ASSCG: Just-Right Gating over Chattering for Fast–Slow LLM Planning in Autonomous Driving

Sining Ang[1,2], Yuan Chen[1,3], Liu Haiyan[4], Xuanyao Mao[4], Jason Bao[4], Xuliang[4], Bingchuan Sun[4*], and Yan Wang[1*]

[1] Institute for AI Industry Research (AIR), Tsinghua University
angsn@mail.ustc.edu.cn, wangyan@air.tsinghua.edu.cn
[2] Department of Automation, University of Science and Technology of China
[3] Beihang University
chenyuan1@buaa.edu.cn
[4] Lenovo Group Limited
{liuhy46,maoxy6,jbao,xuliang18,sunbc1}@lenovo.com

Abstract. Large language models (LLMs) can improve autonomous driving planning but are costly to query online, and existing fast–slow planners often rely on hand-designed triggering rules that either over-call the slow system or call it at the wrong times. We formulate slow-system invocation as a resource-aware sequential decision problem and propose the Adaptive Slow-System Control Gate (ASSCG), which makes frame-level Query/Cache/Drop decisions to refresh, reuse, or suppress slow guidance. ASSCG uses an RWKV backbone for efficient long-horizon gating and is trained with supervised fine-tuning followed by GRPO-style compute-aware reinforcement fine-tuning. We apply ASSCG to two different fast–slow architectures: (i) AsyncDriver on nuPlan Hard20 closed-loop evaluation, where ASSCG improves score to 67.28 (+2.28) while reducing average end-to-end inference latency by ∼60%; and (ii) a RecogDrive-based dual system that we build by replacing its original VLM-2B module with a lightweight ViT-based fast planner and adding an LLM slow planner, evaluated on NAVSIM, where ASSCG achieves 91.4 PDMS (+0.6) and increases average speed by ∼25%. The project page, including video visualizations and additional results, is available at https://williamxuanyu.github.io/asscg/.

Keywords: Autonomous driving · LLM · Dual system

1 Introduction

Motion planning is a core component of autonomous driving. Despite notable progress in learning-based planning and standardized benchmarks [11,33], modern planners still struggle in complex, dynamic, long-tail interactions [5,6,12,14]. In parallel, large language models (LLMs) exhibit strong commonsense reasoning and cross-domain transfer; with instruction tuning, they can internalize traffic

* Corresponding authors.

rules and scenario priors to provide more context-aware guidance [2, 3, 8, 10, 13]. This has motivated LLM-augmented planning.

However, LLM inference latency and compute cost make per-step deployment difficult. A practical compromise is the decoupled fast–slow paradigm: a real-time fast planner runs every frame while an LLM-based slow module provides high-level guidance at a lower frequency [4, 25, 26]. Yet in practice, peak performance often requires querying the slow module nearly every frame; reducing the query rate typically degrades results. More importantly, adding a slow LLM module does not always help and can even hurt in certain segments (Fig. 2), raising a central question: *when should the system rely on the fast planner, and when should it consult (or ignore) the slow module, to maximize closed-loop performance under a compute budget?*

Two coordination strategies are widely used (Fig. 1): (i) fixed-interval triggering and (ii) heuristic difficulty-based triggering. Fixed schedules ignore intra-scene variability, leading to over-calling in easy segments and under-calling in hard ones. Difficulty-based gating requires defining and estimating "scene complexity", yet such proxies do not necessarily align with the *marginal utility* of invoking the slow module at a particular time. Empirically, VLMPlanner evaluates multiple gating variants (rule-based and learned complexity criteria) across different settings and hyperparameters, but none improves over the always-querying baseline in terms of final score; the best accuracy is still achieved by per-frame slow-module invocation, while the best efficiency–accuracy trade-off comes from a learned complexity gate that slightly reduces score but significantly improves throughput [25]. Moreover, most existing schemes operate at the scene level with fixed fast/slow frequencies per class, overlooking fine-grained temporal dynamics within a single episode.

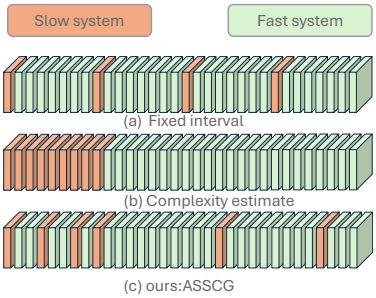

Fig. 1: Common fast–slow coordination strategies: (a) fixed-interval triggering ignores temporal variation, wasting compute in easy periods and missing critical moments; (b) difficulty/complexity-based triggering relies on imperfect proxies that not align with the value of slow reasoning, leading to unnecessary oscillation and mistimed queries; (c) ours (ASSCG), a learned gate that makes frame-level QUERY/CACHE/DROP decisions to balance planning quality and resource cost.

We propose the **A**daptive **S**low-**S**ystem **C**ontrol **G**ate (ASSCG), an architecture-agnostic control interface that adaptively schedules and regulates a slow module at the frame level. Here, architecture-agnostic means that ASSCG only assumes a fast branch, an optional slow branch, and a cached guidance representation; the gate is still trained or lightly adapted for the target planner and benchmark. We formulate slow-module control as a sequential decision problem with query costs under partial observability (POMDP/SMDP), and recast it as sequence

generation: the controller predicts a gating token per frame. This LLM-style formulation enables standard SFT-to-RL fine-tuning. Concretely, we implement ASSCG with an RWKV backbone [19] and fine-tune it with a compute-aware GRPO-style algorithm [23]. On nuPlan Hard20 closed-loop evaluation, integrating ASSCG into AsyncDriver (hereafter referred to as *AdaptiveAsyncDriver*) reduces average end-to-end latency by about 60% while improving the score by +2.28 (to 67.28). We further demonstrate the applicability of the same gating principle on a separate RecogDrive-based fast–slow instantiation evaluated on NAVSIM, where it improves PDMS by +0.6 and increases average speed by ∼25%. Our contributions are:

1. We analyze fast–slow collaboration and introduce *effective* and *failure* intervals to characterize when slow guidance helps or hurts over time.
2. We present ASSCG, a frame-level gate trained with GRPO-style reinforcement fine-tuning to optimize closed-loop metrics under query costs.
3. We evaluate ASSCG on two representative fast–slow instantiations and benchmarks: AsyncDriver on nuPlan closed-loop Hard20 and a RecogDrive-based fast–slow planner on NAVSIM, demonstrating improved performance–efficiency trade-offs across architectures and evaluation protocols.

