SCALABLE ROLLOUT GENERATION FOR GENERATIVE DRIVING SIMULATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative simulators are emerging as a promising foundation for post-training of autonomous driving policies. However, scaling training in these environments is limited by the cost of sensor simulation and the sequential dependency between observation rendering and policy inference. The simulator waits for the policy to predict the next ego pose, while the policy waits for the resulting observation before replanning. This ordering introduces idle time and limits rollout throughput. Inspired by speculative decoding, we introduce *Speculative Rendering*, a rollout-generation framework that speculates *future simulation states* ahead of planner execution. Our approach renders future observations in parallel with the planner, and employs a verifier to determine whether the speculated state remains sufficiently close to the realized state. When the divergence exceeds an acceptance threshold, the speculated renders are discarded and the simulator state is resynchronized with the target planner. We evaluate Speculative Rendering for GRPO post-training of AutoVLA using WorldEngine, a generative driving simulator. Speculative Rendering reduces mean simulation-step latency from 364 to 138 ms and lowers rollout-generation cost from 840 to 488 H100 GPU-hours, a 42% reduction. The resulting policy trained on speculative rollouts improves PDMS on the challenging `navtest-failure` benchmark by 12.66% and remain within 1% of the performance obtained using standard sequential rendering.

1 INTRODUCTION

Generative simulators, including neural reconstruction (Yang et al., 2023; Tonderski et al., 2024; NVIDIA et al., 2025) and video diffusion models (NVIDIA et al., 2026; Wayve, 2025; Agarwal et al., 2025), are emerging as a key technology to advance autonomous driving. By enabling realistic closed-loop interaction with simulated environments, these models create new opportunities for post-training of planners beyond the fixed trajectories available in logged datasets (Li et al., 2026a). However, scaling closed-loop RL in generative simulators remains challenging due to the significant computational cost of sensor simulation. Rendering observations at each environment step often requires generating high-resolution multi-camera video frames, making rollout generation a major bottleneck and large-scale policy training prohibitively slow.

To accommodate the high cost of closed-loop generative simulation, modern RL training systems parallelize rollout generation across multiple GPUs by distributing scenes, scenarios, or policy rollouts across workers (Espeholt et al., 2018; Liang et al., 2018; Shi et al., 2026). While these strategies allow to scale with the available compute, they do not address a fundamental bottleneck within each rollout. Indeed, simulator and planner executions remain *inherently sequential and synchronous*: the simulator waits for each planner action before rendering the next observation, and the planner waits for that observation before predicting the next action. As illustrated in Figure 1(a), this dependency leaves both components idle, leading to poor GPU utilization. Because these stalls occur within each rollout, conventional data-parallel scaling cannot eliminate them.

To overcome this bottleneck, we introduce *Speculative Rendering* (SpecRender), a novel rollout generation paradigm that breaks the strict sequential dependency between simulator and planner. Instead of waiting for planner predictions at every timestep, Speculative Rendering employs a lightweight draft model to speculate future planner states ahead of time. This design is motivated by the temporal continuity of driving behavior, which makes future actions locally predictable over

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

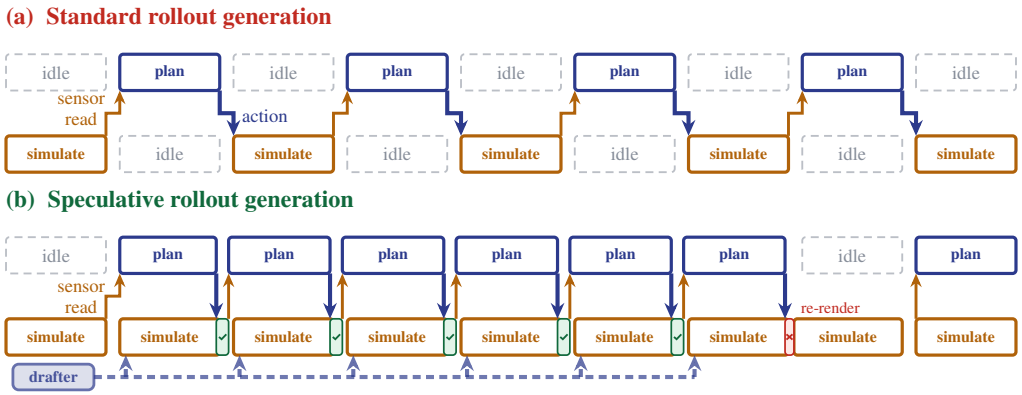


Figure 1: *Reducing idle time with speculative rendering.* (a) Standard rollout generation alternates simulation and planning, leaving each component idle while the other executes. (b) The drafter uses future ego poses predicted by the planner to prepare observations during planner inference. The simulator verifies each observation at its target step, returning accepted observations from the cache and rendering rejected drafts from the realized state (eq. 10).

short horizons. The future states predicted by the drafter are then proactively rendered and buffered in parallel, allowing simulation to proceed without blocking on planner execution. By relaxing the simulator-planner dependency, Speculative Rendering increases GPU utilization for both rendering and policy inference, substantially reducing rollout generation time and improving the scalability of closed-loop RL training as shown in Figure 1(b).

Inspired by speculative decoding for language tokens (Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2023) and policy tokens (Wang et al., 2025; Zheng et al., 2026a), Speculative Rendering uses a draft-and-verification mechanism to render future observations in advance. Specifically, our drafter constructs future simulator states from ego trajectory already predicted by the planner in previous timesteps. At each step, a verifier then uses a kinematics-based trajectory distance (Zheng et al., 2026a) to compare the drafted and target states from the planner. When this distance falls within an acceptance threshold, the corresponding cached observation is used. Otherwise, the affected drafts are invalidated and the simulator renders a new observation from the clean target state. The acceptance threshold controls the trade-off between rollout fidelity and acceleration: relaxing the threshold allows more speculative observations to be used and reduces rendering stalls, but permits larger discrepancies between drafted and target states.

We evaluate speculative rendering in World Engine (Li et al., 2026a) and show that mean simulation-step latency decreases from 364 to 138 ms. Aggregate rollout-generation cost decreases from 840 to 488 H100 GPU-hours, a 42% reduction. GRPO post-training (Shao et al., 2024) on speculative rollouts improves PDMS by 12.66% on `navtest-failure`, within 1% of post-training with standard rendering. Matched rollouts further show similar reward distributions and closely aligned trajectories with and without speculative rendering.

2 Related Work

Generative driving simulators Data-driven simulators convert recorded sensor logs into interactive environments that support novel ego trajectories and modified scene configurations. Early systems used image-based synthesis and neural fields to generate camera and LiDAR observations beyond the logged trajectory (Amini et al., 2022; Yang et al., 2023; Tonderski et al., 2024). Neural fields subsequently scaled novel-view synthesis to large and dynamic urban environments, while instance-aware decompositions enabled controllable sensor simulation (Rematas et al., 2022; Tancik et al., 2022; Turki et al., 2023; Wu et al., 2023; Yang et al., 2024). The introduction of 3D Gaussian Splatting led to explicit scene representations with substantially higher rendering rates. Subsequent methods decompose urban scenes into static backgrounds and dynamic actors, extend object-centric reconstruction to diverse traffic participants, improve rendering away from the recorded trajectory, and replace per-scene optimization with feed-forward reconstruction (Kerbl et al., 2023; Yan et al.,

2024; Zhou et al., 2024; Khan et al., 2025; Chen et al., 2025; Zhou et al., 2025a; Ni et al., 2025a; NVIDIA, 2026; NVIDIA et al., 2025). A complementary line uses learned generative models to synthesize controllable multi-view sensor observations, forecast action-conditioned futures, model occupancy evolution, and produce diverse or long-horizon driving scenarios (Kim et al., 2021; Wang et al., 2024c; Gao et al., 2024a; Wen et al., 2024; Li et al., 2024; Lu et al., 2024; Wang et al., 2024a; Zhao et al., 2025; Zheng et al., 2024; Guo et al., 2024; Ni et al., 2025b; Wang et al., 2024b; Hu et al., 2023; Russell et al., 2025; Wayve, 2025; Gao et al., 2024b; Rowe et al., 2025; Waabi, 2023; NVIDIA et al., 2026; Li et al., 2026a). These methods improve scene coverage and reconstruction fidelity, or reduce the cost of rendering an individual observation. However, each observation is still rendered after the corresponding simulator state is reached, which places every render on the sequential rollout path. Our method instead renders candidate observations ahead of time and reuses them after verification, moving accepted renders off the critical path while remaining agnostic to the scene representation and renderer implementation.

Speculative inference for vision-language-action models Recent speculative methods reduce the inference latency of VLA policies. They draft action tokens with smaller policies, kinematic predictors, retrieval, or diffusion models, then apply relaxed, state-aware, or kinematic verification (Wang et al., 2025; Zheng et al., 2026a;b; Wen et al., 2026; Niu et al., 2026). Related methods verify long-horizon plans against current observations or sampled action trajectories to determine when replanning is required (Wang et al., 2026b; Chen et al., 2026). Speculative computation has also been applied to token pruning, autoregressive reasoning, block-diffusion decoding, visual context reuse, and specialized hardware (Wang et al., 2026a; Dinh et al., 2026; Li et al., 2026b; Zhang et al., 2026; Qi et al., 2026). These methods speculate within the policy to reduce the cost of producing actions. Speculative rendering instead anticipates future states and observations, impacting the overall simulation cost.

