SWIFTWM: AN EVENT-GUIDED WORLD MODEL FOR FAST MOTION PLANNING IN AUTONOMOUS DRIVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Timely adaptation to changing traffic is essential for autonomous driving. Dynamic vision sensors (DVS) capture sparse, asynchronous brightness changes with fine temporal resolution. Frame aggregation can obscure event timing and weaken their low-latency advantage. Iterative diffusion planning adds computational cost to trajectory updates. We propose SwiftWM, an event-guided latent world model for fast motion planning in autonomous driving. SwiftWM encodes dynamic spatiotemporal event graphs with a spiking graph neural network. Geometry-guided alignment and gated fusion integrate event features with RGB–LiDAR context and radar observations into a persistent bird's-eye-view latent state. DVS packets incrementally update affected regions while retaining the state elsewhere. A predictor forecasts future latents from the updated state. At each planning update, a diffusion planner conditions on current and predicted future latents to generate a new path with a single solver step, while a separate head predicts target speed. Predictive and planning supervision jointly optimize the world model and planner over causal sequences. We evaluate SwiftWM through closed-loop driving on Bench2Drive and recorded-input diagnostics of future-state prediction and computational cost. SwiftWM achieves a driving score of 89.82 and a success rate of 76.36% on Bench2Drive, while delivering a 76.38% reduction in state-maintenance computation through incremental execution compared with full recomputation in recorded-input evaluations.

1 INTRODUCTION

End-to-end Autonomous Driving (AD) connects sensor representation learning to motion planning (Prakash et al., 2021). Planning-oriented systems coordinate perception and prediction toward this goal (Hu et al., 2023), while unified transformer architectures integrate these tasks within a common model (Jia et al., 2025). In dynamic traffic, decisions depend on both scene understanding and the timely use of new observations. A pedestrian appearing or a vehicle changing lanes may require action between sensor updates. Computation-induced delays can further degrade closed-loop behavior (Zhang et al., 2026c). Responsive driving therefore requires an up-to-date scene representation that also anticipates near-term changes.

Dynamic vision sensors (DVS) report timestamped brightness changes with high temporal resolution (Gallego et al., 2022). They capture changes between image frames (Gehrig & Scaramuzza, 2024) and complement RGB appearance and LiDAR geometry. However, events are often aggregated into dense frames, which can obscure precise event timing and weaken their low-latency advantage. Asynchronous graph processing offers an alternative (Schaefer et al., 2022). Recurrent event–frame fusion (Gehrig et al., 2021) and collaborative graph representations (Li et al., 2025a) further connect event evidence to frame context. For planning, this evidence must also be linked to scene context and integrated over time as sensors update at different rates.

Driving world models anticipate scene evolution through visual forecasting (Wang et al., 2024), occupancy prediction (Zheng et al., 2024a), or latent dynamics (Li et al., 2025b). Predicted BEV states can support trajectory evaluation (Li et al., 2025c) and refinement (Zhang et al., 2026a). Meanwhile, truncated diffusion reduces trajectory-generation cost (Liao et al., 2025), and BridgeDrive provides

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.

another efficient diffusion planning formulation (Liu et al., 2026c). However, faster sampling cannot reveal scene changes missing from the planner's input. Predictions must also incorporate new evidence as it arrives. This motivates a shared predictive state that connects asynchronous sensing with fresh planning.

We introduce SwiftWM, an event-guided world model with streaming state updates for autonomous driving. As shown in Figure 1, it reuses the scene state across time and replans the trajectory from the latest updated state. SwiftWM maintains a persistent bird's-eye-view (BEV) latent shared by prediction and planning. A spiking graph encoder processes the incoming event stream. Geometry-guided correspondence aligns event features with RGB–LiDAR context, while gated fusion integrates radar features. Cell-wise recurrence updates affected regions and retains the state elsewhere. A predictor forecasts future latents using the current latent, ego-motion information, and observation ages. At each planning update, a diffusion planner uses current and predicted latents to generate a new path with one solver step, while a separate head predicts target speed. Training unrolls finite causal sequences, while inference retains the recurrent state across event packets.

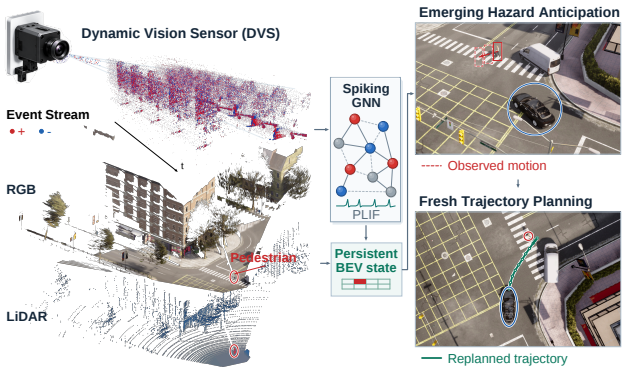

Figure 1: Conceptual overview of SwiftWM.

We collect DVS-augmented expert demonstrations using LEAD (Nguyen et al., 2026) and PDM-Lite (Sima et al., 2024) as expert policies. The SwiftWM models evaluated here are trained on the LEAD-DVS training split. Our evaluation combines Bench2Drive closed-loop driving (Jia et al., 2024), future-state prediction, and planning-cost measurements. Controlled comparisons examine future-state conditioning and the diffusion solver budget, and qualitative driving examples illustrate the resulting behavior. Our contributions are summarized as follows:

- We propose SwiftWM, a multimodal world model designed for rapid responses to scene changes. It combines DVS sensing, causal dynamic event graphs, and incremental inference to incorporate new observations.
- We design a geometry-guided fusion mechanism that incrementally updates a shared BEV latent from multimodal observations. The current and predicted future latents jointly condition single-step diffusion planning.
- We report competitive closed-loop results on Bench2Drive and evaluate future-state prediction, sensitivity to observation timing, computational cost, and inference latency.

2 Related Work

End-to-End Driving. End-to-end driving has progressed from multimodal fusion in TransFuser to task-coordinated planning in UniAD (Prakash et al., 2021, Hu et al., 2023). DriveTransformer reuses historical queries, while Hydra-NeXt combines control prediction with trajectory refinement (Jia et al., 2025, Li et al., 2025d). Additionally, DiffusionDrive and BridgeDrive reduce the sampling cost of diffusion planning (Liao et al., 2025, Liu et al., 2026c). Although these methods improve temporal reuse and action generation, faster planning cannot reveal changes absent from its inputs.

Event-Based Vision. Events offer fine temporal resolution but limited appearance context (Gallego et al., 2022). AEGNN reduces computation through sparse graph updates (Schaefer et al., 2022). RAM, DAGr, and ACGR incorporate complementary frame information through recurrent fusion or asynchronous graphs (Gehrig et al., 2021, Gehrig & Scaramuzza, 2024, Li et al., 2025a). However, their perception outputs alone do not specify how new evidence should change a driving action. EventDrive extends event–RGB integration to language-grounded prediction and planning

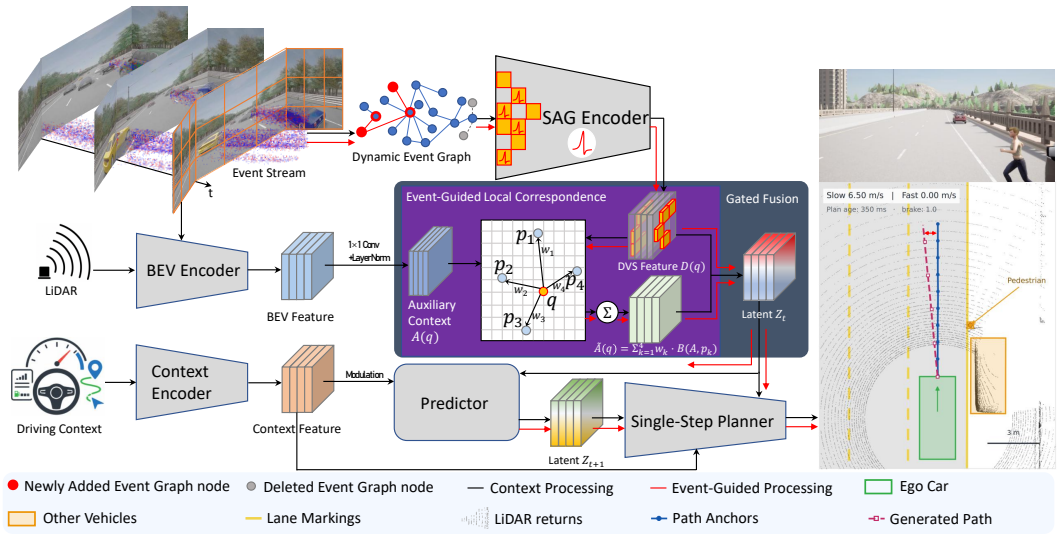

Figure 2: Overview of SwiftWM. Event-guided multimodal fusion maintains a persistent BEV state. Current and predicted future latents condition single-step diffusion path generation, with a separate head for target-speed prediction.