2 Related Work

2.1 Language-augmented motion planning with LLMs

Large language models (LLMs) exhibit strong generalization and reasoning after pre-training on massive corpora, motivating their use in autonomous driving decision-making. Recent works encode ego state, surrounding agents, and map context as linguistic descriptions or BEV-style representations for LLM/VLM-based scene understanding and trajectory guidance [24, 30, 35]. For example, PlanAgent [35] structures scenes in BEV to guide a base planner, and LLM-ASSIST [24] integrates LLM reasoning with rule-based modules.

Purely language-mediated interfaces, however, face an expressivity–latency mismatch for time-critical control: limited context length and the difficulty of encoding precise continuous states can reduce reliability. Multimodal approaches—e.g., DrivingWithLLM [3], DriveGPT4 [29], and RAGDriver [31]—align vectorized or image/video inputs with language to enrich interpretation, but translating language outputs into stable closed-loop control remains challenging. Another line couples LLMs more tightly with low-level controllers for closed-loop driving, such as LMDrive [22] and LanguageMPC [28], yet these designs still rely on frequent LLM inference or serial decoding, stressing real-time responsiveness.

2.2 Fast–slow (dual-system) planners and adaptive LLM invocation

To better satisfy real-time constraints while leveraging LLM reasoning, fast–slow planners decouple a real-time fast planner from an LLM-based slow module that provides high-level guidance at a lower frequency [4, 26]. AsyncDriver [4] caches

slow guidance and reuses it across frames, amortizing cost but typically requiring careful scheduling to avoid stale or mis-timed interventions under non-stationary traffic.

A number of works study *when* to invoke the slow module. VLMPlanner [25] introduces CAI-Gate to adjust query frequency using simple/complex judgments from a VLM; its reported gains are marginal under multiple gating criteria, suggesting that generic scene-difficulty proxies may not reliably track the *value* of invoking the slow planner. Uncertainty-based triggering (e.g., FASIONAD [20]) calls the slow module under predicted waypoint uncertainty, but uncertainty calibration in interactive long-tail scenes can be brittle. Task-specific systems (e.g., Chameleon [34]) allocate extra slow reasoning for detected edge cases, often requiring curated traces or additional annotations.

Recent concurrent work. AdaDrive [32] proposes an adaptive slow–fast framework that learns *when* to activate LLM assistance via an adaptive activation loss, and further introduces a continuous fusion mechanism that scales the LLM contribution based on scene complexity and prediction confidence. Our work is complementary but differs in several aspects: (i) we learn an *architecture-agnostic*, *frame-level* discrete gate with explicit QUERY, CACHE, and DROP actions, where DROP enables intentionally suppressing slow guidance when it is likely to be harmful; (ii) we train the gate with supervised pretraining followed by compute-aware RL fine-tuning to directly optimize task-level driving metrics under an explicit query-cost constraint; and (iii) rather than scaling LLM influence using confidence as a primary signal, we focus on *when to consult* and *whether to trust* the slow module—in our setting, the fast planner already performs probability-based trajectory aggregation (GameFormer [16]), which can make confidence among shortlisted candidates less discriminative for further modulation. Overall, our goal is to learn just-right *timing* and *usage* of slow reasoning under resource constraints.

3 Analysis

3.1 Setup and assumptions

Following the AsyncDriver paradigm, we treat the LLM-based slow system as a high-level intent generator that does not need to be queried at every frame. The real-time fast planner consumes and caches the most recent slow-system guidance and continues planning until the next refresh. Concretely, let q_k denote the frames at which the slow system is invoked, z_{q_k} the resulting high-level guidance (e.g., latent features, waypoints, or policy logits), and u_t the low-level control produced by the fast planner at frame t using the latest cached guidance z_{q_k}, where $q_k \leq t < q_{k+1}$.

3.2 Concepts and operational definitions

To analyze when and how the slow system should be queried, we introduce three intervals and provide operational criteria for measurement in practice.

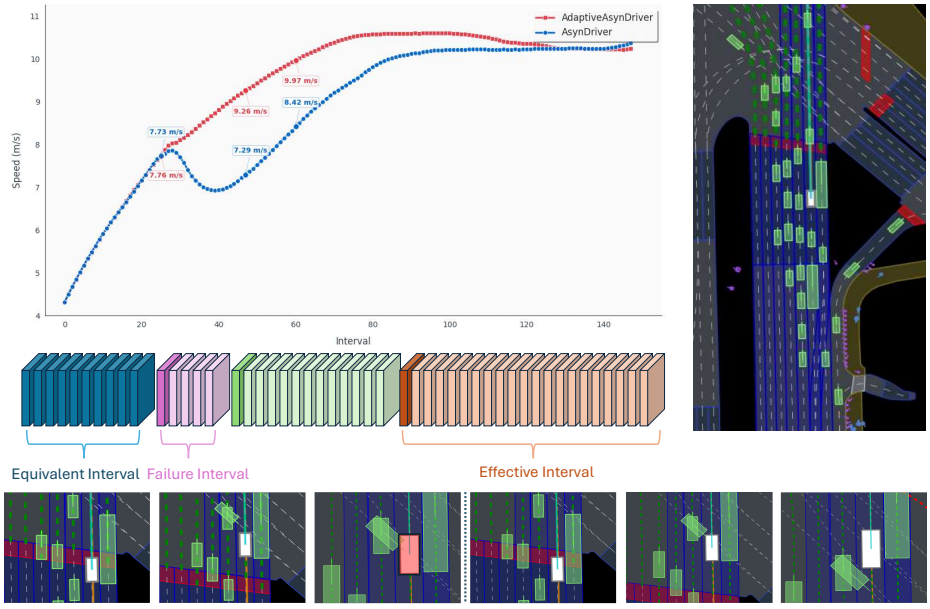

Fig. 2: Straight-driving case study comparing AsyncDriver (always querying the slow planner) and AdaptiveAsyncDriver with ASSCG (querying only at frames 0, 22, and 89). Despite far fewer slow-system calls, ASSCG avoids a collision and achieves better closed-loop behavior. Frames 0–25 form an Equivalent Interval (EI); frames 25–40 reveal a Failure Interval (FI) where slow guidance is harmful; and frames 1–25 and 90–149 are Effective Intervals (EfI) for guidance issued at frames 0 and 89, respectively. Additional qualitative visualizations are provided in the supplementary material.

