuration preserve similar performance. Across the tested nonzero alignment weights, grouping settings, descriptor dimensions, thresholds, and nonzero counterfactual ratios, DS remains between 85.24 and 85.97, and IAC-F1 remains between 77.16 and 78.93. Setting $\lambda_a = 0$ causes the largest IAC-F1 reduction, directly supporting the language-planning alignment objective. Increasing the executability threshold reduces unsafe acceptance while lowering SR, and increasing the counterfactual ratio produces a similar safety-progress trade-off. These results support local stability around the selected operating point within the evaluated ranges.

SFT Protocol Sensitivity: Table 11 keeps the architecture, supervised losses, evaluation setting, and 12-epoch budget fixed. Removing the late non-executable phase increases unsafe acceptance from 4.1 ± 0.2% to 6.1 ± 0.3%. Increasing the non-executable ratio or extending this phase further reduces unsafe acceptance, with small reductions in DS or SR. Moving the non-executable phase before balanced counterfactual training also lowers IAC-F1. These controlled results support the selected 3/7/2 schedule with a 40% non-executable ratio as a balance between driving performance and unsafe-instruction rejection.

Table 10: Hyperparameter sensitivity analysis on Bench2Drive. One factor is varied each time, and all other settings follow the default configuration: $\lambda_a = 0.2$, $\lambda_s = 0.05$, $G = 8$, $d_z = 256$, $\gamma = 0.5$, and $r_{\mathrm{CF}} = 50\%$. d_z denotes the compact moment-descriptor dimension, and r_{CF} denotes the ratio of counterfactual samples in the SFT stage. Values are mean ± std over three runs. All entries are simulation results. ↑/↓ indicate that higher/lower values are better, respectively.

Changed Factor	Setting	DS ↑	SR (%) ↑	IAC-F1 ↑	Unsafe Accept. (%) ↓
Default	–	86.17 ± 0.06	70.91 ± 0.37	79.22 ± 0.07	3.7 ± 0.1
λ_a	0.00	84.52 ± 0.07	67.73 ± 0.37	69.38 ± 0.11	5.0 ± 0.2
λ_a	0.10	85.63 ± 0.06	70.00 ± 0.37	78.42 ± 0.08	4.1 ± 0.2
λ_a	0.40	85.82 ± 0.07	70.45 ± 0.37	78.76 ± 0.09	3.8 ± 0.1
λ_s	0.00	85.39 ± 0.06	69.55 ± 0.37	77.86 ± 0.08	4.4 ± 0.2
λ_s	0.20	85.24 ± 0.08	68.64 ± 0.37	77.16 ± 0.10	4.1 ± 0.1
G	4	85.97 ± 0.07	70.45 ± 0.37	78.90 ± 0.08	3.9 ± 0.1
G	16	85.88 ± 0.07	70.45 ± 0.37	78.80 ± 0.09	3.8 ± 0.1
d_z	128	85.61 ± 0.06	70.00 ± 0.37	78.31 ± 0.09	4.0 ± 0.2
d_z	512	85.93 ± 0.07	70.45 ± 0.37	78.74 ± 0.08	3.8 ± 0.1
γ	0.40	85.74 ± 0.06	69.55 ± 0.37	78.80 ± 0.08	4.9 ± 0.2
γ	0.60	85.49 ± 0.07	69.09 ± 0.37	78.50 ± 0.09	2.6 ± 0.1
r_{CF}	0%	84.21 ± 0.07	66.82 ± 0.37	76.84 ± 0.10	6.4 ± 0.3
r_{CF}	25%	85.24 ± 0.07	68.64 ± 0.37	77.72 ± 0.09	5.2 ± 0.2
r_{CF}	75%	85.79 ± 0.06	70.00 ± 0.37	78.93 ± 0.08	3.1 ± 0.1

Table 11: Sensitivity of the SFT training protocol on Bench2Drive. N denotes nominal expert training, CF denotes balanced nominal-counterfactual training, and NE denotes non-executable-upweighted training. All variants use the same architecture, losses, and 12-epoch SFT budget, and are evaluated before RFT. Values are mean ± std over three runs. ↑/↓ indicate that higher/lower values are better, respectively.