(Lu et al., 2026). In contrast, SwiftWM maintains a recurrent metric BEV state shared by future prediction and trajectory generation, linking asynchronous sensing directly to closed-loop planning.

World Models. Drive-WM and OccWorld predict visual and occupancy futures, respectively (Wang et al., 2024, Zheng et al., 2024a). Although image synthesis provides explicit scene predictions, it adds computation to frequent planning (Li et al., 2025c). Latent approaches instead learn dynamics and predictive features for control and planning (Hu et al., 2022, Li et al., 2025b). WoTE and ResWorld use future BEV states for trajectory evaluation and refinement (Li et al., 2025c, Zhang et al., 2026a). Additionally, Drive-HWM separates prediction and action generation across two timescales (Fan et al., 2026). These designs support anticipation, but forecasts can become outdated without timely observation updates.

3 Method

This section details the SwiftWM architecture. As shown in Figure 2, SwiftWM integrates spiking graph encoding, gated multimodal fusion, and latent-state prediction with single-step diffusion planning. We detail these components below, followed by the training objective and online inference procedure.

3.1 Event Graph Construction and Encoding

Let k index updates and t_k denote physical acquisition time. The new event packet is $\mathcal{E}_k = \{(\boldsymbol{\xi}_i, t_i^e, p_i)\}$, with pixel location $\boldsymbol{\xi}_i$, event time t_i^e, and polarity p_i. The state uses an anchor grid $\mathcal{Q}$ of size $H_g \times W_g$ over $[x_-, x_+] \times [y_-, y_+]$. Context and radar retain their acquisition times. Their ages are $\mathbf{a}_k = (t_k - t_k^C, t_k - t_k^R)$.

Calibrated rays define a ground-plane prior:

$$\mathbf{r}_i = \mathbf{R}_{\text{cam}} \mathbf{K}_{\text{cam}}^{-1} [\xi_i^x, \xi_i^y, 1]^\top, \qquad \mathbf{x}_i = (\mathbf{o}_{\text{cam}} + \eta_i \mathbf{r}_i)_{xy}, \quad \eta_i = -o_{\text{cam},z}/r_{i,z} > 0. \tag{1}$$

Dynamic graph construction. A causal pose maps the ground-plane points to the anchor. We retain points in $\mathcal{D}_E$ and quantize them into bins of size $(\delta_x, \delta_y, \delta_e)$. Each occupied bin supplies its first event and an updated count. The latest N_E representatives form $\mathcal{V}_k$. Their features encode normalized position, packet-relative time, polarity, and log count. For node i, causal neighbors satisfy

$$\mathcal{N}_k(i) = \text{Latest}_{K_E} \{ j : \|\mathbf{x}_j - \mathbf{x}_i\|_\infty \le r_E, \quad 0 < t_i^e - t_j^e \le \delta_{\text{edge}} \}. \tag{2}$$

The edges are $\mathcal{A}_k = \{(j,i) : j \in \mathcal{N}_k(i) \cup \{i\}\}$. Thus $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{A}_k)$ follows the arriving events and includes self edges. Relative spatial offsets and time lags, normalized by $(r_E, r_E, \delta_{\text{edge}})$, provide fixed physical coordinates for graph convolution.

Spiking graph encoding. A linear–spiking projection initializes node features $\mathbf{h}_i^{(0)}$. The encoder has L_g layers of widths d_l. Each combines spline messages with a parametric leaky integrate-and-fire (PLIF) neuron through message aggregation, membrane integration, spike generation, and reset:

$$\mathbf{j}_i^{(l)} = \mathrm{LN}\Big(\mathrm{SplineConv}_l(\mathbf{h}^{(l-1)}, \mathcal{A}_k)_i\Big), \tag{3}$$

$$\mathbf{u}_i^{(l)} = (1-\beta_l)\mathbf{u}_{i,\text{prev}}^{(l)} + \beta_l \mathbf{j}_i^{(l)}, \qquad \beta_l = \sigma(w_l), \tag{4}$$

$$\boldsymbol{\zeta}_i^{(l)} = \Theta(\mathbf{u}_i^{(l)} - \vartheta_l), \qquad \mathbf{u}_{i,+}^{(l)} = \mathbf{u}_i^{(l)} \odot (1 - \boldsymbol{\zeta}_i^{(l)}). \tag{5}$$

Spline messages use max aggregation. Spikes form the next node features, with an identity residual where channel widths match. The membrane starts at zero for each packet graph; persistent scene information is carried by the BEV state below. Training uses an arctangent surrogate gradient and detaches the spike reset. A linear node readout is max-pooled within each BEV cell. Its pooled feature and log node count are projected and normalized to obtain $E_k(q)$; $m_k(q)$ marks nonempty cells. Empty graphs contribute no evidence.

Incremental evidence and state. A finite bank retains packet contributions:

$$\overline{E}_k(q) = \frac{\sum_{j \in \mathcal{B}_k} \rho^{k-j} m_j(q) E_j(q)}{\max\Big(\epsilon_B, \sum_{j \in \mathcal{B}_k} \rho^{k-j} m_j(q)\Big)}, \qquad \mathcal{B}_k = \{j : k - L_B < j \le k\}. \tag{6}$$

A revised packet prefix replaces its contribution without duplicating evidence. The active set $\mathcal{U}_k$ includes event-supported cells and cells whose bank evidence expires. A context or radar refresh activates the entire grid. Other cells retain their recurrent state. On reanchoring, a bounded causal history is replayed in the new frame, including the current packet exactly once. Numerical settings remain in Appendix A.1.

3.2 Geometry-Guided Fusion and Latent Rollout

A frozen RGB–LiDAR backbone produces $C_k^{\text{nat}} \in \mathbb{R}^{d_C \times H_c \times W_c}$. Backward sampling maps it to the anchor as C_k. A pointwise projection gives $C_k^a \in \mathbb{R}^{d_E \times H_g \times W_g}$. The original context channels remain in the shared state.

At supported cell q, event and context features predict N_s offsets and softmax weights. Each offset is bounded by r_s. Bilinear sampling with zero padding yields $\widetilde{C}_k^a(q)$. With $D_k(q) = P_E \overline{E}_k(q)$, the event gate, event–context fusion, and radar fusion are

$$G_{E,k}(q) = \sigma\Big(g_E[\widetilde{C}_k^a(q), D_k(q)]\Big), \tag{7}$$

$$F_{E,k}(q) = C_k^a(q) + \overline{m}_k(q) G_{E,k}(q) \odot f_E[\widetilde{C}_k^a(q), D_k(q)], \tag{8}$$

$$F_k(q) = F_{E,k}(q) + \sigma(g_R[F_{E,k}(q), B_{R,k}(q)]) \odot B_{R,k}(q). \tag{9}$$

Here $\overline{m}_k(q)$ is true when any retained packet supports cell q. $B_{R,k}$ is the radar field obtained by point encoding and mean scatter. Figure 3 depicts these gates, using $A = C_k^a$, $D = D_k$, $m_E = \overline{m}_k$, $f = f_E$, and $g = g_E$.

Incremental state updates retain unaffected cells:

$$M_k(q) = \begin{cases} \mathrm{GRU}(F_k(q), M_{k^-}(q)), & q \in \mathcal{U}_k, \\ M_{k^-}(q), & q \notin \mathcal{U}_k, \end{cases} \tag{10}$$

$$U_k = \mathrm{LN}(M_k + C_k^a), \qquad Z_k = [C_k + r_C(U_k), U_k].$$

M_{k^-} is the packet-start memory. Prefix replacement reuses this starting state. r_C maps recurrent features to context channels. Thus $Z_k \in \mathbb{R}^{d_Z \times H_g \times W_g}$ with $d_Z = d_C + d_E$. Reanchoring reconstructs state from a bounded causal history.

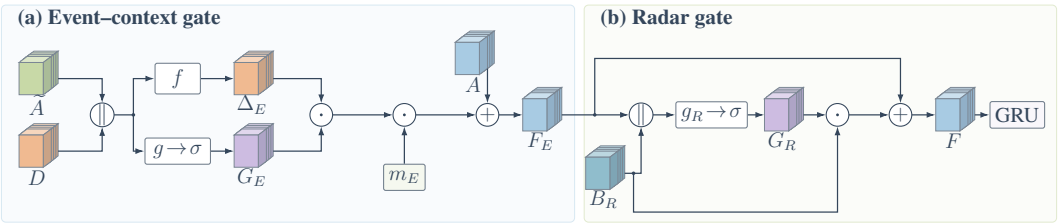

Figure 3: Gated multimodal fusion in SwiftWM. (a) Event–context fusion integrates event features with spatially aligned RGB–LiDAR context through a learned gate. (b) Radar fusion further incorporates radar features through a separate gate. The resulting fused representation is passed to the recurrent unit to update the persistent BEV state.