Equivalent Interval (EI). A post-query time window in which re-querying the slow system would be redundant because the incremental information is negligible. Operationally, for $t \in [q_k, q_{k+1})$: (i) $similarity(z_{q_k}, z_{t'}) \geq \tau_{sim}$ for all t' in the window, where similarity can be cosine similarity in the slow-system latent space or an embedding distance after a projection; or (ii) the induced control difference is negligible, e.g., $||u_t(z_{q_k}) - u_t(z_{q_{k'}})|| \leq \tau_{ctrl}$ for a hypothetical re-query at frame $q_{k'} = t$. Intuitively, EI indicates that the slow guidance remains valid and "fresh," and repeated slow queries would not change the planned motion in a meaningful way.

Effective Interval (EfI). The span during which the current slow guidance is actually used by the fast planner before being refreshed, i.e., $[q_k, q_{k+1}]$. By definition EfI is the cache-validity window; in practice it may be longer or shorter than EI. When $EfI \approx EI$, caching is efficient; when $EfI \gg EI$, stale guidance can accumulate; when $EfI \ll EI$, compute is wasted.

Failure Interval (FI). A time window in which invoking the slow system degrades downstream performance relative to not invoking it. Operationally, for a candidate call at t: $\Delta J_t = J(u_t \mid \text{call at } t) - J(u_t \mid \text{no call at } t) < -\tau_{\text{perf}}$, where J

aggregates safety and efficiency objectives (e.g., collision, lane-keeping, progress, comfort). Typical causes include misestimated distances, occlusion-induced hallucinations, or misinterpretation of local interaction rules. FI emphasizes that more frequent slow reasoning is not always better.

These definitions align with a value-of-information perspective: query the slow system only when the expected improvement in J exceeds the query cost and the risk of FI.

3.3 Case study and empirical observations

Fig. 2 illustrates a straight-driving episode comparing AsyncDriver (querying the slow module every frame) and our approach (querying only at frames 0, 22, and 89). Despite making only three slow-module calls, our gate avoids a collision and outperforms the always-query baseline. Notably, frames 0–25 behave as an Equivalent Interval (re-querying yields negligible change), while frames 25–40 expose a Failure Interval where slow guidance is detrimental in this instance. This example motivates the need for a gate that can both *reuse* and *suppress* slow guidance depending on context.

Beyond a single case: fixed-schedule grid search across scenario types. To check whether the above behavior is isolated, we run a grid search of *fixed* slow-module query schedules on nuPlan Hard20, separately for each of the 14 scenario types. We evaluate fixed query stride $K \in \{1, 3, 5, 10, 20, 50, 100\}$ (query every K frames) and a *Never-Query* setting that disables the slow module. Across types, the best fixed schedule varies substantially—from always querying (K=1) to rarely querying (K=50), and in one case *Never-Query* performs best. Notably, this grid search assigns a *single* stride to all episodes within a type; the optimal stride can still differ across episodes within the same type due to intra-scene temporal dynamics. This further motivates learning a frame-level policy that adapts query decisions online. Detailed per-type results are provided in the supplementary material.

3.4 Implications for gating policy design

The analysis motivates a frame-level gating policy with three objectives:

1. Minimize redundant slow queries within Equivalent Intervals by reusing previously issued slow guidance and deferring new slow calls while the scene and control remain within the same equivalence window.
2. Suppress slow queries in segments unsuitable for LLM involvement (e.g., Failure Intervals), thereby avoiding adverse interventions.
3. Extend Effective Intervals by continuing to use cached slow guidance until a refresh is needed, subject to a maximum reuse horizon.

ASSCG operationalizes these objectives by casting the slow-trigger decision as a sequential problem under a constrained query budget. It seeks to maximize J

by invoking the slow system only when the expected marginal benefit justifies the query cost; otherwise, it reuses cached guidance and leverages the fast planner. Implementation details are provided in the following section.

4 Methodology

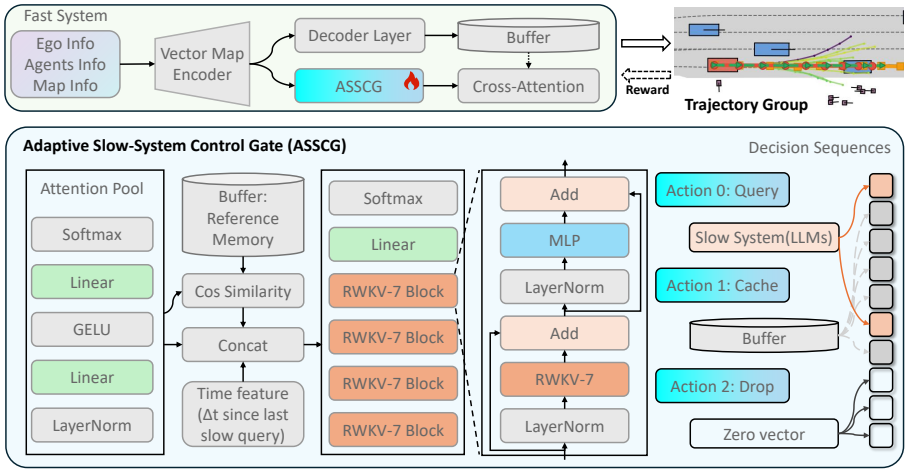

Fig. 3: Overview of our framework. We couple a GameFormer-based [16] fast planning system with an LLM-based slow system, coordinated by an Adaptive Slow-System Control Gate (ASSCG). At each frame, encoded vector-map (and ego/agent features) is fed to both the fast decoder and ASSCG. ASSCG outputs a discrete gating action: **Query** invokes the slow system to refresh a reference-memory buffer, **Cache** reuses the buffer without querying the slow system, and **Drop** ignores the buffer (zero vector) to prevent stale memory from affecting planning. Conditioned on the gating decisions, the fast decoder fuses fast features with the (optional) buffered reference via cross-attention and outputs a trajectory for one rollout. A *trajectory group* is obtained by sampling multiple rollouts, whose scores are converted to rewards to optimize the per-frame gating policy with group-based RL (GRPO). Inside ASSCG, a lightweight attention pool aggregates cues, which are compared with the buffer via cosine similarity and concatenated with time feature before an RWKV backbone predicts the gating policy.