Protocol	Order	Epochs	NE Ratio	DS ↑	SR (%) ↑	IAC-F1 ↑	Unsafe Accept. (%) ↓
NE ratio 20%	N→CF→NE	3/7/2	20%	85.28 ± 0.07	68.64 ± 0.37	77.74 ± 0.08	5.6 ± 0.2
Default	N→CF→NE	3/7/2	40%	85.72 ± 0.05	69.55 ± 0.37	78.51 ± 0.06	4.1 ± 0.2
NE ratio 50%	N→CF→NE	3/7/2	50%	85.48 ± 0.06	69.24 ± 0.21	78.31 ± 0.07	3.5 ± 0.1
No NE phase	N→CF	3/9/0	–	85.13 ± 0.06	68.48 ± 0.21	77.18 ± 0.08	6.1 ± 0.3
NE-heavy	N→CF→NE	3/5/4	40%	85.41 ± 0.06	69.09 ± 0.37	78.36 ± 0.07	3.6 ± 0.1
Early NE phase	N→NE→CF	3/2/7	40%	85.17 ± 0.07	68.64 ± 0.37	77.56 ± 0.09	5.1 ± 0.2

Executability-Threshold Analysis: Finally, Table 12 reports the validation-set calibration used to select the executability threshold γ, which is then fixed for all reported experiments. Executable instructions are treated as the positive class: precision measures the reliability of accepted instructions, recall measures the retention of executable instructions, and unsafe-acceptance rate measures how often unsafe instructions are incorrectly accepted. Lower thresholds retain more executable instructions but accept more unsafe ones, whereas higher thresholds reduce unsafe acceptance at the cost of recall. The selected $\gamma = 0.50$ gives a balanced operating point, with 86.8% precision, 86.1% recall, and a 3.7% unsafe-acceptance rate.

Table 12: Executability-threshold analysis on validation counterfactual instructions. Higher precision and recall are better, while a lower unsafe-acceptance rate is better. Bold indicates the selected threshold; ↑/↓ indicate that higher/lower values are better, respectively.

γ	Precision (%) ↑	Recall (%) ↑	Unsafe Accept. (%) ↓
0.30	79.4	92.1	6.8
0.40	83.6	89.0	4.9
0.50	86.8	86.1	3.7
0.60	89.5	80.3	2.6
0.70	91.2	72.4	2.1

4.4 Analysis and Discussion

Instruction-Conditioned Decoding. Table 13 examines the effect of control-instruction token sources on trajectory decoding. The reference-control-instruction setting feeds the decoder with the ground-truth reference control instruction $c_t^\star$ and acts as an upper-bound diagnostic for control-instruction token quality. The generated-control-instruction setting corresponds to normal inference. On all routes, generated control instructions reduce DS from 87.05 to 86.20 and IAC-F1 from 81.5 to 79.2, giving a gap of 0.85 DS and 2.3 IAC-F1. This confirms that the teacher-forcing mismatch introduces a measurable degradation.

Table 13: Instruction-conditioned decoding analysis under different scene splits. The trained decoder is kept fixed, while the control-instruction token source is replaced at inference. The complex subset contains routes involving interaction constraints, traffic-rule constraints, obstacle bypassing, or short-horizon conflicts. ↑ indicates that higher values are better.

Scene Split	Control-Instruction Source	DS ↑	SR (%)↑	IAC-F1 ↑
All routes	Reference control instruction	87.05	72.27	81.5
All routes	Generated control instruction	86.20	70.91	79.2
Complex subset	Reference control instruction	82.64	63.18	76.4
Complex subset	Generated control instruction	81.31	61.36	72.9
All routes	No control-instruction tokens	83.08	63.18	67.3
All routes	Mismatched control instruction	74.32	51.36	44.9

The complex subset is further evaluated. The gap between reference and generated control instructions increases to 1.33 DS and 3.5 IAC-F1, indicating that the gap becomes larger when generated control instructions need to encode finer interaction, traffic-rule, and conflict-avoidance semantics. Removing control-instruction tokens causes a larger drop, and randomly mismatching control-instruction tokens reduces IAC-F1 to 44.9. These results show that generated control-instruction states remain useful for planning, while exposure bias remains a limitation under complex or shifted traffic distributions.