For horizon $h \in \mathcal{H}_+$, the predictor computes

$$\widehat{U}_{k,h} = U_k + f_h(U_k, v_k/v_{\text{ref}}, \mathbf{a}_k), \qquad \widehat{Z}_{k,h} = [C_k + r_C(\widehat{U}_{k,h}), \widehat{U}_{k,h}]. \tag{11}$$

v_k is ego speed. Forecast $\widehat{Z}_{k,h}$ targets time $t_k + h$ in the same anchor. Set $\widehat{Z}_{k,0} = Z_k$ and $\mathcal{H} = \{0\} \cup \mathcal{H}_+$. Forecasts condition on observations, not candidate actions.

3.3 Single-Step Fresh Planning

The planner mixes horizons independently per channel:

$$\begin{aligned} V_k[c,q] &= \sum_{h\in\mathcal{H}} \alpha_{h,c} \widehat{Z}_{k,h}[c,q], \\ \alpha_{h,c} &= \text{softmax}_h(\gamma_{h,c}), \qquad B_k = \mathcal{S}_{T_k}(V_k). \end{aligned} \tag{12}$$

T_k is the ego pose relative to the anchor. $\mathcal{S}_{T_k}$ samples the ego-frame planner grid without discarding state channels. Navigation inputs $\mathbf{n}_k$ contain measured speed, route command, and target points.

Let $\{A^m\}_{m=1}^{N_a}$ be path anchors, each with N_p spatial points. A diffusion transition generates candidate paths:

$$\begin{aligned} \widehat{\Pi}_k^m &= \Phi_{\theta_p}^{\tau_{\max}\to 0}(A^m; B_k, \mathbf{n}_k), \quad \mu_k = \arg\max_m c_k^m, \\ \widehat{\Pi}_k &= \widehat{\Pi}_k^{\mu_k}, \qquad\qquad\qquad \mathbf{p}_k^v = \text{softmax}(g_v(B_k, \mathbf{n}_k)). \end{aligned} \tag{13}$$

τ denotes diffusion time, distinct from acquisition time t_k. The transition uses a single solver step. Anchor scores c_k^m select mode μ_k. A separate categorical head predicts speed. Each update generates a new plan from the latest state. The path-and-speed controller executes the newly generated output.

3.4 Training Objective

We unroll causal sequences after burn-in. The main supervision teaches the shared state to predict scene features and their temporal change:

$$\begin{aligned} \mathcal{L}_f &= \text{Avg}_{k,h,q} \, \|\widehat{Z}_{k,h}(q) - Y_{k,h}(q)\|_2^2 / d_Z, \\ \mathcal{L}_\Delta &= \text{Avg}_{k,h>0,q} \, \|(\widehat{Z}_{k,h} - Z_k)(q) - (Y_{k,h} - Y_{k,0})(q)\|_2^2 / d_Z. \end{aligned} \tag{14}$$

$Y_{k,h}$ is a frozen feature target aligned to the anchor. The weighted averages use valid labeled instants, horizons, and cells. Future poses align supervision only. Joint training uses

$$\mathcal{L} = \lambda_f \mathcal{L}_f + \lambda_\Delta \mathcal{L}_\Delta + \mathcal{L}_{\text{aux}} + \lambda_P \mathcal{L}_P. \tag{15}$$

$\mathcal{L}_{\text{aux}}$ groups context reconstruction, occupancy, semantic, and alignment-offset supervision. $\mathcal{L}_P$ retains the planner's path, anchor-selection, waypoint, and speed losses. Full and Current receive identical predictive supervision; they differ in the state supplied to planning. Appendix A.2 gives all terms, reductions, and coefficients.

Online procedure. Algorithm 1 performs one update from state $\mathsf{s} = (\mathcal{X}, \mathcal{B}, M, \Gamma)$, where Γ stores anchor metadata. Initialize $\mathsf{s}_{-1} = (\varnothing, \varnothing, \mathbf{0}, \varnothing)$ once. The caller retains the returned state and executes the returned path and speed after burn-in.

Algorithm 1: Online inference of SwiftWM

Input: $k, \mathcal{E}_k, C_k^{\mathrm{nat}}, \mathcal{R}_k, T_k, \mathbf{a}_k, \mathbf{n}_k, \mathsf{s}_{k-1}$
Parameters: $N_{\mathrm{burn}}, N_{\mathrm{replay}}, L_B, \rho, \mathcal{H}$
Output: $\widehat{\Pi}_k, \mathbf{p}_k^v, \mathsf{s}_k$

$(\mathcal{X}, \mathcal{B}, M, \Gamma) \leftarrow \mathsf{s}_{k-1}$
$\mathcal{X} \leftarrow \mathrm{Append}_{N_{\mathrm{replay}}}(\mathcal{X}; k, \mathcal{E}_k, C_k^{\mathrm{nat}}, \mathcal{R}_k, T_k, \mathbf{a}_k), \quad \mathcal{I}_k \leftarrow \{k\}$
if $k = 0 \ \vee$ ReanchorRequired(k) **then**
  $\mathcal{I}_k \leftarrow \mathrm{Indices}(\mathcal{X})$ // causal replay
  $(\Gamma, \{T_j\}_{j \in \mathcal{I}_k}) \leftarrow \mathrm{Reanchor}(\mathcal{X}, \Gamma), \quad M \leftarrow \mathbf{0}, \quad \mathcal{B} \leftarrow \varnothing$
foreach $j \in \mathcal{I}_k$ *in acquisition order* **do**
  $\mathcal{G}_j \leftarrow \mathrm{Graph}(\mathcal{E}_j, T_j), \quad (E_j, m_j) \leftarrow \mathrm{Pool}(\mathrm{SAG}(\mathcal{G}_j))$
  $(\mathcal{B}, \overline{E}_j, \overline{m}_j, \mathcal{U}_j) \leftarrow \mathrm{ReplaceExpire}(\mathcal{B}; j, E_j, m_j; L_B, \rho)$
  $(C_j, C_j^a, B_{R,j}) \leftarrow \mathrm{AlignEncode}(C_j^{\mathrm{nat}}, \mathcal{R}_j, T_j)$
  if $\mathrm{Arrived}(C_j^{\mathrm{nat}}) \ \vee \ \mathrm{Arrived}(\mathcal{R}_j)$ **then**
    $\mathcal{U}_j \leftarrow \mathcal{Q}$
  $F_j \leftarrow \mathrm{GatedFusion}(C_j^a, \overline{E}_j, \overline{m}_j, B_{R,j})$
  $M(q) \leftarrow \mathrm{GRU}(F_j(q), M(q)), \quad q \in \mathcal{U}_j$ // retain other cells
  $U_j \leftarrow \mathrm{LN}(M + C_j^a), \quad Z_j \leftarrow [C_j + r_C(U_j), U_j]$
$\mathsf{s}_k \leftarrow (\mathcal{X}, \mathcal{B}, M, \Gamma)$
if $k < N_{\mathrm{burn}}$ **then return** $(\varnothing, \varnothing, \mathsf{s}_k)$
$\widehat{Z}_{k,0} \leftarrow Z_k, \quad \widehat{Z}_{k,h} \leftarrow \mathrm{Predict}_h(Z_k, v_k, \mathbf{a}_k), \quad h \in \mathcal{H}_+$
$V_k \leftarrow \sum_{h \in \mathcal{H}} \alpha_h \odot \widehat{Z}_{k,h}, \quad B_k \leftarrow \mathcal{S}_{T_k}(V_k)$
$(\{\widehat{\Pi}_k^m\}_m, \{c_k^m\}_m) \leftarrow \Phi_{\theta_p}^{\tau_{\max} \to 0}(\{A^m\}_m; B_k, \mathbf{n}_k)$ // one solver step
$\mu_k \leftarrow \arg\max_m c_k^m, \quad \widehat{\Pi}_k \leftarrow \widehat{\Pi}_k^{\mu_k}, \quad \mathbf{p}_k^v \leftarrow \mathrm{softmax}(g_v(B_k, \mathbf{n}_k))$
return $(\widehat{\Pi}_k, \mathbf{p}_k^v, \mathsf{s}_k)$

4 Experiments

In this section, we evaluate the effectiveness of SwiftWM for autonomous driving. The model is implemented in PyTorch and trained on two NVIDIA RTX 4090 GPUs using the 935-route training split of LEAD-DVS. We evaluate closed-loop driving on Bench2Drive (Jia et al., 2024) in CARLA 0.9.15 and measure policy-service time on a single NVIDIA RTX 3090 GPU. Our evaluation covers closed-loop driving performance, future semantic and occupancy prediction, and computational cost. Recorded-input ablations examine future-state conditioning, event-input sensitivity, and the diffusion solver budget, while qualitative examples illustrate driving behavior and generated paths. Implementation and training settings are detailed in Appendix A, with further evaluation details in Appendix B.

Datasets. We collect two DVS-augmented demonstration datasets in CARLA 0.9.15 (Dosovitskiy et al., 2017) using the LEAD and PDM-Lite expert pipelines (Nguyen et al., 2026; Sima et al., 2024). The resulting collections are LEAD-DVS and PDM-Lite-DVS, with LEAD-DVS providing the training data for the reported SwiftWM models. LEAD-DVS draws from the official LEAD route set with a cap on the number of routes per scenario category. Removing incomplete and unsuccessful runs leaves 1,579 routes, partitioned into 935 training, 105 validation, and 539 test routes. Recorded-input evaluation uses the test split. PDM-Lite-DVS covers the full official PDM-Lite route release across Towns 01–05, Town10HD, Town12, and Town13, retaining 5,246 routes after quality filtering. Closed-loop evaluation uses the separate 220-route Bench2Drive benchmark (Jia et al., 2024).