4.1 Overview

Our framework follows a decoupled fast–slow architecture. A learning-based real-time planner (fast system) runs every frame, while an LLM-based planner (slow system) is invoked on demand to provide high-level guidance that is cached in a buffer and fused into the fast planner via cross-attention. The Adaptive Slow-System Control Gate (ASSCG) decides at each frame whether to (i) QUERY the

slow system and refresh the buffer, (ii) CACHE and reuse the buffered guidance, or (iii) DROP the guidance by substituting a null feature.

Training pipeline and parameter freezing. Following AsyncDriver, the underlying fast–slow planner is trained in three stages: (i) we first train the fast planner (GameFormer-style) independently; (ii) we then fine-tune the slow LLM (LLaMA2-13B) with LoRA for driving-domain adaptation; and (iii) we subsequently train the full fast–slow planner end-to-end while freezing the LLM weights, so that the fast planner and fusion modules adapt to the LLM guidance without updating the LLM itself. After inserting ASSCG, we perform an additional, separate training stage for the gate: we freeze *all* parameters of the original AsyncDriver and optimize only ASSCG. This isolates the effect of improved slow-system scheduling and selective usage from further planner adaptation.

4.2 Model architecture

The core dual-system architecture follows the AsyncDriver paradigm. Let m_t denote the vectorized map-and-agent features produced by a Vector Map Encoder at frame t; the fast planner's decoder consumes m_t to predict low-level control and a short-horizon trajectory. When available, cached slow guidance b_t from the buffer is injected into the decoder through a cross-attention path. The slow system (LLM) is queried only when gated "on," producing high-level intent z_t (e.g., latent features, waypoints, or policy logits), which is written to the buffer.

Concretely, in a decoder block l, we extend the standard attention with an instruction-injection:

$$s^{l+1} = g \cdot \text{softmax}\left(\frac{QK_{\widehat{h}}^T}{\sqrt{C}}\right) V_{\widehat{h}} + \text{softmax}\left(\frac{QK_{s^l}^T}{\sqrt{C}}\right) V_{s^l}$$

where h is the instruction feature projected from the slow guidance (last-token hidden state after a small adapter), s^l is the scene feature at block l, and g is a learnable gate.

Per iteration, we obtain vectorized ego&agent histories and map elements in the ego-centric frame, similar to GameFormer-style encoders. The Vector Map Encoder produces tokenized features for lanes, topology, and dynamic agents. These tokens drive the fast planner, while ASSCG decides whether to reuse features in the buffer, refresh the buffer with new LLM-derived features, or bypass LLM features altogether.

4.3 Buffering and guidance representation

Buffer content. The buffer stores a compact slow-guidance representation $b = Proj(z)$, along with map tokens and the current timestamp (frame). **Read/write policy.** When the slow system is queried at frame t, we overwrite $b \leftarrow Proj(z_t)$. When not queried, we reuse the previous b. When the gate predicts "drop," we expose a null vector b_0 to the planner, effectively disabling slow guidance.

4.4 ASSCG: Adaptive Slow-System Control Gate

The gating policy must determine how the planner should interact with the buffer at every frame— whether to refresh it with new slow guidance, keep using the existing contents, or intentionally ignore it to avoid harmful interventions. This decision depends on the temporal context, cached features, and the learned scheduling strategy.

Problem framing: At each frame t, ASSCG produces an action $a_t \in \{0, 1, 2\}$:

- 0 = Query (invoke the slow system and refresh the buffer),
- 1 = Cache (reuse the buffer as-is),
- 2 = Drop (ignore the buffer by substituting a null feature).

This realizes our interval analysis: CACHE extends effective intervals, QUERY marks the start of a new interval, and DROP suppresses suspected failure intervals. The first frame of every episode is forced to QUERY to initialize the buffer, so CACHE is never applied without valid slow guidance.

Inputs to the gate: Current scene embedding x_t from the Vector Map Encoder (token sequence); Reference memory b_{ref} from the buffer (the last slow-guidance embedding); Cosine similarity $s = cos(Pool(x_t), b_{ref})$, where $Pool(\cdot)$ is a lightweight attention pooling that summarizes x_t; Time feature is Δt since the last slow query (normalized via logarithmic scaling). The gate concatenates $[Pool(x_t), s, \Delta t]$ as its input feature.

Attention pooling: We employ a compact attention-pooling block (Linear–GELU–Linear–Softmax with LayerNorm) to summarize the token sequence. It mirrors the Map Adapter design in AsyncDriver for map embeddings, ensuring architectural consistency and integrating vectorized map features with minimal computational overhead.

Why RWKV for gating. ASSCG must make low-latency, frame-level decisions while tracking long-horizon temporal context (up to an entire episode). We adopt RWKV [19] for two reasons. First, its recurrent time-mixing can be viewed as maintaining a compact, recursively updatable latent state, which matches the intuition that the gate benefits from an internal belief state under partial observability. Second, RWKV provides a favorable deployment profile for online decision-making with long context: after state caching, it has per-step linear time and constant memory, avoiding the quadratic attention cost of Transformers as the episode length grows. We empirically compare RWKV with a parameter-matched Transformer alternative in the ablation study (§6.4).

RWKV backbone and head: The pooled feature and auxiliaries are fed to a 4-layer RWKV-7 stack. A linear classification head with softmax outputs $\pi(a_t \mid s_t)$ over {QUERY, CACHE, DROP}. As discussed above, RWKV offers an efficient long-horizon gating backbone; quantitative accuracy/latency comparisons against Transformer-based gating are reported in the ablations.

5 Data Collection and Training Protocol

5.1 Datasets

Scenario types and splits. We follow the nuPlan Challenge 2023 closed-loop setting and evaluate on the 14 official scenario types [11]. For a fair comparison with prior fast–slow planners, we use the *Hard20* test split released by AsyncDriver [4] and VLM-planner [25] (same scene IDs and evaluation protocol), and report results on this fixed split throughout.