Controlled-Space Coverage. The generated-control-instruction coverage of the controlled instruction space is further quantified on the validation counterfactual set. A generated control instruction is counted as covered if it exactly matches or can be normalized to one of the 30 canonical control-instruction classes. Otherwise, it is counted as unsupported or ambiguous and is handled by the fallback rule. On all validation routes, 3.8% of generated control instructions are unsupported or ambiguous. On the complex subset, this rate increases to 6.4%. This trend is consistent with Table 13, where generated control instructions show a

larger gap on complex scenes. The fallback behavior is also related to the executability-threshold analysis in Table 12, where the selected threshold gives 86.8% precision, 86.1% recall, and a 3.7% unsafe-acceptance rate. This diagnostic is limited to the current controlled setting and does not claim robustness to unrestricted open-ended language.

Sensitivity of IAC-F1. The sensitivity of IAC-F1 to the thresholds used for action-label assignment is further evaluated. Table 14 varies one threshold at a time while keeping the others at their default values. The results show that IAC-F1 remains stable under moderate threshold changes. For example, when the lateral-displacement threshold varies from 0.4 to 0.8 m, IAC-F1 changes only from 79.0 to 78.6. When the speed-change threshold varies from 0.3 to 0.7 m/s, IAC-F1 remains within 78.8–79.2. Similar stability is observed for the stop-speed and executability thresholds. These results indicate that the main conclusion of instruction-action consistency is not sensitive to a specific threshold choice.

Table 14: Sensitivity analysis of IAC-F1 with respect to action-label assignment thresholds. One threshold is varied at a time, while the others are kept at their default values.

Threshold	Low Setting		Default Setting		High Setting	
	Value	IAC-F1	Value	IAC-F1	Value	IAC-F1
Lateral displacement δ_y	0.4 m	79.0	0.6 m	79.2	0.8 m	78.6
Speed change δ_u	0.3 m/s	78.8	0.5 m/s	79.2	0.7 m/s	78.9
Stop speed v_{stop}	0.2 m/s	79.1	0.3 m/s	79.2	0.4 m/s	78.7
Executability γ	0.4	78.8	0.5	79.2	0.6	78.5

Failure-Source Diagnosis. Failures are further diagnosed according to where the error appears in the language-to-trajectory pipeline. A language-generation error means that the generated control instruction gives an incorrect driving intent, such as describing a partially occluded traffic-sign scene as a keep-speed case. A trajectory-planning error means that the generated control instruction is reasonable, but the decoded path and speed profile do not realize it. Table 15 shows that most remaining failures are planning-side errors. For example, in Give Way cases, the model often generates a correct yielding instruction, but the Modulation Branch may miss a short executable gap.

Table 15: Failure-source diagnosis on Bench2Drive evaluation routes.

Error Type	Share (%)	Typical Case
Language-generation error	13.8	a partially occluded traffic-sign scene is described as keep-speed
Maneuver-decoding error	24.6	a correct bypass instruction is generated, but the path has insufficient lateral return
Modulation/executability error	31.9	a correct yield instruction is generated, but the speed profile is overly conservative or misses a short gap
Perception-route ambiguity	21.7	a sign or priority cue is associated with an adjacent lane or side road
Fallback-selection error	8.0	an unsafe instruction is rejected, but the fallback progress is too low

Qualitative Visualization. Fig. 3 shows representative instruction-conditioned planning cases. In the right-turn, red-light, and car-following scenes, the generated control instructions closely match the reference control intent while the decoded trajectories follow the corresponding maneuver and speed requirements.