Both collections use a forward-facing DVS with a resolution of 768×768, a horizontal field of view of 60°, and positive and negative contrast thresholds of 0.3. In LEAD-DVS, the sensor shares the position of the central front RGB camera. In PDM-Lite-DVS, it is mounted at $(0.35, 0, 2.25)$ m in

the ego frame with zero relative roll, pitch, and yaw. The x, y, and z axes point forward, rightward, and upward, respectively.

4.1 Prediction Quality and Planning Cost

Future prediction. We compare learned future predictions with current-state persistence on fixed recorded observations from the LEAD-DVS test split. Persistence assumes the scene remains unchanged and maps current predictions to future evaluation coordinates. Figure 4 shows that learned prediction outperforms persistence at both horizons in Full and Current. For Full, semantic mIoU improves by 8.49 and 8.91 percentage points at 0.5 and 1.0 s, respectively. Occupancy AP improves by 41.98 and 46.97 points. Both methods become less accurate when predicting further into the future, but occupancy AP drops less for learned prediction than for persistence. These results show that learned forecasting captures scene changes that coordinate transport alone cannot represent.

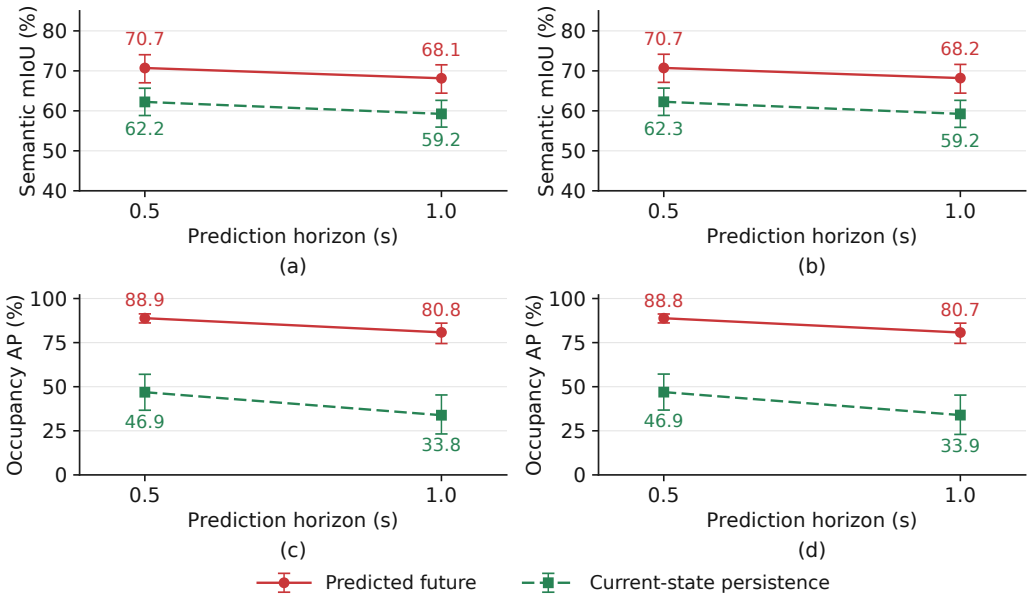

Figure 4: Future prediction versus persistence. Semantic mIoU for (a) Full and (b) Current, and occupancy AP for (c) Full and (d) Current. Full plans with current and predicted latents, while Current uses only current latents. Error bars show pointwise group-bootstrap confidence intervals over 32 test groups for semantics and 31 for occupancy.

Planning cost. We compare diffusion planning with 20, 5, and 1 solver steps. These configurations are denoted F20, F5, and F1, respectively. Each update generates a new path. On 95 test routes, F1 reduces mean planner time from 141.04 to 17.45 ms, yielding an $8.08\times$ speedup over F20 (Table 1). Mean ADE also decreases from 0.0264 to 0.0219 m. Thus, additional solver steps increase computation without improving mean path accuracy on these inputs. Speed CE remains unchanged because the separate speed head produces identical logits across solver budgets.

We further measure policy runtime on ten test scenes, totaling 6,000 updates per setting across three timing repetitions (Table 2). Timing includes the backbone, state updates, planning, control, and trace writing. F1 reduces mean policy time from 193.90 to 60.65 ms, a $3.20\times$ speedup, while mean ADE decreases from 0.02039 to 0.01671 m. The computational benefit therefore extends beyond the planner alone.

Table 1: Solver-budget diagnostic.

Steps	ADE (m)	Speed CE	Mean (ms)
20	0.0264	0.2573	141.04
5	0.0256	0.2573	43.53
1	0.0219	0.2573	17.45

Table 2: Recorded-input planning cost and error.

Policy	Mean (ms)	ADE (m)	Speed CE
F20	193.90	0.02039	0.4324
F5	88.68	0.01984	0.4324
F1	60.65	0.01671	0.4324

Incremental computation. We compare full and incremental execution with the same recorded inputs and single-step planner. Full execution recomputes backbone features, context projections, and BEV state operations. Incremental execution reuses unchanged features and updates only affected cells under the same state-update rules. The event graph encoder and planner run in both modes. Evaluation uses eight scenes from the LEAD-DVS test split on one NVIDIA RTX 3090, with three repetitions and 3,672 measured updates per mode after warm-up. Timing covers packet preprocessing through control output, including GPU synchronization and trace writing. State GMAC counts only matrix multiply–accumulates in state-maintenance Linear, Conv2d, and GRUCell operations, averaged over measured updates. Peak denotes PyTorch allocated GPU memory.

Table 3: Full and incremental computation per event-window update.

Mode	State GMAC	Policy (ms)	Peak (GiB)
Full update	3.162	178.16	0.766
Incremental update	0.747	144.00	0.755

Table 3 shows that incremental execution reduces state-maintenance computation from 3.162 to 0.747 GMAC, a 76.38% reduction. Mean policy runtime decreases from 178.16 to 144.00 ms, yielding a $1.24\times$ speedup. Outputs remain within the predefined numerical tolerances across all 48 replays. The reported speedup covers the full policy computation, including backbone-feature reuse. These results demonstrate lower arithmetic and policy runtime while preserving numerical agreement between the two executions.

4.2 Closed-Loop Evaluation

Table 4 summarizes closed-loop performance on Bench2Drive. SwiftWM obtains a DS of 89.82 and SR of 76.36%. Compared with SwiftWM w/o DVS, the full model achieves 5.71 points higher DS and 14.09 percentage points higher SR. It also improves all five driving-ability scores, showing that event input benefits multiple driving behaviors.

The emergency-braking results are particularly relevant to our design goal. SwiftWM achieves an EB score of 96.67%, compared with 85.00% without DVS, a gain of 11.67 percentage points. It also obtains the highest reported EB score among the methods in the table, exceeding R2SE (Liu et al., 2026b) by 6.67 percentage points. These results support the value of DVS input for emergency-braking decisions and are consistent with SwiftWM's goal of responding promptly to scene changes.

Table 4: Comparative results on Bench2Drive. EB means emergency braking. GW means giving way. Sign means traffic signs. Bold means the best. w/o means without.

Method	WM	Diff.	DS	SR	Merging	Overtaking	EB	GW	Sign
GenAD (Zheng et al., 2024b)	×	×	44.81	15.90	–	–	–	–	–
TF++ (Zimmerlin et al., 2024)	×	×	84.21	67.27	58.75	57.77	83.33	40.00	82.11
DriveTransformer-L (Jia et al., 2025)	×	×	63.46	35.01	17.57	35.00	48.36	40.00	52.10
Hydra-NeXt (Li et al., 2025d)	×	✓	73.86	50.00	40.00	64.44	61.67	50.00	50.00
WoTE (Li et al., 2025c)	✓	×	61.71	31.36	–	–	–	–	–
SimLingo (Renz et al., 2025)	×	×	85.07	67.27	54.01	57.04	88.33	**53.33**	82.45
DeepSight (Zhang et al., 2026b)	✓	×	86.23	71.36	60.00	**91.11**	78.33	50.00	71.58
R2SE (Liu et al., 2026b)	×	×	86.28	69.54	53.33	61.25	90.00	50.00	84.21
DIVER (Song et al., 2026)	×	✓	68.90	36.75	35.08	25.09	41.09	50.00	59.21
TakeAD (Liu et al., 2026a)	×	×	71.39	40.83	30.77	35.56	56.67	50.00	42.02
PerlAD (Gao et al., 2026)	✓	×	78.70	57.27	40.00	75.56	68.33	50.00	52.11
Diffusion-Phy (Hu et al., 2026)	×	✓	67.49	–	–	–	–	–	–
SwiftWM w/o DVS	✓	✓	84.11	62.27	43.75	53.33	85.00	30.00	78.95
SwiftWM	✓	✓	**89.82**	**76.36**	**67.50**	64.44	**96.67**	40.00	**88.42**

Beyond emergency braking, merging shows the largest improvement over SwiftWM w/o DVS, rising from 43.75 to 67.50 (+23.75 points). Overtaking also improves by 11.11 points, extending the gains to maneuvers that require interaction with nearby vehicles. SwiftWM achieves the highest merging

and traffic-sign scores among the compared methods, alongside its leading DS and SR. Performance varies across abilities: DeepSight achieves a higher overtaking score, while SimLingo performs better on giving way. This pattern identifies merging, emergency braking, and traffic-sign handling as the model's strongest categories in this comparison.