Draft construction and verification The effectiveness of speculation depends on both proposal accuracy and the acceptance rule. Existing rules use fixed action-space tolerances, kinematic adaptation, scene context, agreement among sampled trajectories, or a budget over accumulated local regret (Wang et al., 2025; Zheng et al., 2026a; Wen et al., 2026; Chen et al., 2026; Feng et al., 2026). We evaluate corresponding fixed, state-dependent, kinematic, and regret-based verification rules. We also compare the policy trajectory with training-free constant-velocity and constant-turn-rate proposals, reuse of the preceding plan, and temporal consensus over recent plans (Schöller et al., 2020; Schubert et al., 2008). These alternatives isolate whether a separate motion prior improves speculation over using the VLA trajectory directly. Unlike action-token verification, our verifier compares a proposed ego state with the state realized by the simulator and determines whether its previously rendered observation remains valid.

3 Speculative Rendering

This section presents speculative rendering for closed-loop simulation, where we focus on a vision-language-action (VLA) model as the planner. We first define the simulator state, rendering process, and planner trajectory. We then describe how the drafter combines future ego poses with reactive actor motion to construct and render future scene configurations. Finally, we introduce the verifier that compares drafted and realized states to determine whether a cached observation can be reused. An overview of Speculative Rendering is shown in Figure 2.

3.1 Closed-Loop VLA Simulation

Scene representation and state. A generative simulator represents each driving scene with 3D Gaussian Splatting, separating the static background from dynamic actors (Li et al., 2026a). Background primitives remain in world coordinates. Actor primitives are transformed from object-centric coordinates using their simulated poses. The simulator updates the ego and actor poses at intervals of $\Delta = 0.5\,\mathrm{s}$. These poses define the scene configuration

$$s_t = \left(x_t^{\mathrm{e}}, \{x_t^i\}_{i \in \mathcal{A}_t}\right), \qquad x_t^{\mathrm{e}}, x_t^i \in \mathcal{X}, \tag{1}$$

where $\mathcal{X} = \mathbb{R}^2 \times \mathbb{S}^1$ and $\mathcal{A}_t$ contains the non-ego actors present at step t, as determined by the log's validity masks. Each actor pose consists of a position and heading direction,

$$x_t^i = (p_t^i, \psi_t^i), \qquad p_t^i \in \mathbb{R}^2, \quad \psi_t^i \in [-\pi, \pi), \tag{2}$$

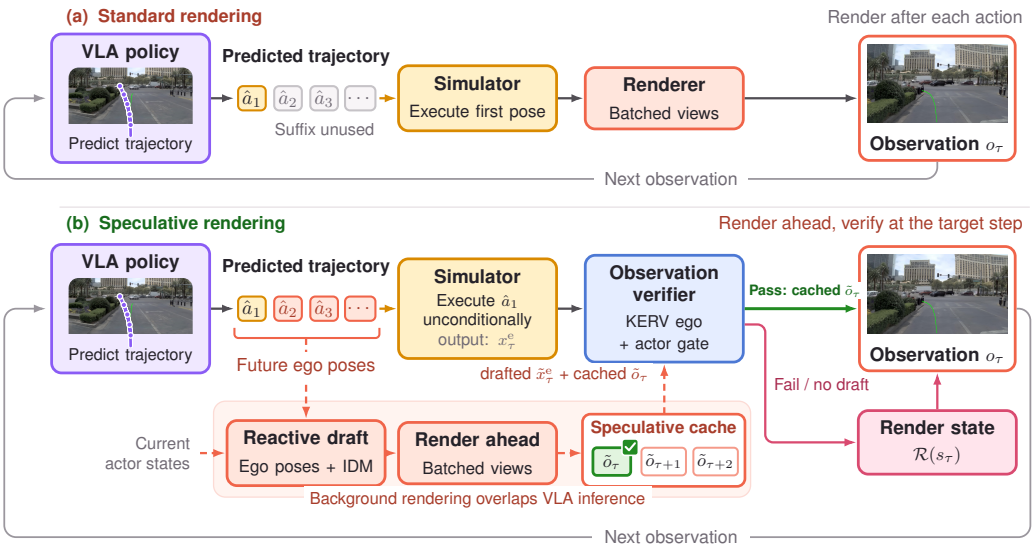


Figure 2: *Speculative rendering.* (a) Standard simulation executes the first predicted pose, advances the surrounding actors, and then renders the next observation. The trajectory drawn on the VLA input represents the policy prediction and is not part of the rendered observation. (b) Speculative rendering uses the remaining predicted poses to draft future scene configurations. These configurations are rendered during VLA inference and cached for their target steps (eq. 6). The simulator executes the first predicted pose before verification. At the target step, the drafted and target states are compared with a KERV ego check on position and heading (eq. 7) together with a position check on the nearby IDM actors (eq. 8). A draft that passes both checks supplies the cached observation; otherwise the simulator uses the target scene state to render the observations (eq. 10).

with the ego pose defined in the same way. The full simulator state z_t also includes velocities and navigation information.

Rendering observations. The renderer $\mathcal{R}$ places actor primitives and calibrated cameras according to s_t. It produces an observation containing C camera views,

$$o_t = \mathcal{R}(s_t) = \{I_t^k\}_{k=1}^{C}. \tag{3}$$

For a fixed scene representation and deterministic rendering settings, identical configurations produce identical observations,

$$s_t = s_{t'} \quad \Longrightarrow \quad \mathcal{R}(s_t) = \mathcal{R}(s_{t'}). \tag{4}$$

An observation can therefore be rendered before the simulator reaches that configuration if the required poses are known. We use predicted states to render camera frames in advance and verify them against the target states before use.

VLA prediction and reactive dynamics. The VLA policy takes the latest H observations, ego dynamic state u_t, and navigation command c_t as input. It predicts N future poses in the current ego frame,

$$\hat{\boldsymbol{a}}_t = (\hat{a}_{t,1}, \ldots, \hat{a}_{t,N}) = f_\theta(o_{t-H+1:t}, u_t, c_t), \qquad \hat{a}_{t,j} = (\hat{p}_{t,j}, \hat{\psi}_{t,j}) \in \mathcal{X}. \tag{5}$$

The poses are separated by a fixed time interval (e.g. 0.5s), and u_t contains ego velocity and acceleration. In the first simulation step, the simulator executes the first pose and updates surrounding actors using the Intelligent Driver Model (IDM) (Treiber et al., 2000) and supplies the resulting observation to the policy. The remaining poses predict future ego motion and provide the lookahead for speculative rendering.

3.2 Reactive Drafting and Observation Reuse

Reactive scene prediction. We draft observations for a lookahead of $K \leq N$ steps. First, we transform each predicted ego pose into world coordinates as $\tilde{x}^{e,(t)}_{t+j} = x^e_t \circ \hat{a}_{t,j}$, where $\circ$ denotes pose

composition. Starting from the actor states in z_t, we then roll out IDM conditioned on the predicted ego motion. This rollout updates actor accelerations, velocities, and positions at each step. The predicted ego and actor poses define the draft configuration $\tilde{s}^{(t)}_{t+j}$, which we render to obtain

$$\tilde{o}^{(t)}_{t+j} = \mathcal{R}\left(\tilde{s}^{(t)}_{t+j}\right), \qquad j \in \{1, \dots, K\}. \tag{6}$$

The subscript $t+j$ denotes the target simulation step, and the superscript (t) identifies the simulation timestep at which the drafted observation was produced. At depth one, the draft uses the ego pose selected for immediate execution. Later drafts use the remaining VLA predicted poses, which may differ from those selected when the policy updates its plan.

Asynchronous rendering and caching. Draft rendering runs on a background worker in parallel with VLA inference. The camera views for each drafted configuration are rendered as a batch. The cache holds one draft per target step, with up to K drafts for the steps $\tau+1, \dots, \tau+K$ ahead of the simulator. Each new trajectory produces drafts for the next K steps. At overlapping target steps, these drafts replace earlier estimates with predictions conditioned on the latest observations.

At step τ, the simulator checks whether the cached draft $\tilde{o}_\tau$ has finished rendering and passes verification. If both conditions hold, it returns the draft. Otherwise, it renders the target configuration $\mathcal{R}(s_\tau)$. Drafts that finish after the simulator requests the observation are discarded, as are drafts for steps the simulator has already passed.

Kinematic verification. Replanning can change the ego trajectory and the resulting actor motion. We therefore check both the ego and the reactive actors before reusing a draft. For the ego, we adapt the kinematic verification rule of KERV (Zheng et al., 2026a). Let $(p^{\mathrm{e}}_\tau, \psi^{\mathrm{e}}_\tau)$ denote the realized ego position and heading, and $(\tilde{p}^{\mathrm{e}}_\tau, \tilde{\psi}^{\mathrm{e}}_\tau)$ the drafted values. The ego check requires both position and heading errors to remain within their tolerances,

$$\|p^{\mathrm{e}}_\tau - \tilde{p}^{\mathrm{e}}_\tau\|_2 \le \epsilon_{\mathrm{pos}} \quad \text{and} \quad |\angle(\psi^{\mathrm{e}}_\tau - \tilde{\psi}^{\mathrm{e}}_\tau)| \le \epsilon_{\mathrm{yaw}}, \tag{7}$$

where $\angle(\cdot)$ wraps the heading difference to $[-\pi, \pi)$.