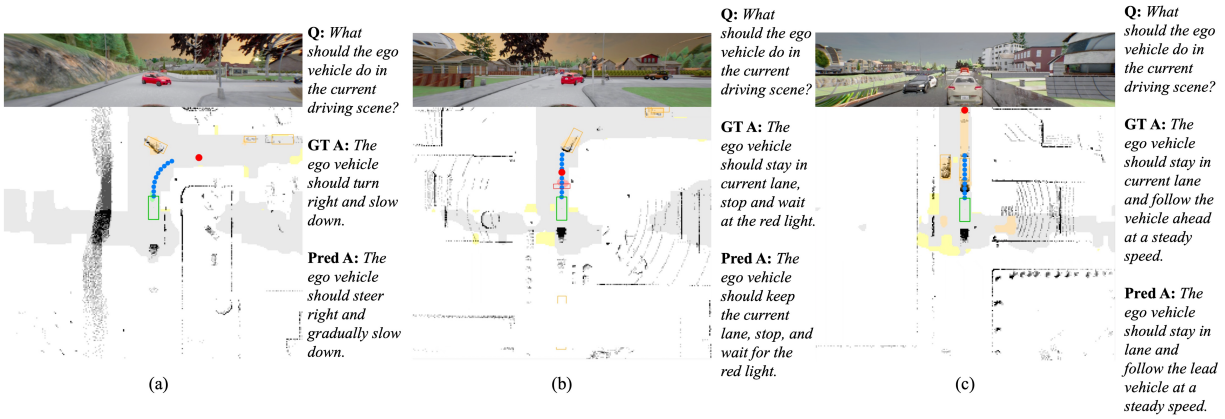

Figure 3: Qualitative visualization of MALT-Drive on Bench2Drive. (**a**) Right turn with gradual deceleration. (**b**) Lane keeping and stopping at a red light. (**c**) Lane keeping and steady-speed car following. In each panel, the generated control instruction is compared with the canonical control instruction and the decoded trajectory.

Fig. 4 adds two diagnostic cases for instruction rejection and safety fallback. In the green-light intersection case, the generated control instruction asks the ego vehicle to accelerate and turn left, while straight-through traffic occupies the conflict area. The decoded trajectory waits near the current lane, showing that the modulation and executability states route the output to a conservative fallback. In the stationary-front-vehicle case, the generated control instruction asks the ego vehicle to overtake, while a left-front oncoming vehicle makes the lateral maneuver unsafe. The decoded trajectory remains in lane and rejects the non-executable instruction. These cases make the counterfactual control-instruction supervision visible at inference time and complement the executability-threshold analysis in Table 12.

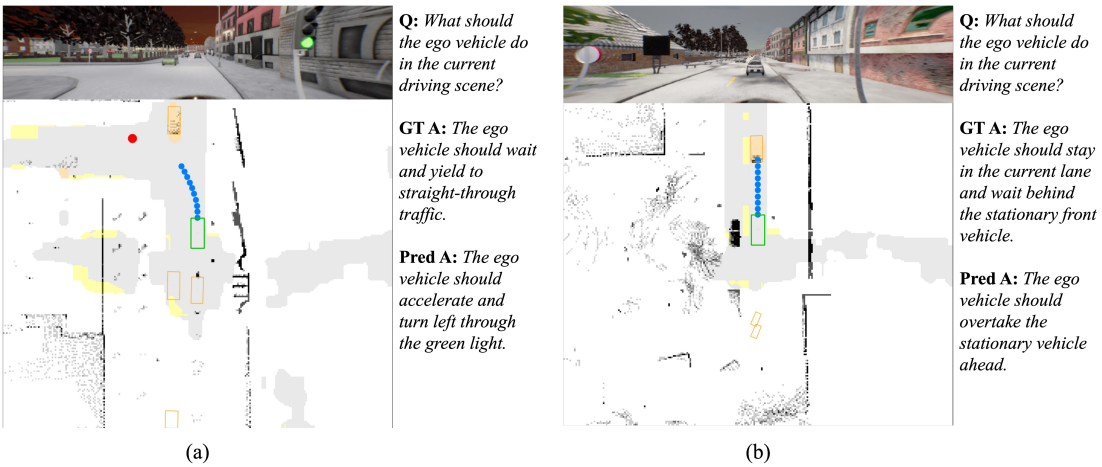

Figure 4: Diagnostic visualization of counterfactual and safety-fallback cases. (**a**) Rejected acceleration-and-left-turn instruction: straight-through traffic occupies the conflict area, so the fallback trajectory waits in the current lane. (**b**) Rejected overtaking instruction: an oncoming vehicle blocks the lateral maneuver, so the fallback trajectory remains behind the stationary vehicle.