SFT training pool. From the training split, we randomly sample 60 scenes per scenario type (14 types). We collect training rollouts by executing a set of fixed slow-module query schedules in the nuPlan simulator, using query stride $K \in \{1, 3, 5, 10, 20, 50, 100\}$ (query every K frames) and a *Never-Query* setting that disables the slow module. For each rollout, we log per-frame encoder features, the slow-module output (when queried), the gate action token, and metric components of the nuPlan score.

To obtain supervision for the gate, we select, *for each scene*, the rollout with the highest closed-loop score among the above schedules, and use its per-frame QUERY/CACHE/DROP action sequence as pseudo labels for supervised fine-tuning (SFT).

RL data preparation. We construct an RL dataset by rolling out the SFT-initialized gate policy in the nuPlan simulator and then training from the logged trajectories. The gate action space has $A = 3$ actions (QUERY, CACHE, DROP); each episode spans the full closed-loop horizon.

We adopt an offline RL workflow: using the frozen SFT policy π_0, each training scene is rolled out $R = 10$ times with different random seeds. Actions are selected via entropy-shaped temperature scaling with ε-mixing. For a state s, let $p = \mathrm{softmax}(\mathrm{logits}(s))$ over the action set A, and define

$$\begin{aligned} H(s) &= -\sum_{i=1}^{A} p_i \log p_i, \quad H_{\mathrm{norm}}(s) = \frac{H(s)}{\log A} \in [0,1], \\ T(s) &= T_{\mathrm{low}} + \big(T_{\mathrm{high}} - T_{\mathrm{low}}\big)\big(H_{\mathrm{norm}}(s)\big)^{\alpha}, \\ \varepsilon(s) &= \varepsilon_{\mathrm{low}} + \big(\varepsilon_{\mathrm{high}} - \varepsilon_{\mathrm{low}}\big)\big(H_{\mathrm{norm}}(s)\big)^{\alpha}. \end{aligned} \tag{1}$$

With $p_T(a \mid s) = \mathrm{softmax}(\mathrm{logits}(s)/T(s))$ and $U(a) = 1/A$, the behavior distribution is

$$\begin{aligned} p_{\mathrm{mix}}(a \mid s) &= (1 - \varepsilon(s))\, p_T(a \mid s) + \varepsilon(s)\, U(a), \\ a &\sim \mathrm{Cat}\big(p_{\mathrm{mix}}(\cdot \mid s)\big) \end{aligned} \tag{2}$$

using T_{low}=0.9, T_{high}=1.6, $\varepsilon_{\mathrm{low}}$=0.02, $\varepsilon_{\mathrm{high}}$=0.12, α=1.0. We force a QUERY at the first step of each episode to initialize the buffer. The resulting static dataset D logs per-step s (encoder features), a, reward metrics for offline reward reconstruction (e.g., TTC, Lim, Comf, Prog), the behavior log-probability $\log b = \log p_{\mathrm{mix}}(a \mid s)$, as well as $H_{\mathrm{norm}}(s)$, $T(s)$, $\varepsilon(s)$, and metadata (done, episode_id, step_idx, timestamps).

5.2 Training protocol

We train ASSCG in two stages: supervised fine-tuning (SFT) on pseudo labels, followed by compute-aware RL fine-tuning.

Stage I: Supervised fine-tuning. Over the action set $a_t \in \{\textsc{Query}, \textsc{Cache}, \textsc{Drop}\}$, we train a 3-way classifier using the per-frame action sequence from the highest-scoring fixed-schedule rollout of each scene (§5.1). We minimize cross-entropy with class-balancing to avoid the degenerate Cache-only policy. We use AdamW (lr 3×10^{-4}, weight decay 0.01) with cosine decay and 10% warmup, label smoothing 0.05, and gradient-norm clipping at 1.0.

Stage II: Compute-aware RL fine-tuning. The nuPlan closed-loop score is computed as a weighted sum of per-episode metrics, normalized by the sum of weights $W = 16$, and multiplied by gating terms (e.g., collision/drivable/direct). Since the official score is episode-level, it provides sparse supervision. We introduce dense intermediate feedback via potential-based shaping while preserving alignment with the official weighting. Let $W_{\text{np}} = 11$ denote the total weight of the non-progress terms (*TTC*, *Lim*, *Comf*). Define

$$N_t = 5\,\text{TTC}_t + 4\,\text{Lim}_t + 2\,\text{Comf}_t, \quad G_t = \text{Coll}_t \times \text{Drivable}_t \times \text{Direct}_t, \tag{3}$$

and the normalized potential

$$\Phi_t = \frac{1}{W} N_t\, G_t \;=\; \frac{W_{\text{np}}}{W} \cdot \underbrace{\Big(\frac{N_t}{W_{\text{np}}} G_t\Big)}_{\psi_t \in [0,1]}. \tag{4}$$

The per-step reward is

$$r_t = (\gamma \Phi_{t+1} - \Phi_t) \;-\; \lambda_{\text{call}} \cdot \mathbb{I}[a_t = \textsc{Query}], \tag{5}$$

with an additional terminal progress settlement

$$r_T \leftarrow r_T + \frac{W - W_{np}}{W}\,\text{Prog}_T\, G_T. \tag{6}$$

With $\gamma \approx 1$, the shaping term telescopes, yielding a constant-shifted version of the non-progress part of the official nuPlan score, and the terminal term restores the progress component. This yields an objective consistent with the official evaluation but with denser learning signal. The compute-awareness is explicit in the objective: every Query action receives a penalty controlled by λ_{call}, while Cache and Drop avoid this cost. Increasing λ_{call} shifts the policy toward longer effective query intervals, trading slow-module usage against closed-loop score in a controlled way.

We fine-tune the policy using a GRPO-style clipped policy-gradient objective with (i) off-policy correction using logged behavior probabilities, (ii) KL regularization to the SFT reference policy π_{ref}, and (iii) an entropy bonus for exploration. Concretely, with importance ratio $\rho_t = \pi_\theta(a_t|s_t)/b(a_t|s_t)$ truncated by $\rho_{\max}$, we apply PPO-style clipping with range ϵ; full objective and advantage estimation details are provided in the supplementary material.

Hyperparameters. Unless otherwise stated: γ=0.99, GAE λ=0.95, λ_{call}=0.001, clip ϵ=0.2, $\rho_{\max}$=10, entropy coefficient η=0.01, value loss coefficient λ_V=0.5, and KL coefficient β is set to 0.01. We use AdamW with lr 1×10^{-4} and gradient-norm clipping at 1.0.