4.3 VISUALIZATIONS

Right-side cut-in. Figure 5 compares three methods in a Town05 scenario where a vehicle cuts into the ego lane from the right. At $t = -0.2\,\text{s}$, the DVS view reveals a partial wheel contour at the right image boundary. The vehicle becomes visible in the displayed RGB views at $t = 0$. SwiftWM is already braking before this point and shows stronger sustained braking around the cut-in onset than the two baselines. This temporal correspondence is consistent with early event cues supporting timely braking.

The event stream also captures the cut-in's spatial progression: the initially peripheral response develops into a recognizable vehicle outline as the car moves inward, with its rear silhouette visible by $+0.75\,\text{s}$. This evolution aligns with the BEV sequence, where the neighboring vehicle moves from the ego vehicle's right side into the lane ahead.

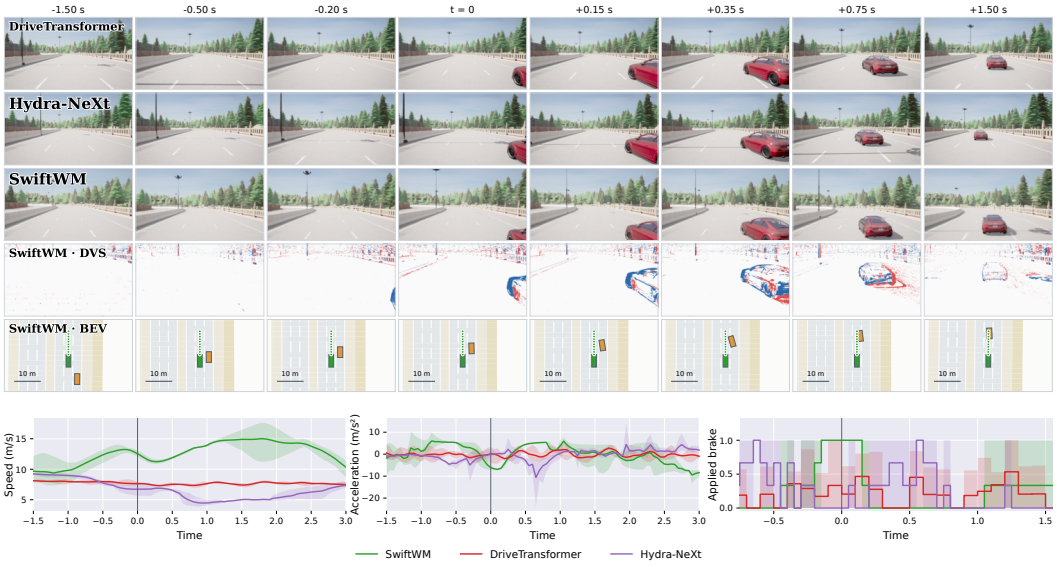

Figure 5: Comparison of DriveTransformer, Hydra-NeXt, and SwiftWM in a Town05 scenario where a vehicle cuts into the ego lane from the right. The first three rows show their front-view RGB images, respectively. The next two rows show SwiftWM's event observations and BEV views. The bottom plots compare vehicle speed, acceleration, and brake input.

5 CONCLUSION

We propose SwiftWM, an event-guided multimodal world model for responsive autonomous driving. DVS captures scene changes between slower sensor updates. Aggregating these events into frames can obscure their timing and weaken this temporal advantage. We therefore organize events into causal dynamic graphs that retain event-time information and local spatial relationships. A spiking graph neural network aggregates information along these connections to extract local spatiotemporal features. Frequent event updates also make repeated full-scene computation costly. Incremental inference reduces this overhead by reusing unchanged features and updating affected cells in a persistent BEV state. Geometry-guided alignment and gated fusion integrate event features with RGB–LiDAR context and radar observations. The updated state supports near-term forecasting, and current and predicted states condition single-step diffusion planning, with a separate head predicting target speed. Bench2Drive experiments show improved closed-loop driving with DVS. Recorded-input evaluations show that learned forecasts outperform current-state persistence and that incremental execution reduces state-maintenance computation by 76.38%.

AI USE STATEMENT

We used large language models (LLMs) to polish writing. Specifically, we use ChatGPT (OpenAI, GPT-6) to (i) polish grammar and phrasing of paragraphs drafted by the authors, and (ii) suggest alternative expressions for figure captions and section headings. The models did not generate novel technical ideas, algorithms, experimental designs, or results, all scientific content, claims, and analysis are created and verified by the authors.

REPRODUCIBILITY STATEMENT

The main text describes our LEAD-DVS dataset, Bench2Drive evaluation, and evaluation metrics. Baseline comparisons and ablation studies are presented with their experimental settings. Appendix A provides implementation details, training settings, and loss functions. Appendix B specifies the dataset splits, evaluation procedures, and computational measurement scope. We will provide the source code and download links for both datasets in the anonymous supplementary material to support reproduction of our experiments.

REFERENCES

Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. In *Conference on Robot Learning*, pp. 1–16, 2017.

Zhaoxin Fan, Tianbao Zhang, Wenjun Wu, Xiaofeng Wang, Yeying Jin, Jian Zhao, Zheng Zhu, and Shuicheng Yan. Drive-HWM: Hierarchical world models for dynamic-latent guided autonomous driving. *arXiv preprint arXiv:2609.03572*, 2026.

Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(1):154–180, 2022.

Yinfeng Gao, Qichao Zhang, Deqing Liu, Zhongpu Xia, Guang Li, Kun Ma, Guang Chen, Hangjun Ye, Long Chen, Da-Wei Ding, and Dongbin Zhao. Perlad: Towards enhanced closed-loop end-to-end autonomous driving with pseudo-simulation-based reinforcement learning. *IEEE Robotics Autom. Lett.*, 11(5):5821–5828, 2026. doi: 10.1109/LRA.2026.3675928. URL https://doi.org/10.1109/LRA.2026.3675928.

Daniel Gehrig and Davide Scaramuzza. Low-latency automotive vision with event cameras. *Nat.*, 629(8014):1034–1040, 2024.

Daniel Gehrig, Michelle Rüegg, Mathias Gehrig, Javier Hidalgo-Carrió, and Davide Scaramuzza. Combining events and frames using recurrent asynchronous multimodal networks for monocular depth prediction. *IEEE Robotics Autom. Lett.*, 6(2):2822–2829, 2021.

Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zachary Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. Model-based imitation learning for urban driving. In *Advances in Neural Information Processing Systems 35*, 2022.

Jinchao Hu, Zixian Wang, Xiaogang Wu, Yue Hu, Yiqiang Wu, and Hao Liu. Kinematic priors guide generative trajectory planning for autonomous vehicles. *Pattern Recognit.*, 175:113061, 2026.

Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. Planning-oriented autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17853–17862, 2023.

Xiaosong Jia, Zhenjie Yang, Qifeng Li, Zhiyuan Zhang, and Junchi Yan. Bench2drive: Towards multi-ability benchmarking of closed-loop end-to-end autonomous driving. In *Advances in Neural Information Processing Systems 37*, 2024.

Xiaosong Jia, Junqi You, Zhiyuan Zhang, and Junchi Yan. Drivetransformer: Unified transformer for scalable end-to-end autonomous driving. In *International Conference on Learning Representations*, 2025.

Dianze Li, Jianing Li, Xu Liu, Xiaopeng Fan, and Yonghong Tian. Asynchronous collaborative graph representation for frames and events. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1655–1666, 2025a.

Yingyan Li, Lue Fan, Jiawei He, Yuqi Wang, Yuntao Chen, Zhaoxiang Zhang, and Tieniu Tan. Enhancing end-to-end autonomous driving with latent world model. In *International Conference on Learning Representations*, 2025b.

Yingyan Li, Yuqi Wang, Yang Liu, Jiawei He, Lue Fan, and Zhaoxiang Zhang. End-to-end driving with online trajectory evaluation via BEV world model. In *IEEE/CVF International Conference on Computer Vision*, pp. 27137–27146, 2025c.

Zhenxin Li, Shihao Wang, Shiyi Lan, Zhiding Yu, Zuxuan Wu, and José M. Álvarez. Hydra-next: Robust closed-loop driving with open-loop training. In *IEEE/CVF International Conference on Computer Vision*, pp. 27305–27314, 2025d.