Runtime Reference. Table 16 reports closed-loop performance with source-reported latency for recent VLA-based driving methods. The latency values are collected from different papers and hardware settings, so they are used only as runtime context. The latency of MALT-Drive is measured on a single NVIDIA A100 80GB GPU with batch size 1, using the same input resolution, four-frame setting, VQA generation procedure, and 8-step path-speed prediction horizon as in the Bench2Drive evaluation. This measurement reports the practical runtime of the MALT-Drive implementation under a fixed setting.

Table 16: Performance and source-reported runtime reference on Bench2Drive. DS and SR denote driving score and success rate. Latency values are reported from the corresponding papers or measured for MALT-Drive, and are not used for strict cross-method runtime comparison due to different hardware and implementation settings. ↑/↓ indicate that higher/lower values are better, respectively.

Method	VLM Backbone	DS↑	SR (%)↑	Hardware	Latency (s)↓
ORION [11]	Qwen2-0.5B	66.10	36.57	NVIDIA A100	0.655
	LLaMA3-1.1B	72.23	48.33	NVIDIA A100	0.685
	Vicuna-7B (7B)	77.74	54.62	NVIDIA A100	1.106
AutoVLA-Fast [8]	Qwen2.5-VL-3B	78.84	57.73	–	1.072
AutoVLA-Slow [8]	Qwen2.5-VL-3B	78.84	57.73	–	10.518
SimLingo [10]	InternVL2-1B	85.07	67.27	NVIDIA A100 40 GB	–
MindDrive [36]	Qwen2-0.5B	78.04	55.09	32 × NVIDIA A800 80 GB	–
DriveMoE [34]	PaliGemma-3B	74.22	48.64	8 × NVIDIA H200	–
MALT-Drive	InternVL2-1B	86.20	70.91	NVIDIA A100 80 GB	0.982

Mixed-Instruction Diagnosis. Because the controlled instruction library includes coupled lateral-longitudinal instructions, mixed maneuver-modulation cases are further diagnosed. The mixed subset contains instructions whose targets jointly affect path geometry and speed or yielding behavior. Table 17 shows that mixed instructions benefit from weak branch specialization. Removing $\mathcal{L}_{\text{sep}}$ weakens both maneuver and modulation F1, while an overly strong separation weight reduces mixed-instruction consistency. The default weak setting gives the best mixed IAC-F1, indicating that the regularizer supports channel specialization without harming coupled lateral-longitudinal instructions.

Table 17: Analysis of mixed maneuver-modulation instructions. Higher F1 scores are better, and lower unsafe-acceptance rate is better. ↑/↓ indicate that higher/lower values are better, respectively.

Variant	Mixed IAC-F1 ↑	Maneuver F1 ↑	Modulation F1 ↑	Unsafe Accept. (%) ↓
w/o $\mathcal{L}_{\text{sep}}$	74.9	78.3	76.2	4.5
Strong separation $\lambda_s = 0.20$	73.6	77.8	74.1	4.1
Default weak separation $\lambda_s = 0.05$	78.4	81.7	80.3	3.7

5 Conclusion

This paper has presented MALT-Drive, a VQA-supervised language-to-trajectory planning framework for autonomous driving. The core idea is to treat the generated control instruction as an intermediate control variable and explicitly align its token states with the latent planning states that produce motion. By separating maneuver-related and modulation-related planning factors, the framework provides a structured way to connect the generated control instruction with path geometry, speed regulation, and executability estimation.

Several limitations remain. The current formulation relies on a controlled instruction space and is evaluated mainly in a simulator-based closed-loop benchmark. Unsupported or ambiguous generated control instructions are routed to a conservative fallback behavior, but this mechanism only reduces the risk of directly executing out-of-library instructions and does not establish robustness to unrestricted language. In addition, part of the counterfactual supervision is instantiated from SimLingo Dreamer data, which introduces a data-construction dependency when comparing with SimLingo on Bench2Drive. Repeated-run results cover controlled ablation and sensitivity settings but do not establish formal statistical significance or robustness under unrestricted hyperparameter, language, or domain shifts. Robustness to open-ended language, richer traffic interactions, independent counterfactual sources, route-disjoint evaluation, and real-world sensor noise requires further study.