For the actor check, $\mathcal{N}_\tau$ contains the drafted actors within the IDM radius of the drafted ego. These are the actors whose motion is advanced by IDM. Every actor in this set must remain within the position tolerance,

$$\max_{i \in \mathcal{N}_\tau} \|p^i_\tau - \tilde{p}^i_\tau\|_2 \le \epsilon_{\mathrm{act}}. \tag{8}$$

The verification decision combines the ego and actor checks,

$$v_\tau = \mathbb{1}[\|p^{\mathrm{e}}_\tau - \tilde{p}^{\mathrm{e}}_\tau\|_2 \le \epsilon_{\mathrm{pos}}]\, \mathbb{1}\left[|\angle(\psi^{\mathrm{e}}_\tau - \tilde{\psi}^{\mathrm{e}}_\tau)| \le \epsilon_{\mathrm{yaw}}\right] \prod_{i \in \mathcal{N}_\tau} \mathbb{1}\left[\|p^i_\tau - \tilde{p}^i_\tau\|_2 \le \epsilon_{\mathrm{act}}\right]. \tag{9}$$

Let $c_\tau \in \{0, 1\}$ indicate whether rendering of the draft for step τ has completed. The simulator applies $\hat{a}_{\tau-1,1}$ before verification. The decision determines which observation is returned at step τ,

$$o_\tau = \begin{cases} \tilde{o}_\tau, & c_\tau v_\tau = 1, \\ \mathcal{R}(s_\tau), & c_\tau v_\tau = 0. \end{cases} \tag{10}$$

Checking reactive actors in addition to the ego constrains position errors in the surrounding traffic, but can reduce the acceptance rate. Actors beyond the IDM radius advance at their snapshot speed and are excluded from verification. An accepted draft can therefore retain differences in distant traffic. The policy receives an observation and predicts a new trajectory at every step, whether the draft is accepted or the realized state is rendered. Since observation reuse is approximate, it can change subsequent policy predictions. We therefore evaluate both rollout-generation time and post-training driving performance.

4 Experiments

We evaluate speculative rendering for closed-loop VLA rollout generation, measuring both generation cost and downstream driving performance. We first study the predictive horizon available for drafting and the runtime implications of verifier choice. We then compare SpecRender with standard rendering and matched-cost alternatives, evaluate the resulting post-training signal, and inspect matched trajectories to characterize the effect of speculated rollout trajectories.

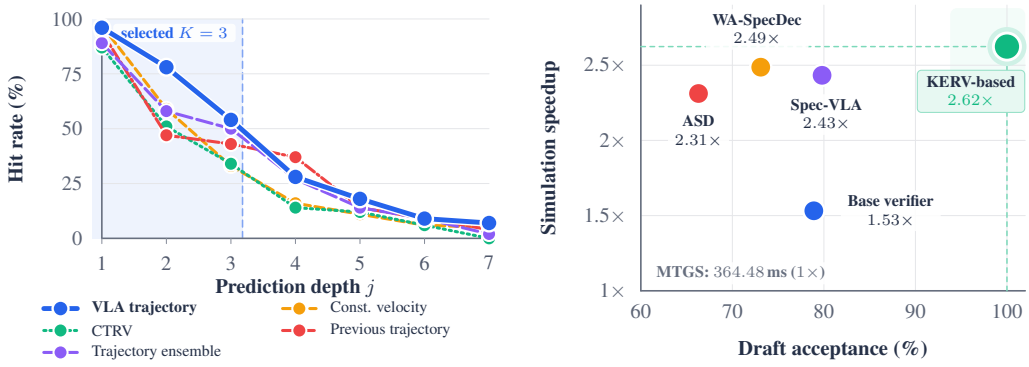
4.1 Experimental Setup

Simulation setup. We conduct closed-loop experiments in World Engine (Li et al., 2026a) on NAVSIM-v1 scenes with fitted multi-traversal Gaussian-splatting reconstructions. The driving policy is AutoVLA (Zhou et al., 2025b), built on Qwen2.5-VL-3B-Instruct (Bai et al., 2025), which predicts ten poses over five seconds from multi-camera history, ego dynamics, and navigation. All experiments use reactive IDM traffic (Treiber et al., 2000). Unless specified otherwise, SpecRender uses $K = 3$ with the KERV-based verifier defined in eq. 7, eq. 8, and eq. 9.

We use reactive simulation in all experiments. Surrounding actors follow IDM (Treiber et al., 2000), and their motion during drafting is rolled out conditioned on the VLA trajectory. Unless specified otherwise, SpecRender uses a lookahead of $K = 3$ and the KERV-based verifier defined in eq. 7, eq. 8, and eq. 9. The ego-position threshold has a base value of 0.5 m and is scaled by the drafted ego speed relative to 5 m/s, with the scale clipped to $[0.5, 2.0]$. Thus, ϵ_{pos} varies from 0.125 to 0.5 m. We disable heading verification by setting $\epsilon_{\text{yaw}} = \infty$, use an actor-position threshold of $\epsilon_{\text{act}} = 0.5$ m, and verify reactive actors within 40 m of the drafted ego.

Post-training and evaluation. We first evaluate the supervised AutoVLA checkpoint on NAVSIM-v1 and identify scenes in which the policy collides or leaves the drivable area. From scenes with valid reconstructions, we retain $1{,}087$ examples spanning these failure cases and clean scenes. We generate 16 closed-loop candidates per scene, each containing 20 frames over 12 seconds, and score them with the World Engine PDM reward. GRPO (Shao et al., 2024) computes each candidate's advantage relative to the mean reward of its scene group. We compare standard rendering, SpecRender, and matched-cost alternatives while holding the VLA checkpoint, camera configuration, policy inputs, and reactive simulation fixed. Checkpoints are selected using a self-normalized importance-sampling estimate on held-out scenes and evaluated by PDMS under reactive simulation on `navtest-failure`. Additional details are in subsection A.1.

4.2 Experimental Results



(a) Drafter hit rate.

(b) Verifier trade-off.

Figure 3: *Draft agreement and verifier runtime.* (a) Hit rates across prediction depths, with shading marking the selected lookahead $K = 3$. Depth one uses the pose selected for immediate execution, while later depths use the unexecuted trajectory suffix. (b) Draft acceptance and simulation speedup for the VLA drafter at $K = 3$, relative to the 364.48 ms MTGS reference used in this benchmark. Method names denote adapted verification rules. These measurements characterize acceptance and runtime; post-training driving performance is reported in Table 1.

The VLA trajectory supports reuse beyond the immediate step. We compare the VLA trajectory drafter several baselines including constant-velocity (Schöller et al., 2020) and constant-turn-rate-and-velocity (CTRV) (Schubert et al., 2008) extrapolation, the preceding trajectory, and a temporal trajectory ensemble. The preceding trajectory is transformed into the current ego frame and shifted forward by one step. The ensemble aligns recent predictions by target step and takes their pose-wise median. Definitions and details for all drafter and verifier variants are provided in subsection A.2.

Table 1: *Closed-loop driving performance and rollout-generation.* PDMS is reported on World Engine's `navtest-failure` split. SR is the closed-loop success rate, and EP is ego progress. Rollout generation is aggregate GPU-hours across eight H100 GPUs for reactive MTGS simulation with IDM actors. Short-horizon rollouts use eight frames over six seconds (Li et al., 2026a), while Speculative Rendering uses 20 frames over 12 seconds.

METHOD	RLFT	PDMS ↑	SR ↑	EP ↑	ROLLOUT GPU-H ↓
VADv2 (Li et al., 2026a)	✗	62.74	74.51	47.59	–
VADv2 (Li et al., 2026a)	✓	67.56	73.35	51.86	1,072
AutoVLA (Zhou et al., 2025b)	✗	58.67	63.93	54.03	–
AutoVLA	✓	**72.13**	75.96	68.39	840
Hard-negative subset	✓	67.92	71.33	56.10	496
Fewer rollouts	✓	63.09	68.54	50.50	504
Short horizon (Li et al., 2026a)	✓	64.96	68.44	51.01	489
AutoVLA + SpecRender	✓	71.33	**76.01**	**69.31**	**488**

Under the KERV-based verifier used by SpecRender, the VLA trajectory achieves hit rates of 96%, 78%, and 54% at depths one through three, respectively (Figure 3a). Their unweighted mean is 76%. The distinction between these depths matters because depth one corresponds to the pose selected for immediate execution, while depths two and three extend beyond that pose and must remain accurate across subsequent replanning. Constant velocity slightly exceeds the VLA at depth one (97% *vs.* 96%), but the VLA leads every alternative at depths two and three. Thus, its strongest advantage appears where drafting depends on anticipating subsequent motion rather than reproducing the immediate step. Conditioning on the latest camera history and navigation command offers a plausible explanation for this agreement, without requiring an additional learned drafter.

The advantage does not extend uniformly to longer predictions. The preceding trajectory has a higher hit rate at depth four, and every drafter falls below 20% by depth five. Rendering farther ahead provides more time for a draft to complete, but subsequent replanning can invalidate its predicted state. We therefore select $K = 3$, which covers the range with the strongest agreement before prediction quality drops sharply.

Acceptance rate is not a proxy for end-to-end speed. With the VLA trajectory fixed as the drafter, we compare exact pose matching with fixed-margin, world-aware, regret-based, and kinematics-adaptive rules based on Spec-VLA (Wang et al., 2025), WA-SpecDec (Wen et al., 2026), ASD (Feng et al., 2026), and KERV (Zheng et al., 2026a), respectively. Definitions and implementation details are provided in subsection A.2. In Figure 3b, the base pose-matching rule accepts more drafts than the regret-based rule (78.9% versus 66.3%) yet is substantially slower (1.53× versus 2.31×). Acceptance omits verification overhead and draft readiness, both of which determine whether a cached render is available when the simulator requests it. The KERV-based rule accepts all evaluated drafts and reaches the highest speedup of 2.62×, reducing mean simulation-step latency from 364.48 to approximately 139 ms, with a component-level runtime breakdown provided in Figure 6. This result shows that its speed-adaptive tolerance converts the available drafts into reusable observations more effectively than the alternative verification rules. We therefore use KERV as the verifier for SpecRender. At rollout scale, generation time falls from 105 to 61 wall-clock hours, or from 840 to 488 H100 GPU-hours. The smaller end-to-end gain (1.72×) is expected because VLA inference, simulator updates, and data handling remain unchanged.