6 Experiments

6.1 Experimental Setup

General settings. Unless otherwise stated, all reported scores and inference latencies are measured with single-GPU inference on an NVIDIA A30. To reduce variance from RL fine-tuning, we report the average over three runs with different random seeds.

nuPlan setup. We evaluate in the nuPlan closed-loop reactive setting: non-ego agents follow IDM-based planners and respond to the ego's actions. The simulation runs at 10 Hz, and each planning iteration predicts an 8 s trajectory horizon. We use the official nuPlan closed-loop metrics; definitions and the composite score formula are summarized in the supplementary material. For fairness with prior work, we use the fixed Hard20 split from AsyncDriver & VLMplanner (same scene IDs).

NAVSIM setup. We also evaluate on NAVSIM, a planning-oriented benchmark with a non-reactive (pseudo closed-loop) simulator. Each scene requires one-shot prediction of a 4 s future trajectory (8 waypoints at 0.5 s). We report the official PDMS metric (summarized in the supplementary material).

6.2 Results on nuPlan (AsyncDriver-based Fast–Slow Planner)

On nuPlan Hard20, integrating ASSCG into AsyncDriver improves the overall score to 67.28 (+2.28 absolute) while reducing compute by lowering the slow-module query rate. Detailed per-scenario-type results are provided in the supplementary material. Overall, we observe improvements on most scenario types, with the largest gains on lane-change and intersection-related categories, and only minor regressions on two types (type0 and type7), which are within a small margin.

In addition to the overall score, Table 1 reports a breakdown of nuPlan metric components. ASSCG improves drivable-area and driving-direction compliance as well as TTC, indicating that the gain is primarily driven by safer and more stable closed-loop behavior.

Beyond accuracy, ASSCG yields consistent efficiency benefits. Table 3 compares three settings enabled by the asynchronous fast–slow design of AsyncDriver: always querying the slow module, a fixed-stride schedule that queries once every 5 frames, and ASSCG. The 5-frame fixed schedule has nearly the same average latency as ASSCG (0.32 s/frame), providing a latency-matched

Table 1: nuPlan Hard20 closed-loop evaluation: score breakdown. We report the overall score and the average values of metric components used by the official nuPlan score. *Drivable*: drivable-area compliance; *Direct*: driving-direction compliance; *Comf*: comfort; *Prog*: route progress; *Coll*: no at-fault collision factor; *Lim*: speed-limit compliance; *TTC*: time-to-collision within bound.

Method	Score	Drivable	Direct.	Comf.	Prog.	Coll.	Lim.	TTC
UrbanDriver [21]	35.35	75.53	97.12	**98.56**	**85.23**	55.21	81.62	47.84
GCPGP [12]	36.85	81.29	98.20	77.33	46.96	72.30	97.92	68.34
IDM [27]	53.07	84.94	98.02	83.15	64.79	74.01	96.38	60.57
GameFormer [16]	62.05	93.54	98.74	83.15	66.27	86.02	98.19	74.55
PDM-Hybird [9]	64.05	95.34	99.10	75.98	67.93	**87.81**	**99.57**	72.75
PDM-Closed [9]	64.18	95.69	99.10	77.06	68.20	**87.81**	**99.57**	73.47
VLM-planner [25]	66.66	90.91	95.65	84.58	84.29	76.88	99.37	70.36
AsyncDriver	65.00	94.62	98.75	83.15	67.13	85.13	98.15	73.48
ours	**67.28**	**96.40**	**99.27**	86.66	68.16	86.99	98.14	**75.93**

baseline. Under this comparable efficiency budget, ASSCG improves the score from 64.27 to 67.28 (+3.01), indicating that the gain comes primarily from *better timing and selective suppression* of slow-module calls rather than merely reducing their frequency. On the Hard20 test split, ASSCG achieves an average effective query interval of 5.26 frames, corresponding to a slow-query rate of roughly 19.0% of frames. This is close to the fixed 5-frame schedule in latency, but differs in timing and in whether cached slow guidance should be trusted or dropped.

Compared with VLM-planner, ASSCG has lower route progress but improves the safety-related and rule-compliance terms (drivable area, driving direction, comfort, collision, and TTC), yielding a higher composite score. This suggests that VLM-planner is more aggressive on Hard20, whereas ASSCG trades some progress for more stable closed-loop behavior.

6.3 Results on NAVSIM (RecogDrive-based Fast–Slow Planner)

We evaluate ASSCG on NAVSIM using a RecogDrive-based fast–slow instantiation (see supplementary material). We keep the original VLM module (InternVL-2B) as the slow branch and construct a lightweight fast branch by replacing the VLM with a ViT backbone; both branches use the same diffusion-planner architecture but are trained as separate policies (no weight sharing). Since NAVSIM is one-shot (predicting a single 4 s trajectory per scene), we use a simplified ASSCG without buffer/time features and with a binary decision (fast vs. slow). To better reflect the sensitivity to future trajectory quality in this setting, we additionally encode the fast-branch predicted trajectory with a lightweight MLP (mapped to a 256-d vector) and feed it to the gate as an auxiliary cue. The NAVSIM gate is trained with RL to directly optimize PDMS.

Table 2: Compact NAVSIM comparison. Full results are in the supplementary material. ‡Concurrent preprints that appeared after our initial submission.

Method	NC↑	DAC↑	TTC↑	Comf.↑	EP↑	PDMS↑
UniAD [15]	97.8	91.9	92.9	**100**	78.8	83.4
TransFuser [7]	97.7	92.8	92.8	**100**	79.2	84.0
Hydra-MDP [18]	98.3	96.0	94.6	**100**	78.7	86.5
Curious-VLA [1]	98.4	96.9	97.9	98.1	88.5	90.3
ReCogDrive [17]	97.9	97.3	94.9	**100**	87.3	90.8
DiffusionDriveV2 [36]‡	**98.3**	97.9	94.8	99.9	**88.0**	91.2
ours	98.2	**98.3**	94.8	**100**	87.5	**91.4**

Table 3: Performance and per-frame inference latency.