Acknowledgement: Not applicable.

Funding Statement: This work has been supported by the National Natural Science Foundation of China (Grants 52225212 and 52472433), the Jiangsu Province Frontier Technology Research and Development Plan (BF2025082).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Ziheng Lu, Yingfeng Cai, Hai Wang and Long Chen; data collection and experiment implementation: Ziheng Lu and Wei Dong; analysis and interpretation of results: Ziheng Lu, Wei Dong and Yingfeng Cai; draft manuscript preparation: Ziheng Lu; supervision and manuscript revision: Yingfeng Cai, Hai Wang and Long Chen. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The Bench2Drive and NAVSIM benchmark data used in this study are publicly available from their official repositories.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chitta K, Prakash A, Jaeger B, Yu Z, Renz K, Geiger A. TransFuser: imitation with transformer-based sensor fusion for autonomous driving. IEEE Trans Pattern Anal Mach Intell. 2023;45(11):12878–95. doi:10.1109/tpami.2022.3200245.
2. Hu Y, Yang J, Chen L, Li K, Sima C, Zhu X, et al. Planning-oriented autonomous driving. In: Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2023 Jun 17–24; Vancouver, BC, Canada. p. 17853–62. doi:10.1109/cvpr52729.2023.01712.
3. Jiang B, Chen S, Xu Q, Liao B, Chen J, Zhou H, et al. VAD: vectorized scene representation for efficient autonomous driving. In: Proceedings of the 2023 IEEE/CVF International Conference on Computer Vision (ICCV); 2023 Oct 1–6; Paris, France. p. 8306–16. doi:10.1109/iccv51070.2023.00766.
4. Zheng W, Song R, Guo X, Zhang C, Chen L. GenAD: generative end-to-end autonomous driving. In: Computer vision—ECCV 2024. Cham, Switzerland: Springer Nature; 2025. p. 87–104. doi:10.1007/978-3-031-73650-6_6.
5. Sima C, Renz K, Chitta K, Chen L, Zhang H, Xie C, et al. DriveLM: driving with graph visual question answering. In: Computer vision—ECCV 2024. Cham, Switzerland: Springer Nature; 2025. p. 256–74. doi:10.1007/978-3-031-72943-0_15.
6. Shao H, Hu Y, Wang L, Song G, Waslander SL, Liu Y, et al. LMDrive: closed-loop end-to-end driving with large language models. In: Proceedings of the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2024 Jun 16–22; Seattle, WA, USA. p. 15120–30. doi:10.1109/cvpr52733.2024.01432.
7. Hwang JJ, Xu R, Lin H, Hung WC, Ji J, Choi K, et al. EMMA: end-to-end multimodal model for autonomous driving. Transactions on machine learning research. 2025 [cited 2026 Jan 1]. Available from: https://openreview.net/forum?id=kH3t5lmOU8.