Speculative rendering preserves the post-training signal at lower cost. SpecRender reduces rollout-generation compute by 42% while being within 0.80% PDMS of standard-rendering post-training as shown in Table 1. More importantly, it outperforms all matched-budget alternatives. SpecRender retains the complete set of 1,087 scenes, all 16 candidates per scene, and 20 frames of closed-loop interaction. The hard-negative subset concentrates rollout generation on AutoVLA failure scenes, while the other alternatives reduce the number of candidates or use the short-horizon rollout. SpecRender instead preserves scene diversity, candidate diversity, and temporal coverage within the same generation budget.

Verification converts accurate drafts into reusable policy inputs. We next examine verification decisions during closed-loop rollout generation. For each lookahead depth, Figure 4 compares the

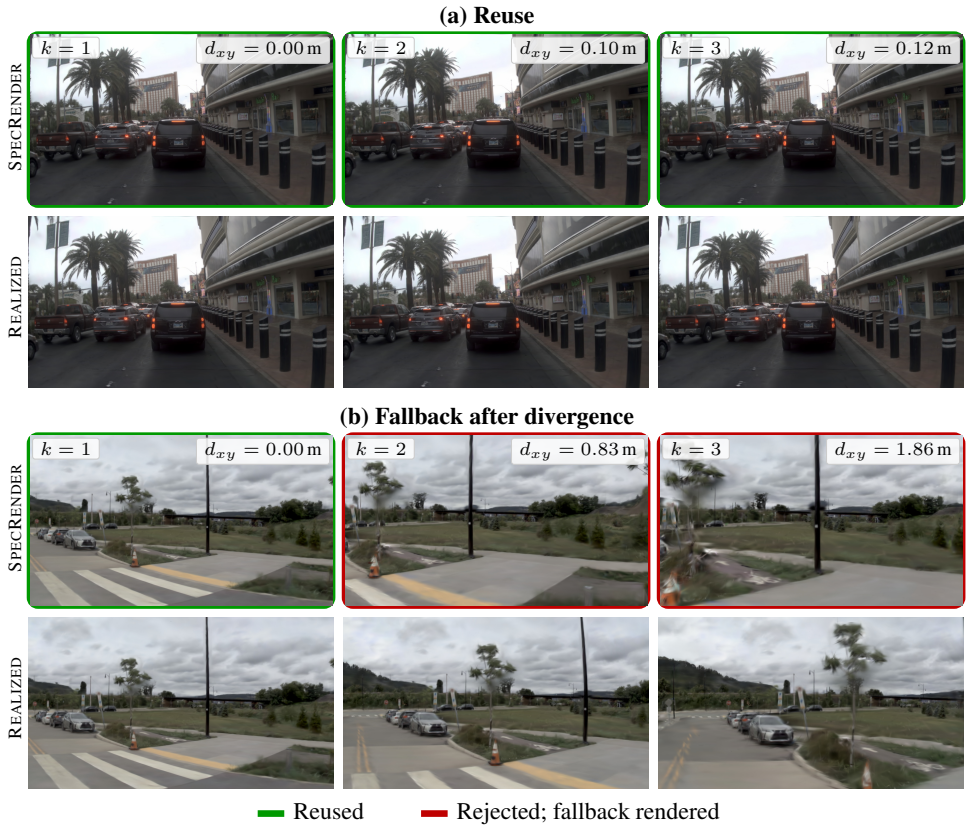
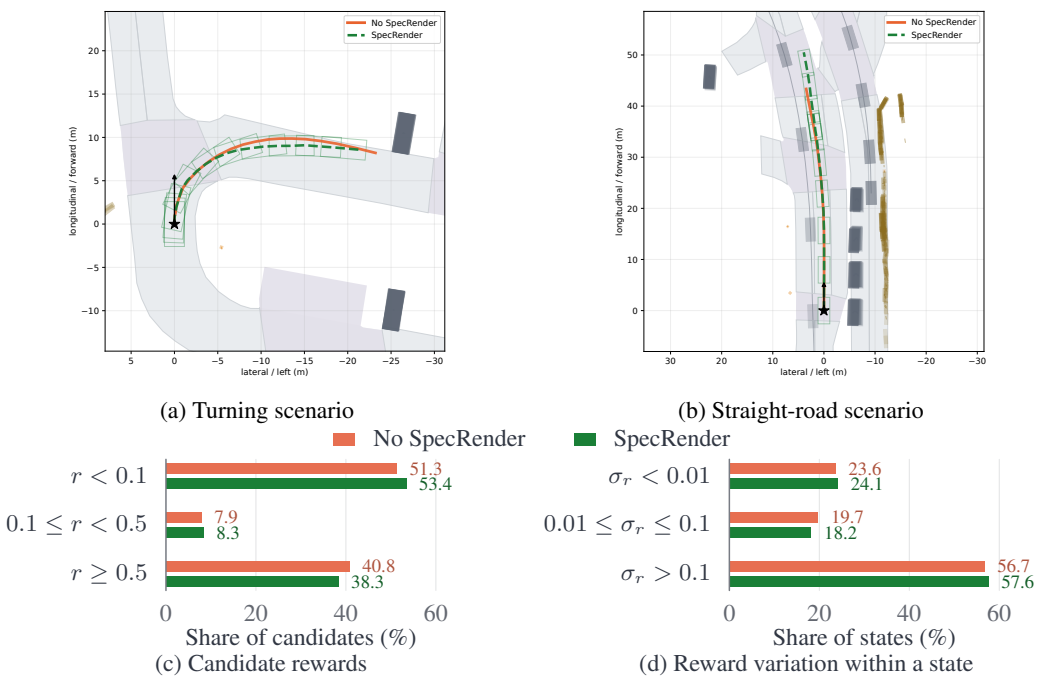


Figure 4: *Speculative rendering reuses accurate drafts and falls back after divergence.* Top rows show observations rendered in advance from one predicted trajectory; bottom rows show the corresponding policy inputs. In (a), all drafts are reused. In (b), only $k = 1$ is reused; the later drafts exceed the fixed $0.25\,\mathrm{m}$ tolerance and trigger realized fallback renders. Pose errors are inset on the speculative images. Borders encode the state-based reuse decision, not image similarity.

state used to render a draft frame with the realized simulator state when the corresponding policy input is required. SpecRender preserves visual fidelity because both drafted and regular frames are produced by the same World Engine renderer with identical rendering settings. It changes only when a frame is rendered and whether that frame is reused. In Figure 4 (a), the pose disagreement remains within tolerance at all three depths. The cached frames are therefore reused as policy inputs without additional rendering. In Figure 4 (b), the first draft frame is reused, but replanning produces larger lateral disagreement at depths two and three. The verifier rejects these frames, and the simulator re-renders each policy input at its realized state. Accurate predictions therefore avoid repeated rendering, while verification keeps the reused frames consistent with the executed path.

Speculative rendering accelerates rollout generation without inducing reward shift. The efficiency of speculative rendering is only useful if the generated rollouts still reflect the policy under training. A rendering strategy that reused stale views too aggressively could gradually steer the ego away from its own behavior, producing off-policy data that no longer supports faithful optimization. Figure 5 tests this directly by comparing matched reactive rollouts generated with and without SpecRender from an identical initial state and simulator configuration. The executed trajectories remain closely aligned throughout each episode, in both the predictable straight-road scenario and the scenario. This alignment indicates that verification prevents accepted drafts from displacing the rollout, so the accelerated data remains on-policy rather than drifting toward a different behavior.

Staying on-policy is necessary but not sufficient for group-relative post-training, which learns from how candidate rewards differ within each scene rather than from their absolute values. If speculative rendering compressed this within-scene spread, the resulting advantages would weaken even when the mean reward and executed trajectory were preserved. The reward panels show that this does not occur. Both the distribution of candidate rewards and the within-state reward variation are closely



(a) Turning scenario (b) Straight-road scenario

(c) Candidate rewards (d) Reward variation within a state

Figure 5: *Trajectory and reward consistency with speculative rendering.* (a,b) The highest-reward trajectories from matched reactive rollouts closely agree with and without SpecRender. The predictable straight-road motion is particularly favorable for successful drafting and reuse. (c,d) SpecRender also preserves the candidate-reward distribution and within-state reward variation.

matched between the two conditions, so the contrast between stronger and weaker candidates that drives each GRPO update is retained. Taken together, the trajectory and reward evidence explains why the accelerated rollouts support post-training as effectively as standard rendering. Speculative rendering reduces when and how often frames are produced without altering which behaviors the policy explores or how their relative quality is scored.

Limitations and scope. While Speculative Rendering substantially accelerates rollout generation and preserves the post-training signal, we expect its benefits to extend further with more stable long-horizon trajectory prediction. Abrupt maneuvers and substantial replanning reduce agreement at later depths, causing more fallback renders and reducing speedup. The verifier measures simulator-state agreement rather than pixel-level agreement, so small pose differences near occlusions and ungated distant actors can produce visual differences in reused frames. Tighter state tolerances or image-space verification could reduce these differences at the cost of reuse rate or verification compute. A promising future direction is adaptive speculation that selects the lookahead depth and rendering priority from planner confidence and scene dynamics. This would concentrate generation compute on drafts most likely to finish on time and pass verification, extending the efficiency gains across diverse driving scenarios without changing the planner or renderer.

5 Conclusion

We introduced SpecRender, which renders future VLA predictions concurrently with policy inference and reuses them after state-based verification. It preserves visual fidelity by using the same World Engine renderer for drafted and regular frames. SpecRender reduces simulation-step latency from 364.48 to approximately 139 ms and rollout collection from 840 to 488 H100 GPU-hours, a 42% reduction. Post-training improves AutoVLA from 58.67 to 71.33 PDMS, an absolute improvement of 12.66%. Matched rollouts remain closely aligned with standard rendering. Their reward distributions and within-state variation are also preserved. At a fixed compute budget, SpecRender retains the full scene set and all candidates per scene. It also supports a longer rollout horizon that captures delayed driving outcomes. SpecRender therefore accelerates closed-loop rollout collection without reducing training-data coverage or quality.