Method	Score ↑	Latency (s/frame) ↓
AsyncDriver	65.00	0.80
AsyncDriver (5-frame interval)	64.27	0.32
AdaptiveAsyncDriver (ours)	**67.28**	**0.32**

Table 2 shows a compact comparison. Our RecogDrive-based dual system with ASSCG achieves 91.4 PDMS, outperforming the original ReCogDrive (90.8) and reducing average wall-clock inference latency from ∼350 ms to ∼270 ms on a single A30 GPU. Although the system maintains two trained branches, only one branch is executed per scene after gating in this one-shot setting, so the reported wall-clock latency includes the gating overhead and selected branch inference. Full NAVSIM comparisons are provided in the supplementary material.

6.4 Ablation Study

Table 4 reports ablations on training strategy (Table 4(a)) and backbone choice (Table 4(b)) for AdaptiveAsyncDriver.

Training strategy. Supervision-only gating achieves 66.21%. Adding on-policy fine-tuning with GRPO improves the score to 67.28%, indicating that, without access to optimal labels, post-SFT RL can outperform the fixed-interval surrogate by exploring and refining schedules, suppressing redundant calls in equivalent intervals, and skipping low-yield windows. Replacing GRPO with PPO yields 66.76%, but we observe instability and mode collapse, likely due to the value network's difficulty in fitting the composite, rule-based score and estimating long-horizon returns.

Backbone choice. Replacing the RWKV backbone in ASSCG with a Transformer block of similar parameter count gives comparable accuracy (67.21% vs. 67.28%), but latency scales worse: Transformer attention is quadratic in context length L ($O(L^2)$ time/memory per layer), while RWKV scales linearly in time and $O(1)$ in memory per step after state caching. For $L = 150$ (covering a full

Table 4: Ablation study of AdaptiveAsyncDriver. LFL denotes last-frame gate latency.

(a) Training strategy

AsyncDriver	SFT	PPO	GRPO	Score ↑
✓				65.00
✓	✓			66.21
✓	✓	✓		66.76
✓	✓		✓	67.28

(b) Backbone and latency

Backbone	Score ↑	LFL (s) ↓
RWKV	67.28	0.02
Transformer	67.21	0.06

scene), last-frame gate latency is ∼0.06 s for Transformer vs. 0.02 s for RWKV on the same A30 GPU. We report *last-frame* gate latency as a practical worst-case proxy: in long-horizon episodes the gate runs with the maximum context length, which upper-bounds the additional per-step overhead and is more relevant for real-time responsiveness than average latency. Although windowing or truncation can reduce L, real driving often exhibits variable and long temporal contexts; RWKV's linear scaling thus provides a more robust latency profile, making it our default backbone.

Additional ablations (e.g., query-cost weight λ_{call}, removing DROP, and DROP usage statistics), qualitative visualizations, and our RecogDrive-based NAVSIM instantiation are provided in the supplementary material. In particular, removing DROP lowers the score from 67.28 to 66.71 despite a similar query interval, showing that ASSCG does more than reduce query frequency: it can intentionally suppress harmful slow guidance.

7 Conclusion

In this work, we analyzed limitations of existing coordination schemes for fast–slow planning and introduced concepts such as the *Equivalent Interval* and *Failure Interval*. Building on this analysis, we proposed the Adaptive Slow-System Control Gate (ASSCG), an architecture-agnostic control interface that schedules and regulates interactions between a learning-based fast planner and an LLM-based slow system. We validated ASSCG on two representative fast–slow instantiations and benchmarks—AsyncDriver on nuPlan Hard20 closed-loop evaluation and a RecogDrive-based fast–slow planner on NAVSIM—showing improved performance–efficiency trade-offs (higher scores with fewer or better-timed slow-module calls). A limitation is that our experiments evaluate when to use a slow branch given existing fast–slow systems, rather than proving that LLM reasoning is necessary for every benchmark or scenario. Harder end-to-end long-tail benchmarks are a natural next step.

Acknowledgements

Funded by Xiongan AI Institute, Lenovo Research and Wuxi Research Institute of Applied Technologies, Tsinghua University under Grant 20242001120.

References

1. Chen, C., Yang, Y., Tan, Z., Wang, Y., Zhan, R., Liu, H., Mao, X., Bao, J., Tang, X., Yang, L., et al.: Devil is in narrow policy: Unleashing exploration in driving vla models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1062–1072 (2026)
2. Chen, L., Zhang, Y., Ren, S., Zhao, H., Cai, Z., Wang, Y., Wang, P., Liu, T., Chang, B.: Towards end-to-end embodied decision making via multi-modal large language model: Explorations with gpt4-vision and beyond. arXiv preprint arXiv:2310.02071 (2023)
3. Chen, L., Sinavski, O., Hünermann, J., Karnsund, A., Willmott, A.J., Birch, D., Maund, D., Shotton, J.: Driving with llms: Fusing object-level vector modality for explainable autonomous driving. In: 2024 IEEE International Conference on Robotics and Automation (ICRA). pp. 14093–14100. IEEE (2024)
4. Chen, Y., Ding, Z.h., Wang, Z., Wang, Y., Zhang, L., Liu, S.: Asynchronous large language model enhanced planner for autonomous driving. In: European Conference on Computer Vision. pp. 22–38. Springer (2024)
5. Cheng, J., Chen, Y., Chen, Q.: Pluto: Pushing the limit of imitation learning-based planning for autonomous driving. arXiv preprint arXiv:2404.14327 (2024)
6. Cheng, J., Chen, Y., Mei, X., Yang, B., Li, B., Liu, M.: Rethinking imitation-based planners for autonomous driving. In: 2024 IEEE International Conference on Robotics and Automation (ICRA). pp. 14123–14130. IEEE (2024)
7. Chitta, K., Prakash, A., Jaeger, B., Yu, Z., Renz, K., Geiger, A.: Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(11), 12878–12895 (2022)
8. Cui, C., Ma, Y., Cao, X., Ye, W., Wang, Z.: Receive, reason, and react: Drive as you say, with large language models in autonomous vehicles. IEEE Intelligent Transportation Systems Magazine **16**(4), 81–94 (2024)
9. Dauner, D., Hallgarten, M., Geiger, A., Chitta, K.: Parting with misconceptions about learning-based vehicle motion planning. In: Conference on Robot Learning. pp. 1268–1281. PMLR (2023)
10. Fu, D., Li, X., Wen, L., Dou, M., Cai, P., Shi, B., Qiao, Y.: Drive like a human: Rethinking autonomous driving with large language models. In: 2024 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW). pp. 910–919. IEEE (2024)
11. H. Caesar, J. Kabzan, K.T.e.a.: Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. In: CVPR ADP3 workshop (2021)
12. Hallgarten, M., Stoll, M., Zell, A.: From prediction to planning with goal conditioned lane graph traversals. In: 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC). pp. 951–958. IEEE (2023)
13. Han, W., Guo, D., Xu, C.Z., Shen, J.: Dme-driver: Integrating human decision logic and 3d scene perception in autonomous driving. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 39, pp. 3347–3355 (2025)
14. Hu, Y., Chai, S., Yang, Z., Qian, J., Li, K., Shao, W., Zhang, H., Xu, W., Liu, Q.: Solving motion planning tasks with a scalable generative model. In: European Conference on Computer Vision. pp. 386–404. Springer (2024)
15. Hu, Y., Yang, J., Chen, L., Li, K., Sima, C., Zhu, X., Chai, S., Du, S., Lin, T., Wang, W., et al.: Planning-oriented autonomous driving. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 17853–17862 (2023)