8. Zhou Z, Cai T, Zhao S, Zhang Y, Huang Z, Zhou B, et al. AutoVLA: a vision-language-action model for end-to-end autonomous driving with adaptive reasoning and reinforcement fine-tuning. Adv Neural Inf Process Syst. 2026;38:27920–56.
9. Cui E, Wang W, Li Z, Xie J, Zou H, Deng H, et al. DriveMLM: aligning multi-modal large language models with behavioral planning states for autonomous driving. Visual Intell. 2025;3(1):22. doi:10.1007/s44267-025-00095-w.
10. Renz K, Chen L, Arani E, Sinavski O. SimLingo: vision-only closed-loop autonomous driving with language-action alignment. In: Proceedings of the 2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2025 Jun 10–17. Nashville, TN, USA. p. 11993–2003. doi:10.1109/cvpr52734.2025.01120.
11. Fu H, Zhang D, Zhao Z, Cui J, Liang D, Zhang C, et al. Orion: a holistic end-to-end autonomous driving framework by vision-language instructed action generation. In: Proceedings of the 2025 IEEE/CVF International Conference on Computer Vision (ICCV); 2025 Oct 19–25. Honolulu, HI, USA. p. 24823–34. doi:10.1109/iccv51701.2025.02302.
12. Gao T, Tan C, Glossop C, Gao T, Sun J, Stachowicz K, et al. SteerVLA: steering vision-language-action models in long-tail driving scenarios. arXiv:2602.08440. 2026.
13. Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, et al. Learning transferable visual models from natural language supervision. In: Proceedings of the 38th International Conference on Machine Learning. Vol. 139 of Proceedings of Machine Learning Research; 2021 Jul 18–24; Virtual. p. 8748–63.
14. Gao Z, Jiang X, Xu X, Shen F, Li Y, Shen HT. Embracing unimodal aleatoric uncertainty for robust multimodal fusion. In: Proceedings of the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2024 Jun 16–22; Seattle, WA, USA. p. 26866–75. doi:10.1109/cvpr52733.2024.02538.
15. Chun S, Kim W, Park S, Yun S. Probabilistic language-image pre-training. In: Proceedings of the International Conference on Learning Representations; 2025 Apr 24–28; Singapore.
16. Xie J, Long F, Lv J, Wang Q, Li P. Joint distribution matters: deep brownian distance covariance for few-shot classification. In: Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2022 Jun 18–24; New Orleans, LA, USA. p. 7962–71. doi:10.1109/cvpr52688.2022.00781.
17. Gao M, Wang Q, Lin Z, Zhu P, Hu Q, Zhou J. Tuning pre-trained model via moment probing. In: Proceedings of the 2023 IEEE/CVF International Conference on Computer Vision (ICCV); 2023 Oct 1–6; Paris, France. p. 11769–79. doi:10.1109/iccv51070.2023.01084.
18. Shao Z, Wang P, Zhu Q, Xu R, Song J, Bi X, et al. DeepSeekMath: pushing the limits of mathematical reasoning in open language models. arXiv:2402.03300. 2024.
19. Jia X, Yang Z, Li Q, Zhang Z, Yan J. Bench2Drive: towards multi-ability benchmarking of closed-loop end-to-end autonomous driving. Adv Neural Inf Process Syst. 2024;37:819–44. doi:10.52202/079017-0025.
20. Dauner D, Hallgarten M, Li T, Weng X, Huang Z, Yang Z, et al. NAVSIM: data-driven non-reactive autonomous vehicle simulation and benchmarking. Adv Neural Inf Process Syst. 2024;37:28706–19. doi:10.52202/079017-0902.
21. Beißwenger J. PDM-Lite: a rule-based planner for CARLA Leaderboard 2.0. Tübingen, Germany: University of Tübingen; 2024.
22. Anderson P, Fernando B, Johnson M, Gould S. SPICE: semantic propositional image caption evaluation. In: Computer vision—ECCV 2016. Cham, Switzerland: Springer; 2016. p. 382–98. doi:10.1007/978-3-319-46454-1_24.
23. Jiang B, Chen S, Liao B, Zhang X, Yin W, Zhang Q, et al. Senna: bridging large vision-language models and end-to-end autonomous driving. arXiv:2410.22313. 2024.
24. Gao Z, Chen Z, Cui E, Ren Y, Wang W, Zhu J, et al. Mini-InternVL: a flexible-transfer pocket multi-modal model with 5% parameters and 90% performance. Visual Intell. 2024;2(1):32. doi:10.1007/s44267-024-00067-6.
25. Wu P, Jia X, Chen L, Yan J, Li H, Qiao Y. Trajectory-guided control prediction for end-to-end autonomous driving: a simple yet strong baseline. Adv Neural Inf Process Syst. 2022;35:6119–32. doi:10.52202/068431-0443.
26. Jia X, Wu P, Chen L, Xie J, He C, Yan J, et al. Think twice before driving: towards scalable decoders for end-to-end autonomous driving. In: Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2023 Jun 17–24; Vancouver, BC, Canada. p. 21983–94. doi:10.1109/cvpr52729.2023.02105.
27. Jia X, Gao Y, Chen L, Yan J, Liu PL, Li H. DriveAdapter: breaking the coupling barrier of perception and planning in end-to-end autonomous driving. In: Proceedings of the 2023 IEEE/CVF International Conference on Computer Vision (ICCV); 2023 Oct 1–6; Paris, France. p. 7919–29. doi:10.1109/iccv51070.2023.00731.