REFERENCES

Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025. 1

Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. VISTA 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. In *IEEE International Conference on Robotics and Automation*, 2022. 2

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025. 6

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023. 2

Feng Chen, Xianghui Wang, Yuxuan Chen, Boying Li, Yefei He, Zeyu Zhang, and Yicheng Wu. Dynamic execution commitment of vision-language-action models. *arXiv preprint arXiv:2605.11567*, 2026. 3

Ziyu Chen, Jiawei Yang, Jiahui Huang, Riccardo de Lutio, Janick Martinez Esturo, Boris Ivanovic, Or Litany, Zan Gojcic, Sanja Fidler, Marco Pavone, Li Song, and Yue Wang. OmniRe: Omni urban scene reconstruction. In *International Conference on Learning Representations*, 2025. 3

Anh Dung Dinh, Simon Khan, and Flora Salim. Reasoning-aware speculative decoding for efficient vision-language-action models in autonomous driving. *arXiv preprint arXiv:2606.31160*, 2026. 3

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1407–1416, 2018. 1

Yuannuo Feng, Zegang Peng, Yuxin Xie, Yubing Ye, Yizhe Chen, Wenshuai Yao, Wenyong Zhou, and Wang Kang. Approximate speculative decoding. *arXiv preprint arXiv:2608.03447*, 2026. 3, 7

Ruiyuan Gao, Kai Chen, Enze Xie, Lanqing Hong, Zhenguo Li, Dit-Yan Yeung, and Qiang Xu. MagicDrive: Street view generation with diverse 3D geometry control. In *International Conference on Learning Representations*, 2024a. 3

Shenyuan Gao, Jiazhi Yang, Li Chen, Kashyap Chitta, Yihang Qiu, Andreas Geiger, Jun Zhang, and Hongyang Li. Vista: A generalizable driving world model with high fidelity and versatile controllability. In *Advances in Neural Information Processing Systems*, 2024b. 3

Xi Guo, Chenjing Ding, Haoxuan Dou, Xin Zhang, Weixuan Tang, and Wei Wu. InfinityDrive: Breaking time limits in driving world models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshop*, 2024. 3

Anthony Hu, Lloyd Russell, Hudson Yeo, Zak Murez, George Fedoseev, Alex Kendall, Jamie Shotton, and Gianluca Corrado. GAIA-1: A generative world model for autonomous driving. *arXiv preprint arXiv:2309.17080*, 2023. 3

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 2023. 2

Mustafa Khan, Hamidreza Fazlali, Dhruv Sharma, Tongtong Cao, Dongfeng Bai, Yuan Ren, and Bingbing Liu. AutoSplat: Constrained gaussian splatting for autonomous driving scene reconstruction. *IEEE International Conference on Robotics and Automation*, 2025. 3

Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. DriveGAN: Towards a controllable high-quality neural simulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. 3

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, 2023. 2

Tianyu Li, Li Chen, Caojun Wang, Haochen Liu, Kashyap Chitta, Zhenjie Yang, Yuhang Lu, Naisheng Ye, Yihang Qiu, Yufei Wang, et al. World engine: Towards the era of post-training for autonomous driving. *arXiv preprint arXiv:2606.19836*, 2026a. 1, 2, 3, 6, 7

Xiaofan Li, Yifu Zhang, and Xiaoqing Ye. Drivingdiffusion: Layout-guided multi-view driving scene video generation with latent diffusion model. *European Conference on Computer Vision*, 2024. 3

Zekai Li, Yihao Liang, Hongfei Zhang, Jian Chen, Yesheng Liang, and Zhijian Liu. FlashDrive: Flash vision-language-action inference for autonomous driving. *Conference on Robot Learning*, 2026b. 3

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, 2018. 1

Jiachen Lu, Ze Huang, Zeyu Yang, Jiahui Zhang, and Li Zhang. WoVoGen: World volume-aware diffusion for controllable multi-camera driving scene generation. In *European Conference on Computer Vision*, 2024. 3

Chaojun Ni, Guosheng Zhao, Xiaofeng Wang, Zheng Zhu, Wenkang Qin, Guan Huang, Chen Liu, Yuyin Chen, Yida Wang, Xueyang Zhang, Yifei Zhan, Kun Zhan, Peng Jia, Xianpeng Lang, Xingang Wang, and Wenjun Mei. ReconDreamer: Crafting world models for driving scene reconstruction via online restoration. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025a. 3

Jingcheng Ni, Yuxin Guo, Yichen Liu, Rui Chen, Lewei Lu, and Zehuan Wu. MaskGWM: A generalizable driving world model with video mask reconstruction. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025b. 3

Jiahui Niu, Kefan Gu, Yucheng Zhao, Shengwen Liang, Tiancai Wang, Xing Hu, Ying Wang, and Huawei Li. Realtime-VLA FLASH: Speculative inference framework for diffusion-based VLAs. *arXiv preprint arXiv:2605.13778*, 2026. 3

NVIDIA. Instant nurec: Feed-forward 3d gaussian reconstruction for driving scene simulation. Technical report, NVIDIA, 2026. URL https://arxiv.org/abs/2607.14203. 3

NVIDIA, Yulong Cao, Riccardo de Lutio, Sanja Fidler, Guillermo Garcia Cobo, Zan Gojcic, Maximilian Igl, Boris Ivanovic, Peter Karkus, Janick Martinez Esturo, Marco Pavone, Aaron Smith, Ellie Tanimura, Michal Tyszkiewicz, Michael Watson, Qi Wu, and Le Zhang. Alpasim: A modular, lightweight, and data-driven research simulator for autonomous driving, October 2025. URL https://github.com/NVlabs/alpasim. 1, 3

NVIDIA, Aarti Basant, Amlan Kar, Despoina Paschalidou, Fangyin Wei, Francesco Ferroni, Guillermo Garcia Cobo, Haithem Turki, Huan Ling, Jaewoo Seo, James Lucas, Jay Zhangjie Wu, Jialiang Wang, Jonathan Lorraine, Jun Gao, Kai He, Katarina Tothova, Kevin Xie, Michał Tyszkiewicz, Qi Wu, et al. NVIDIA OmniDreams: Real-time generative world model for closed-loop autonomous vehicle simulation. *arXiv preprint arXiv:2606.03159*, 2026. 1, 3

Chunyu Qi, Zhuoran Song, Jian Weng, Haozhe Jiang, Xueyuan Liu, Naifeng Jing, Guanghui He, Xiaoyao Liang, and Haibing Guan. Algorithm-architecture co-design for efficient VLA inference via speculative inference and verification. *arXiv preprint arXiv:2608.15636*, 2026. 3

Konstantinos Rematas, Andrew Liu, Pratul P. Srinivasan, Jonathan T. Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban radiance fields. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 2

Luke Rowe, Roger Girgis, Anthony Gosselin, Liam Paull, Christopher Pal, and Felix Heide. Scenario dreamer: Vectorized latent diffusion for generating driving simulation environments. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. 3

Lloyd Russell, Anthony Hu, Lorenzo Bertoni, George Fedoseev, Jamie Shotton, Elahe Arani, and Gianluca Corrado. GAIA-2: A controllable multi-view generative world model for autonomous driving. *arXiv preprint arXiv:2503.20523*, 2025. 3

Christoph Schöller, Vincent Aravantinos, Florian Lay, and Alois Knoll. What the constant velocity model can teach us about pedestrian motion prediction. *IEEE Robotics and Automation Letters*, 2020. 3, 6, 1

Robin Schubert, Eric Richter, and Gerd Wanielik. Comparison and evaluation of advanced motion models for vehicle tracking. In *International Conference on Information Fusion*, 2008. 3, 6, 1

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 2, 6

Yunxiao Shi, Hong Cai, Mohammad Ghavamzadeh, and Fatih Porikli. Clear: Closed-loop reinforcement learning at scale for end-to-end autonomous driving. *arXiv preprint arXiv:2607.02841*, 2026. 1

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-NeRF: Scalable large scene neural view synthesis. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 2

Adam Tonderski, Carl Lindström, Georg Hess, William Ljungbergh, Lennart Svensson, and Christoffer Petersson. NeuRAD: Neural rendering for autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 1, 2

Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 2000. 4, 6, 1

Haithem Turki, Jason Y. Zhang, Francesco Ferroni, and Deva Ramanan. SUDS: Scalable urban dynamic scenes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. 2

Waabi. Waabi World: A closed-loop simulator powered by generative ai. Waabi, 2023. URL https://waabi.ai/waabi-world. 3

Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. SpecPrune-VLA: Accelerating vision-language-action models via action-aware self-speculative pruning. *International Conference on Machine Learning*, 2026a. 3

Songsheng Wang, Rucheng Yu, Zhihang Yuan, Chao Yu, Feng Gao, Yu Wang, and Derek F. Wong. Spec-VLA: Speculative decoding for vision-language-action models with relaxed acceptance. *Conference on Empirical Methods in Natural Language Processing*, 2025. 2, 3, 7

Xiaofeng Wang, Zheng Zhu, Guan Huang, Xinze Chen, Jiagang Zhu, and Jiwen Lu. DriveDreamer: Towards real-world-driven world models for autonomous driving. *European Conference on Computer Vision*, 2024a. 3

Yuqi Wang, Ke Cheng, Jiawei He, Qitai Wang, Hengchen Dai, Yuntao Chen, Fei Xia, and Zhaoxiang Zhang. DrivingDojo dataset: Advancing interactive and knowledge-enriched driving world model. In *Advances in Neural Information Processing Systems*, 2024b. 3

Yuqi Wang, Jiawei He, Lue Fan, Hongxin Li, Yuntao Chen, and Zhaoxiang Zhang. Driving into the future: Multiview visual forecasting and planning with world model for autonomous driving. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024c. 3