Bencheng Liao, Shaoyu Chen, Haoran Yin, Bo Jiang, Cheng Wang, Sixu Yan, Xinbang Zhang, Xiangyu Li, Ying Zhang, Qian Zhang, and Xinggang Wang. Diffusiondrive: Truncated diffusion model for end-to-end autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12037–12047, 2025.

Deqing Liu, Yinfeng Gao, Deheng Qian, Qichao Zhang, Xiaoqing Ye, Junyu Han, Yupeng Zheng, Xueyi Liu, Zhongpu Xia, Dawei Ding, Yifeng Pan, and Dongbin Zhao. Takead: Preference-based post-optimization for end-to-end autonomous driving with expert takeover data. *IEEE Robotics Autom. Lett.*, 11(2):1738–1745, 2026a. doi: 10.1109/LRA.2025.3643264. URL https://doi.org/10.1109/LRA.2025.3643264.

Haochen Liu, Tianyu Li, Haohan Yang, Li Chen, Caojun Wang, Ke Guo, Haochen Tian, Hongchen Li, Hongyang Li, and Chen Lv. Reinforced refinement with self-aware expansion for end-to-end autonomous driving. *IEEE Trans. Pattern Anal. Mach. Intell.*, 48(5):5774–5792, 2026b. doi: 10.1109/TPAMI.2026.3653866. URL https://doi.org/10.1109/TPAMI.2026.3653866.

Shu Liu, Wenlin Chen, Weihao Li, Zheng Wang, Lijin Yang, Jianing Huang, Yipin Zhang, Zhongzhan Huang, Ze Cheng, and Hao Yang. Bridgedrive: Diffusion bridge policy for closed-loop trajectory planning in autonomous driving. In *International Conference on Learning Representations*, 2026c.

Dongyue Lu, Rong Li, Ao Liang, Lingdong Kong, Wei Yin, Lai Xing Ng, Benoit R. Cottereau, Camille Simon Chane, and Wei Tsang Ooi. Eventdrive: Event cameras for vision-language driving intelligence. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22312–22322, 2026.

Long Nguyen, Micha Fauth, Bernhard Jaeger, Daniel Dauner, Maximilian Igl, Andreas Geiger, and Kashyap Chitta. Lead: Minimizing learner-expert asymmetry in end-to-end driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 39775–39785, 2026.

Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7077–7087, 2021.

Katrin Renz, Long Chen, Elahe Arani, and Oleg Sinavski. Simlingo: Vision-only closed-loop autonomous driving with language-action alignment. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pp. 11993–12003. Computer Vision Foundation / IEEE, 2025. doi: 10.1109/CVPR52734.2025.01120. URL https://openaccess.thecvf.com/content/CVPR2025/html/Renz_SimLingo_Vision-Only_Closed-Loop_Autonomous_Driving_with_Language-Action_Alignment_CVPR_2025_paper.html.

Simon Schaefer, Daniel Gehrig, and Davide Scaramuzza. AEGNN: asynchronous event-based graph neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12361–12371, 2022.

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

Ziying Song, Lin Liu, Hongyu Pan, Bencheng Liao, Mingzhe Guo, Lei Yang, Yongchang Zhang, Shaoqing Xu, Caiyan Jia, and Yadan Luo. Diver: Reinforced diffusion breaks imitation bottlenecks in end-to-end autonomous driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–17, 2026. doi: 10.1109/TPAMI.2026.3708096.

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. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14749–14759, 2024.

Jinqing Zhang, Zehua Fu, zelinxu, wenying.dai, Qingjie Liu, and Yunhong Wang. Resworld: Temporal residual world model for end-to-end autonomous driving. In *International Conference on Learning Representations*, pp. 6471–6483, 2026a.

Lingjun Zhang, Changjie Wu, Linzhe Shi, Jiangyang Li, Jiaxin Liu, Lei Yang, Hang Zhang, Mu Xu, and Hong Wang. Deepsight: Long-horizon world modeling via latent states prediction for end-to-end autonomous driving. In *International Conference on Machine Learning*, 2026b.

Zhiyuan Zhang, Zhenghao Jin, Yanlun Peng, Xianda Guo, Haoran Liu, Shaofeng Zhang, Xingjun Ma, Zuxuan Wu, Junchi Yan, Xiaosong Jia, and Yu-Gang Jiang. Bench2Drive-Robust: Benchmarking closed-loop autonomous driving under deployment perturbations. *arXiv preprint arXiv:2605.18059*, 2026c.

Wenzhao Zheng, Weiliang Chen, Yuanhui Huang, Borui Zhang, Yueqi Duan, and Jiwen Lu. Occworld: Learning a 3d occupancy world model for autonomous driving. In *Computer Vision – ECCV 2024*, pp. 55–72, 2024a.

Wenzhao Zheng, Ruiqi Song, Xianda Guo, Chenming Zhang, and Long Chen. Genad: Generative end-to-end autonomous driving. In *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part LXV*, volume 15123 of *Lecture Notes in Computer Science*, pp. 87–104. Springer, 2024b. doi: 10.1007/978-3-031-73650-6_6. URL https://doi.org/10.1007/978-3-031-73650-6_6.

Julian Zimmerlin, Jens Beißwenger, Bernhard Jaeger, Andreas Geiger, and Kashyap Chitta. Hidden biases of end-to-end driving datasets. *CoRR*, abs/2412.09602, 2024. doi: 10.48550/ARXIV.2412.09602. URL https://doi.org/10.48550/arXiv.2412.09602.

A Implementation Details

Coordinates. For a shared-map point (x, y), normalized sampling coordinates are

$$u = 2(x - x_0)/(x_1 - x_0) - 1, \qquad v = 2(y - y_0)/(y_1 - y_0) - 1.$$

Sampling uses `align_corners=False`. Occupancy labels and the shared state use different row and column conventions. Predictions are sampled at physical target-cell centers after transformation to the state anchor. Prediction and persistence are evaluated on identical valid support at each horizon.

A.1 Model Settings

Table 5 summarizes the configuration. Observation updates use index k and time t_k, distinct from diffusion time τ. Symbols M_k, U_k, and Z_k denote recurrent memory, normalized features, and the full state. E_k and $\mathcal{A}_k$ denote event evidence and graph edges. A^m, μ_k, and $m_k(q)$ denote path anchors, the selected anchor index, and event support.

Table 5: Settings for state updates and single-step planning.

Symbol	Setting
$\Delta t_E, \Delta t_C, \Delta t_R$	Event, context, and radar update intervals of 50, 250, and 250 ms
H_g, W_g	Shared grid of 32×40 cells with rows along y and columns along x
$(x_-, x_+), (y_-, y_+)$	$(-32, 64)$ and $(-40, 40)$ m, with cell sizes of 2.4 m along x and 2.5 m along y
d_C, d_E, d_Z	Feature dimensions of 1512, 256, and 1768, with $d_Z = d_C + d_E$
H_c, W_c	Native context and planner grids of 10×12, with native cell size 8 m
$\mathcal{D}_E$	Projected event support with $0 < x \leq 32$ m and $\lvert y \rvert \leq 20$ m
H_E, W_E, φ_E	DVS resolution of 768×768 and horizontal field of view 60°
$\mathbf{o}_{\text{cam}}$	DVS location $(0.35, 0, 2.25)$ m for the evaluated configuration
$(\delta_x, \delta_y, \delta_e)$	Graph bins of $(0.5\,\text{m}, 0.625\,\text{m}, 10\,\text{ms})$
$r_E, \delta_{\text{edge}}, K_E, N_E$	Edge radius 0.5 m, temporal lag 20 ms, at most 16 predecessors and 24,000 nodes
$L_g; (d_l)$	Three PLIF graph layers with widths $(64, 128, 256)$
L_B, ρ, ϵ_B	Evidence-bank lifetime 20 updates, decay 0.95, denominator floor 10^{-6}
N_s, r_s	Four alignment samples with a componentwise offset bound of 4 m
$\mathcal{H}_+$	Forecast horizons $\{0.5, 1.0\}$ s
v_{ref}	Predictor speed normalization of 30 m/s
N_a, N_p, N_v	60 path anchors, 10 spatial path points, and 8 speed classes
N_{solve}	One solver call per new plan
$\tau_{\max}$	Diffusion-time index 1000 at the anchor endpoint
$N_{\text{burn}}, N_{\text{seq}}$	10 burn-in updates and 20 supervised-unroll updates
$N_{\text{anchor}}, N_{\text{replay}}$	Reanchoring every 20 subsequent updates using 11 replay records, starting at update 30
$s_{\text{first}}, \delta_{\text{path}}$	Controller resampling starts at 2.5 m with spacing 1 m
N_{sem}	13 semantic classes, with semantic and occupancy grids of 320×384 and 64×64

A.2 Training Objectives

Training uses two NVIDIA RTX 4090 GPUs. The world-model objective in Section 3.4 is

$$\mathcal{L}_W = \lambda_f \mathcal{L}_f + \lambda_\Delta \mathcal{L}_\Delta + \mathcal{L}_{\text{aux}}, \qquad \mathcal{L}_{\text{aux}} = \lambda_c \mathcal{L}_c + \lambda_o \mathcal{L}_o + \lambda_s \mathcal{L}_s + \lambda_{\text{off}} \mathcal{L}_{\text{off}}. \tag{16}$$