28. Jia X, You J, Zhang Z, Yan J. DriveTransformer: unified transformer for scalable end-to-end autonomous driving. In: Proceedings of the International Conference on Learning Representations; 2025 Apr 24–28. Singapore.
29. Wang T, Zhang C, Qu X, Li K, Liu W, Huang C. DiffAD: a unified diffusion modeling approach for autonomous driving. arXiv:2503.12170. 2025.
30. Shang S, Chen Y, Wang Y, Li Y, Zhang Z. DriveDPO: policy learning via safety DPO for end-to-end autonomous driving. Adv Neural Inf Process Syst. 2026;38:81565–85.
31. Feng L, Gao Y, Zablocki E, Li Q, Li W, Liu S, et al. RAP: 3D rasterization augmented end-to-end planning. In: Proceedings of the International Conference on Learning Representations; 2026 Apr 23–27. Rio de Janeiro, Brazil.
32. Yang Z, Jia X, Li Q, Yang X, Yao M, Yan J. Raw2Drive: reinforcement learning with aligned world models for end-to-end autonomous driving (in CARLA v2). Adv Neural Inf Process Syst. 2026;38:134122–47.
33. Jaeger B, Chitta K, Geiger A. Hidden biases of end-to-end driving models. In: Proceedings of the 2023 IEEE/CVF International Conference on Computer Vision (ICCV); 2023 Oct 1–6; Paris, France. p. 8206–15. doi:10.1109/iccv51070.2023.00757.
34. Yang Z, Chai Y, Jia X, Li Q, Shao Y, Zhu X, et al. DriveMoE: mixture-of-experts for vision-language-action model in end-to-end autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2026 Jun 3–7; Denver, CO, USA. p. 10678–88.
35. Li Y, Xiong K, Guo X, Li F, Yan S, Xu G, et al. ReCogDrive: a reinforced cognitive framework for end-to-end autonomous driving. In: Proceedings of the International Conference on Learning Representations; 2026 Apr 23–27. Rio de Janeiro, Brazil.
36. Fu H, Zhang D, Zhao Z, Cui J, Xie H, Wang B, et al. MindDrive: a vision-language-action model for autonomous driving via online reinforcement learning. arXiv:2512.13636. 2025.
37. Li Q, Jia X, Wang S, Yan J. Think2Drive: efficient reinforcement learning by thinking with latent world model for autonomous driving (in CARLA-V2). In: Computer vision—ECCV 2024. Cham, Switzerland: Springer Nature; 2025. p. 142–58. doi:10.1007/978-3-031-72995-9_9.
38. Chen Y, Wang Y, Zhang Z. DrivingGPT: unifying driving world modeling and planning with multi-modal autoregressive transformers. In: Proceedings of the 2025 IEEE/CVF International Conference on Computer Vision (ICCV); 2025 Oct 19–25. Honolulu, HI, USA. p. 26890–900. doi:10.1109/iccv51701.2025.02496.
39. Shi C, Shi S, Sheng K, Zhang B, Jiang L. DriveX: omni scene modeling for learning generalizable world knowledge in autonomous driving. In: Proceedings of the 2025 IEEE/CVF International Conference on Computer Vision (ICCV); 2025 Oct 19–25; Honolulu, HI, USA. p. 28599–609. doi:10.1109/iccv51701.2025.02656.
40. Liao B, Chen S, Yin H, Jiang B, Wang C, Yan S, et al. DiffusionDrive: truncated diffusion model for end-to-end autonomous driving. In: Proceedings of the 2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2025 Jun 10–17; Nashville, TN, USA. p. 12037–47. doi:10.1109/cvpr52734.2025.01124.
41. Liu Q, Xu H, Li J, Sun B, Hao Z, She D, et al. Uni-World VLA: interleaved world modeling and planning for autonomous driving. arXiv:2603.27287. 2026.
42. Li K, Li Z, Lan S, Xie Y, Zhang Z, Liu J, et al. Hydra-MDP++: advancing end-to-end driving via expert-guided hydra-distillation. arXiv:2503.12820. 2025.