Zihua Wang, Zhitao Lin, Ruibo Li, Yu Zhang, Xu Yang, Siya Mi, and Xiu-Shen Wei. Open-loop planning, closed-loop verification: Speculative verification for VLA. *arXiv preprint arXiv:2604.02965*, 2026b. 3

Wayve. GAIA-3: A new era of evaluation for autonomous driving. Wayve Research, December 2025. URL https://wayve.ai/thinking/gaia-3/. 1, 3

Yuqing Wen, Yucheng Zhao, Yingfei Liu, Fan Jia, Yanhui Wang, Chong Luo, Chi Zhang, Tiancai Wang, Xiaoyan Sun, and Xiangyu Zhang. Panacea: Panoramic and controllable video generation for autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6902–6912, 2024. 3

Zikang Wen, Yuning Zhang, and Dong Yuan. WA-SpecDec: World-aware speculative decoding for vision-language-action models. *arXiv preprint arXiv:2608.08725*, 2026. 3, 7

Zirui Wu, Tianyu Liu, Liyi Luo, Zhide Zhong, Jianteng Chen, Hongmin Xiao, Chao Hou, Haozhe Lou, Yuantao Chen, Runyi Yang, et al. MARS: An instance-aware, modular and realistic simulator for autonomous driving. In *Chinese Conference on Artificial Intelligence*, 2023. 2

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023. 2

Yunzhi Yan, Haotong Lin, Chenxu Zhou, Weijie Wang, Haiyang Sun, Kun Zhan, Xianpeng Lang, Xiaowei Zhou, and Sida Peng. Street gaussians: Modeling dynamic urban scenes with gaussian splatting. In *European Conference on Computer Vision*, 2024. 2

Jiawei Yang, Boris Ivanovic, Or Litany, Xinshuo Weng, Seung Wook Kim, Boyi Li, Tong Che, Danfei Xu, Sanja Fidler, Marco Pavone, and Yue Wang. EmerNeRF: Emergent spatial-temporal scene decomposition via self-supervision. *International Conference on Learning Representations*, 2024. 2

Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. UniSim: A neural closed-loop sensor simulator. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. 1, 2

Kewei Zhang, Jin Wang, Sensen Gao, Chengyue Wu, Yulong Cao, Songyang Han, Boris Ivanovic, Langechuan Liu, Marco Pavone, Song Han, Daquan Zhou, and Enze Xie. Fast-dDrive: Efficient block-diffusion VLM for autonomous driving. *arXiv preprint arXiv:2605.23163*, 2026. 3

Guosheng Zhao, Xiaofeng Wang, Zheng Zhu, Xinze Chen, Guan Huang, Xiaoyi Bao, and Xingang Wang. DriveDreamer-2: LLM-enhanced world models for diverse driving video generation. *Association for the Advancement of Artificial Intelligence*, 2025. 3

Wenzhao Zheng, Weiliang Chen, Yuanhui Huang, Borui Zhang, Yueqi Duan, and Jiwen Lu. OccWorld: Learning a 3D occupancy world model for autonomous driving. *European Conference on Computer Vision*, 2024. 3

Zihao Zheng, Zhihao Mao, Maoliang Li, Jiayu Chen, Xinhao Sun, Zhaobo Zhang, Donggang Cao, Hong Mei, and Xiang Chen. KERV: Kinematic-rectified speculative decoding for embodied VLA models. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference*, 2026a. 2, 3, 5, 7

Zihao Zheng, Zhihao Mao, Sicheng Tian, Maoliang Li, Jiayu Chen, Xinhao Sun, Zhaobo Zhang, Xuanzhe Liu, Donggang Cao, Hong Mei, and Xiang Chen. HeiSD: Hybrid speculative decoding for embodied vision-language-action models with kinematic awareness. *arXiv preprint arXiv:2603.17573*, 2026b. 3

Jingqiu Zhou, Lue Fan, Linjiang Huang, Xiaoyu Shi, Si Liu, Zhaoxiang Zhang, and Hongsheng Li. FlexDrive: Toward trajectory flexibility in driving scene reconstruction and rendering. *arXiv preprint arXiv:2502.21093*, 2025a. 3

Xiaoyu Zhou, Zhiwei Lin, Xiaojun Shan, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. DrivingGaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 3

Zewei Zhou, Tianhui Cai, Seth Z. Zhao, Yun Zhang, Zhiyu Huang, Bolei Zhou, and Jiaqi Ma. AutoVLA: A vision-language-action model for end-to-end autonomous driving with adaptive reasoning and reinforcement fine-tuning. In *Advances in Neural Information Processing Systems*, 2025b. 6, 7, 1

A Additional Experimental Details

A.1 Experimental Protocol

Simulator and policy interface. We use NAVSIM-v1 scenes with fitted multi-traversal Gaussian-splatting reconstructions in World Engine (Li et al., 2026a). AutoVLA (Zhou et al., 2025b) receives rendered observations from the front, front-left, and front-right cameras together with ego velocity, acceleration, and a navigation command. Its visual input contains four frames sampled at 2 Hz. The policy predicts ten ego poses over five seconds. Our interface decodes this trajectory, passes its first pose to the simulator, and makes the remaining poses available for speculative rendering.

Reactive simulation and verification. Surrounding actors follow IDM (Treiber et al., 2000). During drafting, their states are rolled out conditioned on the predicted VLA trajectory. Unless specified otherwise, SpecRender uses $K = 3$ and the KERV-based verifier. The ego-position threshold has base value 0.25 m and scales with drafted ego speed relative to 5 m/s. Clipping the scale to $[0.5, 2.0]$ gives an effective threshold from 0.125 to 0.5 m. We set $\epsilon_{\text{yaw}} = \infty$, use $\epsilon_{\text{act}} = 0.5$ m, and verify reactive actors within 40 m of the drafted ego.

Rollout and evaluation protocol. SpecRender rollouts contain 20 frames over 12 seconds. The short-horizon World Engine configuration (Li et al., 2026a) uses standard rendering for eight frames over six seconds. The longer rollout captures delayed collisions, departures from the drivable area, and sustained route progress. Across rendering strategies, we fix the VLA checkpoint, camera configuration, policy inputs, and reactive simulation settings. We select checkpoints using a self-normalized importance-sampling estimate of reward on held-out scenes and report PDMS under reactive simulation on `navtest-failure`.

A.2 Drafter and Verifier Variants

This section defines the drafter and verifier variants evaluated in Figure 3. A drafter proposes the ego poses used to construct $\tilde{s}_{t+j}^{(t)}$ in Equation 6, and a verifier compares each drafted configuration with the realized configuration at its target step. The drafter study holds the reactive actor rollout, renderer, and verifier fixed. The verifier study holds the drafter and remaining rendering pipeline fixed.

A.2.1 Drafter Variants

Let $\hat{a}_{t,j} = (\hat{p}_{t,j}, \hat{\psi}_{t,j})$ denote the jth pose in the current VLA prediction from Equation 5. All drafter outputs are expressed in the current ego frame and are transformed to world coordinates as in subsection 3.2.

Current VLA trajectory. This drafter uses the current policy prediction directly,

$$\bar{a}_{t,j} = \hat{a}_{t,j}, \qquad j \in \{1, \dots, N\}. \tag{11}$$

SpecRender uses this drafter. It requires neither an additional learned model nor an additional inference pass, and it retains AutoVLA's conditioning on camera history, ego dynamics, and the navigation command. We compare it with the kinematic and temporal baselines below in Figure 3a.

Constant velocity. The constant-velocity drafter (Schöller et al., 2020) estimates a per-step displacement from the first two VLA poses and extrapolates it linearly. With $d_t = \hat{p}_{t,2} - \hat{p}_{t,1}$, its prediction is

$$\bar{p}_{t,j} = \hat{p}_{t,1} + (j-1)d_t, \qquad \bar{\psi}_{t,j} = \hat{\psi}_{t,1}. \tag{12}$$

This baseline preserves the first predicted pose and assumes uniform translation with fixed heading thereafter. It does not represent acceleration or turning.

Constant turn rate and velocity. The CTRV drafter (Schubert et al., 2008) extends constant velocity with fixed curvature. We estimate the yaw increment $\omega_t = \angle(\hat{\psi}_{t,2} - \hat{\psi}_{t,1})$ from the first two predicted poses. If $\ell_t = \|\hat{p}_{t,2} - \hat{p}_{t,1}\|_2$ is their chord length, the traveled distance per step

is $v_t = \ell_t(\omega_t/2)/\sin(\omega_t/2)$ when $|\omega_t| > 10^{-6}$ and $v_t = \ell_t$ otherwise. For $n = j - 1$ and $\theta_n = \hat{\psi}_{t,1} + n\omega_t$, the rollout is

$$\bar{p}_{t,j} = \begin{cases} \hat{p}_{t,1} + \dfrac{v_t}{\omega_t} \begin{bmatrix} \sin\theta_n - \sin\hat{\psi}_{t,1} \\ -\cos\theta_n + \cos\hat{\psi}_{t,1} \end{bmatrix}, & |\omega_t| > 10^{-4}, \\ \hat{p}_{t,1} + nv_t \begin{bmatrix} \cos\hat{\psi}_{t,1} \\ \sin\hat{\psi}_{t,1} \end{bmatrix}, & |\omega_t| \le 10^{-4}, \end{cases} \qquad \bar{\psi}_{t,j} = \angle(\theta_n). \tag{13}$$

CTRV represents constant-radius turns but not changes in speed or curvature. As in the constant-velocity baseline, its motion estimate uses the current VLA prediction and no future simulator state.