16. Huang, Z., Liu, H., Lv, C.: Gameformer: Game-theoretic modeling and learning of transformer-based interactive prediction and planning for autonomous driving. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3903–3913 (2023)
17. Li, Y., Xiong, K., Guo, X., Li, F., Yan, S., Xu, G., Zhou, L., Chen, L., Sun, H., Wang, B., et al.: Recogdrive: A reinforced cognitive framework for end-to-end autonomous driving. arXiv preprint arXiv:2506.08052 (2025)
18. Li, Z., Li, K., Wang, S., Lan, S., Yu, Z., Ji, Y., Li, Z., Zhu, Z., Kautz, J., Wu, Z., et al.: Hydra-mdp: End-to-end multimodal planning with multi-target hydra-distillation. arXiv preprint arXiv:2406.06978 (2024)
19. Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Grella, M., et al.: Rwkv: Reinventing rnns for the transformer era. arXiv preprint arXiv:2305.13048 (2023)
20. Qian, K., Ma, Z., He, Y., Luo, Z., Shi, T., Zhu, T., Li, J., Wang, J., Chen, Z., He, X., et al.: Fasionad: Fast and slow fusion thinking systems for human-like autonomous driving with adaptive feedback. arXiv preprint arXiv:2411.18013 (2024)
21. Scheel, O., Bergamini, L., Wolczyk, M., Osiński, B., Ondruska, P.: Urban driver: Learning to drive from real-world demonstrations using policy gradients. In: Conference on Robot Learning. pp. 718–728. PMLR (2022)
22. Shao, H., Hu, Y., Wang, L., Song, G., Waslander, S.L., Liu, Y., Li, H.: Lmdrive: Closed-loop end-to-end driving with large language models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15120–15130 (2024)
23. Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al.: Deepseekmath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300 (2024)
24. Sharan, S., Pittaluga, F., Chandraker, M., et al.: Llm-assist: Enhancing closed-loop planning with language-based reasoning. arXiv preprint arXiv:2401.00125 (2023)
25. Tang, Z., Zhang, S., Deng, J., Wang, C., You, G., Huang, Y., Lin, X., Zhang, Y.: Vlmplanner: Integrating visual language models with motion planning. arXiv preprint arXiv:2507.20342 (2025)
26. Tian, X., Gu, J., Li, B., Liu, Y., Wang, Y., Zhao, Z., Zhan, K., Jia, P., Lang, X., Zhao, H.: Drivevlm: The convergence of autonomous driving and large vision-language models. arXiv preprint arXiv:2402.12289 (2024)
27. Treiber, M., Hennecke, A., Helbing, D.: Congested traffic states in empirical observations and microscopic simulations. Physical review E **62**(2), 1805 (2000)
28. Wu, D., Han, W., Liu, Y., Wang, T., Xu, C.z., Zhang, X., Shen, J.: Language prompt for autonomous driving. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 39, pp. 8359–8367 (2025)
29. Xu, Z., Zhang, Y., Xie, E., Zhao, Z., Guo, Y., Wong, K.Y.K., Li, Z., Zhao, H.: Drivegpt4: Interpretable end-to-end autonomous driving via large language model. IEEE Robotics and Automation Letters (2024)
30. Yao, R., Wang, Y., Liu, H., Yang, R., Peng, Z., Zhu, L., Ma, J.: Calmm-drive: Confidence-aware autonomous driving with large multimodal model. arXiv preprint arXiv:2412.04209 (2024)
31. Yuan, J., Sun, S., Omeiza, D., Zhao, B., Newman, P., Kunze, L., Gadd, M.: Rag-driver: Generalisable driving explanations with retrieval-augmented in-context learning in multi-modal large language model. arXiv preprint arXiv:2402.10828 (2024)

32. Zhang, R., Xie, J., Zhang, W., Chen, W., Tan, X., Wan, X., Li, G.: Adadrive: Self-adaptive slow-fast system for language-grounded autonomous driving. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5112–5121 (2025)
33. Zhang, Y., Zhang, S., Zhang, Y., Ji, J., Duan, Y., Huang, Y., Peng, J., Zahng, Y.: Multi-modality fusion perception and computing in autonomous driving. J. Comput. Res. Dev **57**, 1781–1799 (2020)
34. Zhang, Z., Li, X., Zou, S., Chi, G., Li, S., Qiu, X., Wang, G., Zheng, G., Wang, L., Zhao, H., et al.: Chameleon: Fast-slow neuro-symbolic lane topology extraction. arXiv preprint arXiv:2503.07485 (2025)
35. Zheng, Y., Xing, Z., Zhang, Q., Jin, B., Li, P., Zheng, Y., Xia, Z., Zhan, K., Lang, X., Chen, Y., et al.: Planagent: A multi-modal large language agent for closed-loop vehicle motion planning. arXiv preprint arXiv:2406.01587 (2024)
36. Zou, J., Chen, S., Liao, B., Zheng, Z., Song, Y., Zhang, L., Zhang, Q., Liu, W., Wang, X.: Diffusiondrivev2: Reinforcement learning-constrained truncated diffusion modeling in end-to-end autonomous driving. arXiv preprint arXiv:2512.07745 (2025)