For valid-support averaging $\langle\cdot\rangle_\Omega$, the per-instant losses are

$$\begin{aligned}
\ell^f_{k,h} &= \Big\langle \|\widehat{Z}_{k,h} - Y_{k,h}\|_2^2/d_Z \Big\rangle_{\Omega^f_{k,h}}, \\
\ell^c_{k,h} &= \frac{\Big\langle \|\widehat{C}_{k,h} - C^{\mathrm{nat}}_{k,h}\|_2^2/d_C \Big\rangle_{\Omega^c_{k,h}}}{\max\Big(\epsilon_c, \Big\langle \|C^{\mathrm{nat}}_{k,h}\|_2^2/d_C \Big\rangle_{\Omega^c_{k,h}}\Big)}, \\
\ell^\Delta_{k,h} &= \Big\langle \|(\widehat{Z}_{k,h} - Z_k) - (Y_{k,h} - Y_{k,0})\|_2^2/d_Z \Big\rangle_{\Omega^\Delta_{k,h}}, \\
\ell^o_{k,h} &= \mathrm{BCE}_{\Omega^o_{k,h},w_+}(\widehat{O}_{k,h}, O^\star_{k,h}) + \lambda_D\,\mathrm{DiceLoss}_{\Omega^o_{k,h}}(\widehat{O}_{k,h}, O^\star_{k,h}), \\
\ell^s_{k,h} &= \mathrm{CE}_{\Omega^s_{k,h}}(\widehat{S}_{k,h}, S^\star_{k,h}).
\end{aligned} \tag{17}$$

Here $\widehat{C}$ is the predicted context sampled in the target frame, $\widehat{O}$ contains occupancy probabilities, and $\widehat{S}$ contains semantic logits. Starred symbols denote labels. The offset regularizer $\mathcal{L}_{\mathrm{off}}$ penalizes squared alignment offsets.

Teacher targets concatenate frozen context features and their fixed projected, normalized features. Target poses align these features for supervision and are not used during inference. Each loss is first averaged over valid cells and channels. Let $\mathcal{T}_{r,h}$ contain the labeled instants with valid support for loss family r at horizon h. The reduction is

$$\mathcal{L}_r = \frac{\sum_{h\in\mathcal{H}_r} \omega_h I_{r,h}\,\mathrm{Avg}_{k\in\mathcal{T}_{r,h}}\,\ell^r_{k,h}}{\sum_{h\in\mathcal{H}_r}\omega_h I_{r,h}}, \qquad I_{r,h} = \mathbf{1}[\mathcal{T}_{r,h}\neq\varnothing]. \tag{18}$$

A loss family with no valid support contributes zero. We use $\mathcal{H}_\Delta = \mathcal{H}_+$, while the other prediction losses also include the current horizon. Occupancy labels encode the annotated boxes of non-ego actors. Occupancy loss combines positive-weighted BCE and masked Dice loss. Semantic CE excludes out-of-anchor and visibility-invalid pixels.

The planning objective is

$$\mathcal{L}_P = \frac{\mathcal{L}_{\mathrm{path}} + \mathcal{L}_{\mathrm{mode}} + \mathcal{L}_{\mathrm{wp}} + \mathcal{L}_{\mathrm{speed}}}{N_{\mathrm{pl}}}, \qquad \mathcal{L}_{\mathrm{speed}} = \mathrm{CE}(\mathbf{p}^v_k, \mathbf{y}^v_k). \tag{19}$$

For expert path $\Pi^\star_k$, the target anchor and mode loss are

$$\mu^\star_k = \arg\min_m \frac{1}{N_p}\sum_{n=1}^{N_p}\|\Pi^\star_{k,n} - A^m_n\|_2, \qquad \mathcal{L}_{\mathrm{mode}} = \sum_{j=1}^{J_{\mathrm{cls}}}\mathrm{FL}(\mathbf{c}_{k,j}, \mathbf{e}_{\mu^\star_k}), \tag{20}$$

where FL is sigmoid focal loss and $\mathbf{e}_{\mu^\star_k}$ is one-hot. Path regression is accumulated once per classification-stage output, giving

$$\mathcal{L}_{\mathrm{path}} = J_{\mathrm{cls}}\,\mathrm{MeanAbs}(\widehat{\Pi}^{\mathrm{train}}_k - \Pi^\star_k), \qquad \mathcal{L}_{\mathrm{wp}} = \mathrm{MeanAbs}(\widehat{W}_k - W^\star_k). \tag{21}$$

Diffusion training corrupts expert paths toward fixed anchors at sampled diffusion times. The decoded path $\widehat{\Pi}^{\mathrm{train}}$ is selected by predicted anchor scores. The auxiliary temporal-waypoint head $\widehat{W}$ is distinct from the executed spatial path. Speed targets distribute probability over the two neighboring speed classes or use the stop class under expert braking. Table 6 lists the coefficients.

A.3 Online Execution

The spiking encoder processes each complete 50 ms event graph and resets its membrane state between packets. The BEV recurrent state persists across packets and is updated in affected cells. Forecasting uses an ANN head, and state updates and planning execute serially. Thus, incremental processing refers to state maintenance, not node-wise incremental execution of the graph encoder. Graph construction, evidence replacement, and reanchoring follow Section 3.1 and Algorithm 1.

Table 6: Loss coefficients and normalization settings.

Symbol	Setting
$(\omega_0, \omega_{0.5}, \omega_{1.0})$	$(0.50, 0.35, 0.15)$, renormalized over supported horizons
$(\lambda_f, \lambda_c, \lambda_o, \lambda_s)$	$(1, 0.5, 1, 1)$
$\lambda_\Delta, \lambda_{\text{off}}, \lambda_P$	$0.25, 10^{-4}, 1$
w_+, λ_D	Occupancy positive weight 15 and Dice weight 0.1
$\epsilon_c, N_{\text{pl}}$	Reconstruction-energy floor 10^{-3} and four planning losses
$\alpha_{\text{FL}}, \gamma_{\text{FL}}$	Sigmoid focal-loss parameters 0.25 and 2
J_{cls}	Two classification-stage outputs

B Evaluation Details

B.1 Data and Statistics

LEAD-DVS contains 1,579 routes, split into 935 training, 105 validation, and 539 test routes. Model training, including pretraining and subsequent optimization on LEAD-DVS, uses only the training split. Recorded-input prediction, planning-error, input-perturbation, and runtime evaluations draw their panels from the 539-route test split. These test routes do not participate in model training. The separately collected PDM-Lite-DVS dataset is not used to train the reported SwiftWM models.

Bench2Drive provides a separate set of 220 routes for closed-loop evaluation. These routes are not used for training. This benchmark evaluation is distinct from the recorded-input measurements on the LEAD-DVS test split.

Full and no-DVS models. The SwiftWM and SwiftWM w/o DVS models in Table 4 are trained separately on the same LEAD-DVS training split, with identical epoch counts and experiment settings apart from the DVS modality. The no-DVS variant is trained without event input and evaluated without a DVS camera. It is distinct from the inference-time event-withholding control used in the occupancy analysis.

Full/Current comparisons use two training runs with shared initialization and data order within each pair. Reported intervals use 10,000 paired group-bootstrap resamples and are pointwise 95% intervals. Repeated timing runs are not counted as additional independent accuracy samples.

B.2 Forecast Quality

Figure 4 compares learned forecasts with transported current-state persistence against future labels from the LEAD-DVS test split on identical support. Future poses define evaluation coordinates only. Semantic mIoU averages the classes present in each group's valid support, then the 32 groups. AP uses identical score-tie handling. One group has no positive occupancy label, leaving 31 groups with defined AP. Current's auxiliary forecast head is evaluated here but is not used by its deployed planner.

For Full, semantic mIoU at 0.5 and 1.0 s is 70.72% and 68.15%, versus 62.23% and 59.24% for persistence. Occupancy AP is 88.87% and 80.82%, versus 46.89% and 33.85%. Current shows similar forecast quality. Results use one matched training pair, with group means and pointwise 95% bootstrap intervals.

B.3 Event-Input and Encoder Analysis

Occupancy prediction. Tables 7 and 8 use one checkpoint on 24 LEAD-DVS test routes and 96 labels, separate from the 32-group comparison. Occupancy AP is evaluated in physical coordinates with future poses used only for target alignment. Table 7 compares the full model, persistence, and inference-time DVS withholding, alongside front-region AP. DVS withholding causes small AP changes and does not involve retraining.

Table 7: Occupancy AP (%) on 24 test routes.

Horizon	Full	Persistence	DVS withheld	Front
0.0 s	93.88	93.88	92.99	94.00
0.5 s	91.59	67.63	90.62	91.70
1.0 s	85.23	50.23	84.40	84.83

Event timing. Table 8 tests event delays of 5 and 20 ms, timestamp quantization at 5 ms, and deterministic removal of 25% of events. Delays restrict input to older events, and quantization reduces within-packet time resolution. AP changes little under these input perturbations. Rendering frequency, simulator physics, and sensor contrast thresholds remain fixed.