Preceding trajectory. This drafter reuses the VLA trajectory produced at step $t-1$. We first transform that trajectory from its original ego frame into the current ego frame. We then align predictions by target step and discard the pose associated with the elapsed step. If $T_{t-1\to t}$ denotes this rigid frame transformation, the aligned prediction is

$$\bar{a}_{t,j} = T_{t-1\to t}(\hat{a}_{t-1,j+1})\,. \tag{14}$$

When the preceding trajectory does not reach the final target step, we repeat its last available pose. This baseline measures consistency between consecutive policy predictions, but cannot incorporate observations that first become available at step t.

Temporal trajectory ensemble. The ensemble aggregates the current prediction and recent VLA predictions that refer to the same absolute target step. Let $\mathcal{P}_{t+j}$ contain these aligned predictions in world coordinates. We compute position coordinate-wise and aggregate heading on the unit circle,

$$\begin{aligned} \bar{p}_{t,j} &= \mathrm{median}_{a\in\mathcal{P}_{t+j}}\, p(a), \\ \bar{\psi}_{t,j} &= \mathrm{atan2}\big(\mathrm{median}_{a\in\mathcal{P}_{t+j}} \sin\psi(a), \mathrm{median}_{a\in\mathcal{P}_{t+j}} \cos\psi(a)\big)\,. \end{aligned} \tag{15}$$

We use the three most recent trajectories. Alignment by target step ensures that the ensemble combines predictions for the same future time. The median reduces sensitivity to an isolated change in the predicted trajectory while retaining the policy's scene and route conditioning. If no earlier trajectory reaches a target step, we use the corresponding pose from the current VLA prediction.

Shared drafting protocol. Each drafter predicts only the ego trajectory and preserves the number of poses in the VLA output. When a baseline lacks the required trajectory history or frame transformation, it uses the current VLA prediction. We restrict alternative drafts to the maximum radial extent of the current prediction with an additional 5 m margin, which prevents extrapolated camera poses from leaving the reconstructed scene. After selecting an ego trajectory, we apply the same IDM rollout to obtain actor poses and render the resulting configuration according to Equation 6. The comparison therefore changes only the ego trajectory used for drafting. All variants are training-free and use no learned model beyond AutoVLA.

A.2.2 Verifier Variants

For a draft targeting step τ, define the ego position and heading errors

$$d_\tau^{\mathrm{pos}} = \|p_\tau^{\mathrm{e}} - \tilde{p}_\tau^{\mathrm{e}}\|_2, \qquad d_\tau^{\mathrm{yaw}} = \left|\angle\left(\psi_\tau^{\mathrm{e}} - \tilde{\psi}_\tau^{\mathrm{e}}\right)\right|. \tag{16}$$

Each verifier maps these errors and its rule-specific inputs to an ego acceptance decision. The ablation isolates this decision. Full SpecRender also applies the actor check in Equation 8, and reuses a draft only when both checks pass. Render completion is represented separately by c_τ in Equation 10.

Exact pose matching. The exact verifier is the strict reference. It accepts only when the drafted and realized ego poses agree up to the implementation tolerance $\varepsilon_{\mathrm{num}} = 10^{-6}$,

$$v_\tau^{\mathrm{exact}} = \mathbb{1}[d_\tau^{\mathrm{pos}} \le \varepsilon_{\mathrm{num}}]\,\mathbb{1}[d_\tau^{\mathrm{yaw}} \le \varepsilon_{\mathrm{num}}]\,. \tag{17}$$

This rule introduces no deliberate approximation, but rejects drafts after arbitrarily small changes due to replanning or numerical state updates. We include it as a conservative reference.

Fixed margin. Following the relaxed action-distance criterion of Spec-VLA (Wang et al., 2025), we adapt token verification to continuous ego poses. The draft passes when

$$v_\tau^{\text{fixed}} = \mathbb{1}[d_\tau^{\text{pos}} \le \epsilon_0]\, \mathbb{1}[d_\tau^{\text{yaw}} \le \epsilon_{\text{yaw}}]\,, \tag{18}$$

where ϵ_0 is constant across states and speeds. We omit the heading factor when the heading gate is disabled. The rule is inexpensive, but does not account for the ego speed or the proximity of surrounding traffic.

World-aware margin. The world-aware verifier adapts the fixed threshold using proximity to surrounding traffic, following the state-dependent principle of WA-SpecDec (Wen et al., 2026). Let d_τ^{near} be the distance from the realized ego to the nearest realized non-ego actor. We use

$$\epsilon_\tau^{\text{world}} = \epsilon_0 \,\text{clip}\left(\frac{d_\tau^{\text{near}}}{d_{\text{ref}}}, \alpha_{\min}, 1\right), \qquad v_\tau^{\text{world}} = \mathbb{1}\left[d_\tau^{\text{pos}} \le \epsilon_\tau^{\text{world}}\right] \mathbb{1}[d_\tau^{\text{yaw}} \le \epsilon_{\text{yaw}}]\,. \tag{19}$$

We use $d_{\text{ref}} = 10\,\text{m}$ and $\alpha_{\min} = 0.2$. When no actor position is available, the threshold defaults to ϵ_0. It decreases near another actor and approaches ϵ_0 as the nearest-actor distance reaches d_{ref}. The verifier therefore requires closer ego-pose agreement near surrounding traffic.

Regret-based verification. The regret-based verifier adapts the request-level accounting of ASD (Feng et al., 2026) to continuous pose error. At draft depth j, the local regret is $r_j = d_{t+j}^{\text{pos}}$ and the remaining suffix value is $q_j = K - j + 1$. A non-exact draft is accepted only when

$$\frac{r_j}{q_j} \le g, \qquad r_j \le B_{j-1}, \qquad m_{j-1} < m_{\max}, \tag{20}$$

where g is the local regret threshold, B_{j-1} is the remaining scene-level budget, and m_{j-1} counts accepted mismatches in the current block of K drafts. The implementation uses an initial budget of $4.0\,\text{m}$, $g = 0.25$, and $m_{\max} = 2$. The scene-level budget persists across blocks. The mismatch count resets at depth one. After accepting a draft, we update the ledger as $B_j = B_{j-1} - r_j$ and $m_j = m_{j-1} + 1$. Errors no greater than 10^{-6} are exact and consume no budget. Unlike the per-frame rules, this verifier limits accumulated pose error, but incurs the additional cost of maintaining and evaluating the ledger.

KERV-based margin. The KERV-based verifier (Zheng et al., 2026a) adapts the position tolerance to the drafted ego speed $\tilde{v}_\tau$. Its effective threshold is

$$\epsilon_\tau^{\text{kin}} = \epsilon_0 \,\text{clip}\left(\frac{\tilde{v}_\tau}{v_{\text{ref}}}, \beta_{\min}, \beta_{\max}\right), \qquad v_\tau^{\text{kin}} = \mathbb{1}\left[d_\tau^{\text{pos}} \le \epsilon_\tau^{\text{kin}}\right] \mathbb{1}[d_\tau^{\text{yaw}} \le \epsilon_{\text{yaw}}]\,. \tag{21}$$

When the drafted speed is unavailable, we use a scale of one. As specified in subsection A.1, we set $\epsilon_0 = 0.25\,\text{m}$, $v_{\text{ref}} = 5\,\text{m/s}$, $\beta_{\min} = 0.5$, and $\beta_{\max} = 2.0$. We disable the heading gate. The resulting position tolerance ranges from 0.125 to $0.5\,\text{m}$, requiring closer agreement at low speeds and allowing larger displacement at higher speeds. We use this verifier for SpecRender in all main experiments.

Verifier metrics. A verifier hit denotes a draft that passes the selected state check. A hit reduces latency only if the asynchronous render also finishes before step τ, as represented by $c_\tau v_\tau$ in Equation 10. End-to-end runtime further includes verification and fallback rendering. Acceptance rate alone therefore does not determine speedup.

A.3 Simulation-Step Runtime Breakdown

Profiling protocol. We report median GPU-worker latency for the SimEngine components in Figure 6, including engine updates, observation generation, MTGS rendering, and frame persistence. Total step latency covers the complete simulator step executed by the environment wrapper and excludes VLA inference. The remaining observation time includes render-state assembly, metadata and annotation construction, planner-frame preparation, and Python overhead. The total also includes environment information merging, completion checks, and wrapper overhead. We therefore measure total step latency directly rather than estimate it by summing the displayed components.



Figure 6: *SimEngine latency on the GPU worker by profiled component.* Each row compares MTGS with SpecRender using the KERV verifier. The indented render and frame-write rows are nested within `sensor.get_observations` and are not additive with their parent. Total step latency is measured independently. Percentages report the reduction from MTGS.

Before-step processing. `engine.before_step` passes the selected action to the simulator components that update the ego vehicle, surrounding actors, and episode state. It also gathers the information these components produce before the state update. This phase performs neither rendering nor speculative verification. Its latency is 0.06 ms under both configurations because SpecRender does not modify this processing.

State and metric updates. `engine.physics_step` advances the ego vehicle and reactive IDM actors. Its median latency changes from 15.54 to 12.18 ms. After this update, `engine.after_step` records the new simulator state and computes step-level metrics. The environment wrapper subsequently evaluates episode termination and truncation. This phase changes from 67.55 to 64.31 ms. SpecRender modifies neither phase, so these differences reflect run-to-run variation rather than the source of the reported speedup.

Observation generation. After the dynamics update, `sensor.get_observations` constructs the policy observation. It transforms the realized ego and actor states into the global reconstruction frame, assigns the timestamp and sensor calibration, obtains the rendered images, and persists the frame. This inclusive latency decreases from 196.15 to 62.36 ms, a 68.2% reduction. A completed draft that passes verification supplies the cached observation and avoids foreground rendering.

MTGS rendering. Within observation generation, `render.total` measures MTGS image synthesis. Standard rendering updates the Gaussian scene using the current actor states, configures the cameras, evaluates view-dependent color, rasterizes the Gaussian splats, composes and undistorts the images, and transfers the outputs to host memory. SpecRender performs this work in advance on a background worker. When KERV accepts a completed draft, the foreground call returns the cached result. Foreground rendering latency decreases from 103.62 to 41.38 ms, a 60.1% reduction. Fallback rendering and synchronization with incomplete drafts account for the remaining latency.

Frame writing. The nested `io.frame_write` timer measures metadata construction, image remapping, JPEG encoding, writes to the shared filesystem, and transfer of the completed frame description to the planner client. SpecRender accelerates this path by performing image remapping and JPEG encoding on the GPU before the camera outputs are written in parallel. Its median latency decreases from 88.67 to 18.58 ms, a 79.0% reduction, further reducing observation-generation and total simulator latency.

Overall reduction. Median simulation-step latency on the GPU worker decreases from 364.48 to 138.91 ms, corresponding to a 61.9% reduction and a 2.62× speedup. KERV accepts all drafts in this profile, so the result represents an operating point with high draft agreement and render readiness. Rejected or incomplete drafts require foreground rendering and reduce the speedup.

A.4 Qualitative Analysis of Challenging Rollouts

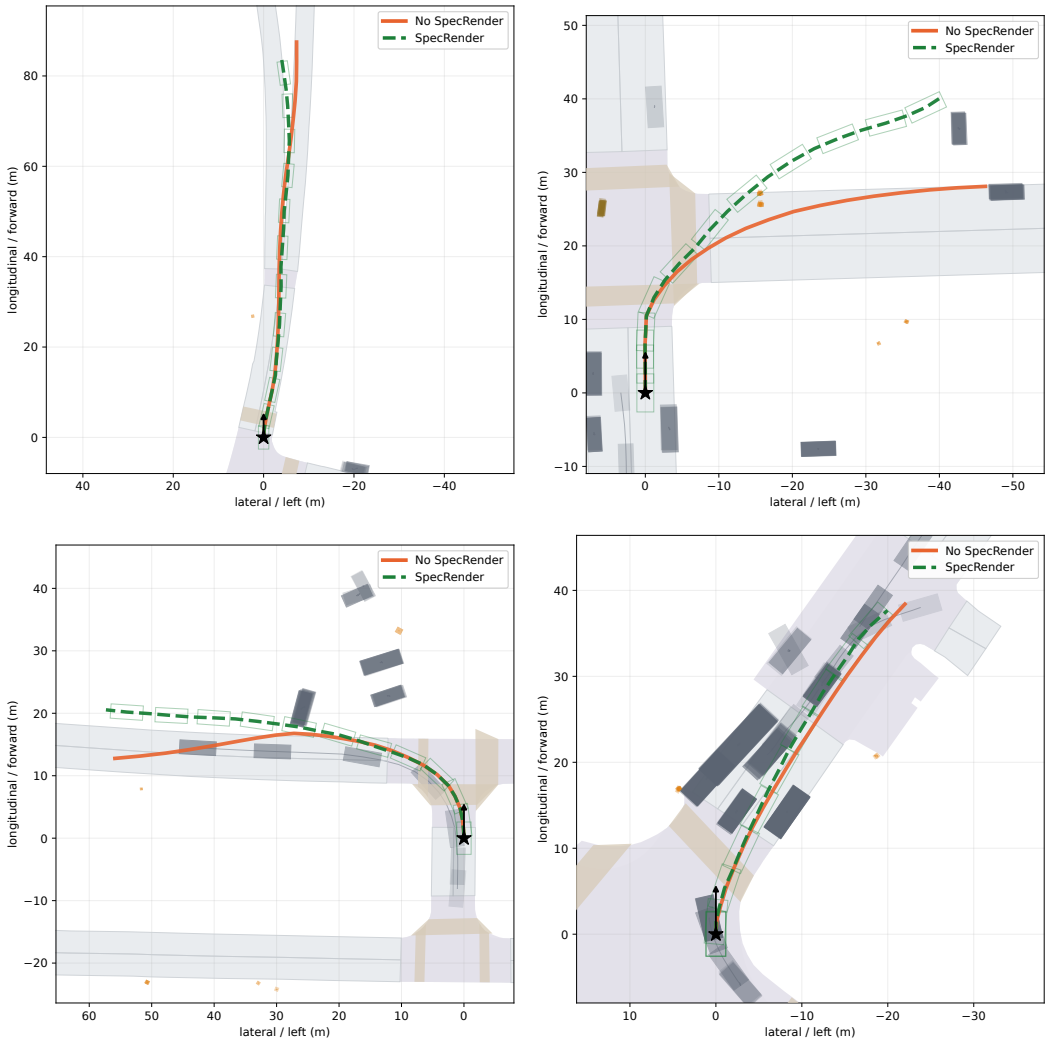


Figure 7: *Failure cases under speculative rendering.* Each bird's-eye-view map overlays matched rollouts initialized from the same scene state, with standard rendering in orange and SpecRender in dashed green. SpecRender departs the drivable area in the upper-left and lower-left examples, exhibits pronounced route separation at the upper-right junction, and terminates in a collision in the lower-right example. The trajectories initially remain close and diverge after a turn or junction decision, illustrating how small step-level differences can be amplified by subsequent replanning and closed-loop interaction.

Observed failure modes. Figure 7 shows four matched rollouts for which SpecRender and standard rendering produce different closed-loop outcomes. The upper-left and lower-left examples remain close through most of the maneuver, but the SpecRender trajectory eventually leaves the drivable area. The upper-right trajectories separate after entering the junction, while the lower-right trajectories follow a similar bend before the SpecRender rollout terminates in a collision. In each case, the difference in outcome is larger than the apparent difference between the plotted paths.

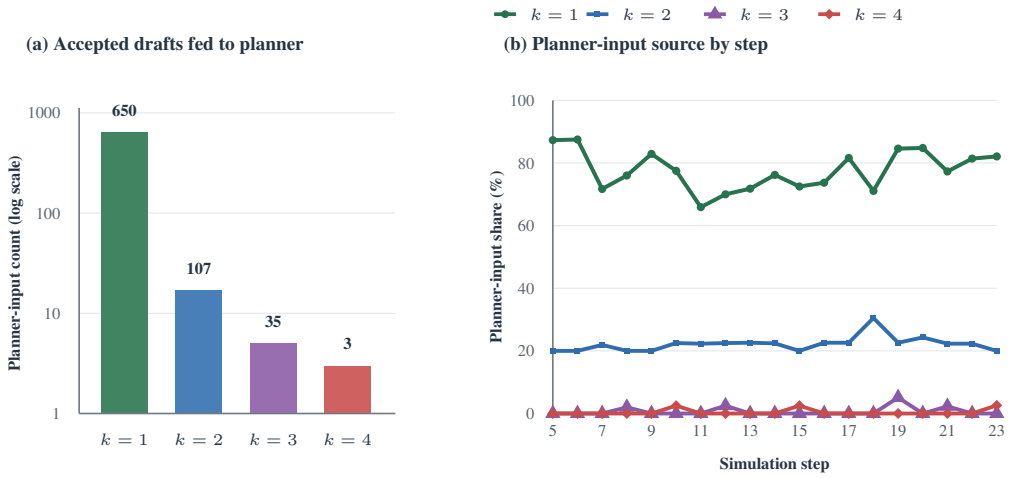


Figure 8: **Prediction depth of speculative frames supplied to AutoVLA.** (a) Number of accepted drafts used as planner inputs at each prediction depth. The logarithmic scale shows the concentration at $k = 1$ together with the contribution from deeper drafts. (b) Planner-input share across simulation steps. Predictions from $k = 1$ supply most inputs throughout the rollout, while $k = 2$ and $k = 3$ provide additional observations when deeper lookahead is useful.

Closed-loop interpretation. The BEV maps show that these failures do not arise from an immediate or large shift in the ego trajectory. Instead, the two rollouts begin to differ slightly near a bend, junction, or traffic interaction. At these points, a speculatively rendered observation can differ modestly from the observation produced by standard rendering, even after the corresponding states pass verification. AutoVLA may consequently predict a slightly different trajectory. The simulator executes the first pose of this prediction, renders the next observation from the resulting state, and replans. Repeating this process can amplify an initially small difference over several steps.

Reactive traffic provides a second source of amplification. Once the ego trajectories differ, surrounding actors respond to different ego motions, which changes the scene observed at later steps and can further separate the policy predictions. The final metrics are also determined by discrete events. A small late-stage deviation can therefore change whether a rollout crosses the drivable-area boundary or makes contact with another actor, even when most of the two trajectories remain close on the BEV map. This explains the pattern in Figure 7. The rollouts agree over much of the episode, diverge around a sensitive maneuver or interaction, and produce different outcomes only near the end.

A.5 Planner-Input Source by Prediction Depth

Multi-depth drafting sustains speculative coverage throughout the rollout. Figure 8 records the prediction depth of every speculative frame supplied to AutoVLA. Of the 795 drafted planner inputs, 650 come from $k = 1$, 107 from $k = 2$, 35 from $k = 3$, and 3 from $k = 4$. The first depth therefore serves 81.8% of speculative inputs and provides the primary path from trajectory prediction to render reuse. This concentration follows the serving policy, which selects the freshest completed draft for each target step.

The deeper predictions complement this primary path. Drafts from $k = 2$ contribute at simulation steps across the full measured interval and reach their largest share near step 18. Drafts from $k = 3$ provide additional coverage at selected steps, while $k = 4$ remains available at the longest lookahead. Their use reflects moments when advance rendering completes before a fresher candidate is ready. The stable dominance of $k = 1$ together with recurring contributions from $k = 2$ and $k = 3$ shows how the prediction horizon acts as a layered render queue. SpecRender turns each multi-step policy output into several opportunities for useful work, allowing rendering to proceed ahead of simulation and keeping reusable observations available throughout closed-loop generation.