Table 8: Occupancy AP (%) under event-input perturbations and inference-only membrane carry.

Setting	0 s	0.5 s	1.0 s
Unperturbed	93.88	91.59	85.23
5 ms delay	93.88	91.59	85.24
20 ms delay	93.94	91.63	85.28
5 ms quantization	93.91	91.60	85.29
25% event removal	93.84	91.59	85.25
Membrane carry	93.84	91.54	85.26

Membrane state. The final row applies membrane carry to reset-trained weights without retraining. Each PLIF layer stores memory by fixed-anchor BEV cell, averaging updated node membranes within each cell. Unobserved cells decay by $(1-\beta)^k$ over k intervals, and new cells start at zero. Non-data nodes do not write memory. Packets follow time order, and memory is cleared or remapped at stream or anchor changes. Carry gives similar AP on 24 sequences and 720 packets. The main encoder retains packet-wise resets.

Graph encoding cost. Packed-bin construction reduces mean graph-construction, transfer, and encoder time from 47.32 to 17.37 ms on eight high-event-count test routes, two anchor transforms, and five repetitions. Inputs and weights are fixed. Graph tensors match exactly and features match numerically. The change affects graph construction, not the spiking network. Timings use sampled recordings and exclude downstream planning.

Observation ages. Reconstructing the pretraining pipeline over 59,700 sequences and 31 epochs yields 7,327,625 supervised instants. Only 82 have BEV age at least 500 ms and none reaches 1,000 ms. Radar age is zero at labels and at most 200 ms across updates. BEV dropout is sampled with probability 0.15 every 50 ms, with no radar dropout. Sustained gaps are therefore poorly represented. These counts exclude distributed-training padding, earlier pipeline versions, and the additional training epoch.

B.4 Runtime Measurements

Tables 1 and 2 compare 20, 5, and 1 solver steps with fixed model weights. Every eligible update generates a new path. The separate speed head produces identical logits across solver budgets on fixed inputs.

Planner runtime. Table 1 uses 95 LEAD-DVS test routes and 380 recorded frames. Timing covers forecast fusion and the planning decoder, including timing-hook overhead. It excludes perception, graph construction, state updates, input transfer, sensor acquisition, control, and queueing. The actual solver-call count is checked for each budget. Results by driving condition appear in Appendix C.4.

Policy runtime. Table 2 uses ten scenes from the LEAD-DVS test split, batch size one, and one RTX 3090. Three repetitions produce 6,000 timed updates per budget, while planning errors use

400 distinct labeled frames. Execution order rotates across budgets with paired randomization and fixed cuDNN settings. GPU access is exclusive, but CPU resources are not fully isolated. Timing includes the backbone, state updates, planning, controller, and trace writing. It excludes record decoding, sensor acquisition, localization filtering, and force-move postprocessing. Recorded DVS packets are sampled, and the LiDAR input contains non-ground points. Nested component timings are not summed.

Including recorded-input preparation increases F1's mean service time from 60.65 to 68.85 ms. Without that preparation, 33.7% of updates exceed 50 ms. These measurements quantify policy computation within the timing scope defined above.

B.5 Vehicle Cut-In from the Right

Figure 5 aligns a Town05 cut-in to lateral-motion onset. At $t = -0.2$ s, a partial vehicle contour appears at the event-view boundary. SwiftWM has been braking since -0.30 s in the illustrated run, with brake input 1.0 at -0.2 s and a three-run mean of $2/3$.

The uncropped SwiftWM RGB frame also contains part of the vehicle at -0.2 s. Camera views and crops differ, and the baselines also brake before onset. This is a qualitative driving example, not a matched detection or causal comparison. Images show one run and curves show the mean and range across three runs on the measured time axis. SwiftWM uses single-step planning.

C Supplementary Results

C.1 Future-Latent Intervention in Closed-Loop Driving

We examine how explicit future latents affect driving behavior through an inference-time intervention on SwiftWM. Both conditions use identical model weights: one retains the predicted future latent maps supplied to the policy, while the other replaces them with the current latent map. This same-weight intervention complements the paired-training Full/Current comparison described in Appendix B by changing the policy's predictive input without retraining. Figure 6 shows the resulting speed and braking traces.

Retaining future latents yields collision-free completions in *HazardAtSideLane* and *ConstructionObstacleTwoWays*, whereas replacing them with the current latent leads to a vehicle collision and a cone collision, respectively. Driving scores increase from 60 to 100 and from 65 to 100. In *ParkingCutIn*, both conditions complete without collisions and achieve a driving score of 100, while retaining future latents reduces completion time from 61.80 to 41.45 s, a 32.9% reduction. The safety benefit involves a time trade-off in *ConstructionObstacleTwoWays*: the collision-free run takes 136.40 s, an increase of 20.30 s over the 116.10 s run with current-latent replacement.

These results demonstrate that explicit future latents improve closed-loop driving performance by supporting collision avoidance and efficient task completion. Together with the forecast-quality results in Appendix B.2, the intervention connects predictive scene representations to improved driving behavior.

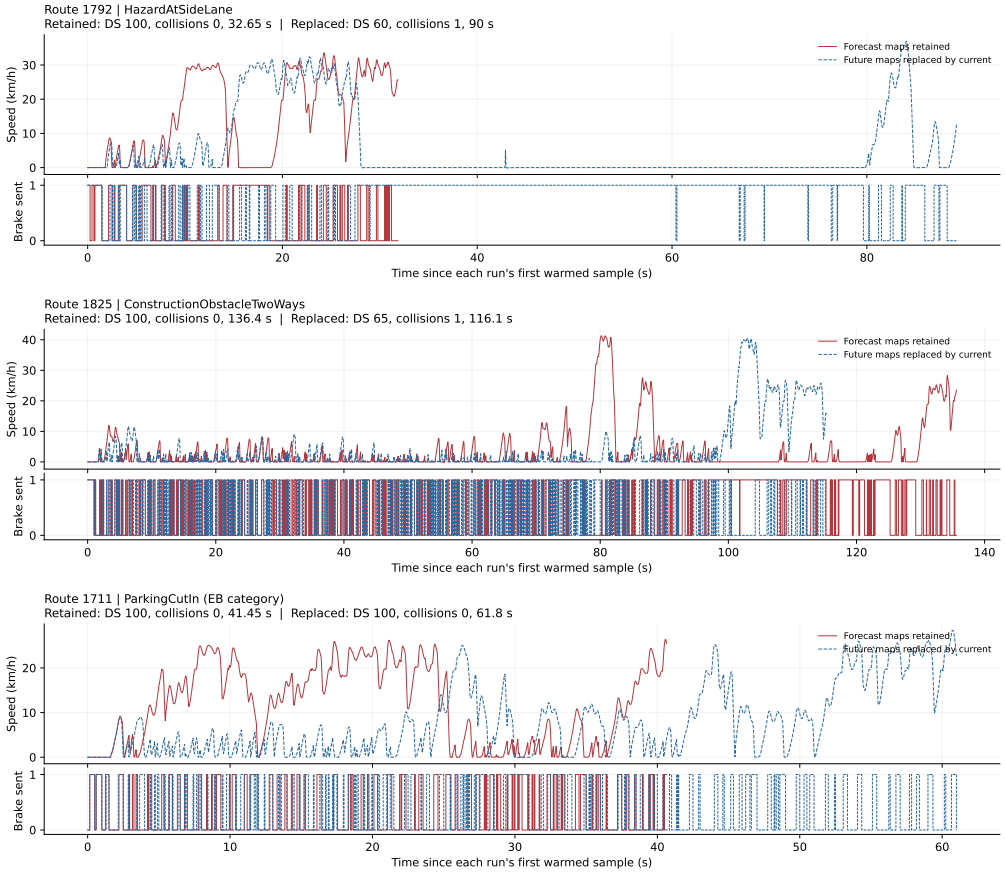

Figure 6: Closed-loop intervention on future latents with fixed SwiftWM weights. Red solid curves retain predicted future latent maps; blue dashed curves replace them with the current latent map. Each scenario shows speed and commanded braking, with time measured from each run's first post-warm-up sample. DS denotes driving score; headings report scores, collision counts, and run durations.

C.2 Path Predictions

Figure 7 shows eight recorded examples each for straight driving, turning, and intersections, selected according to scene geometry, independently of prediction error. Red dots mark ten predicted spatial waypoints. Recorded maps and actors are display-only context. Camera overlays assume flat ground. Coral, blue, and ochre denote the ego vehicle, other vehicles, and static actors or pedestrians, respectively.

C.3 Turning Comparisons

Figure 8 compares predictions at eight junctions with four left and four right turns. Scenes and frames are selected by junction geometry and a fixed turn-phase rule. Each method processes its native sensor subset from the same recorded drive, using its own calibration, preprocessing, and inference cadence. Rows share the physical timestamp, ego pose, and BEV extent. Map and actor geometry are used for display only.

DriveTransformer outputs 20 spatial points, Hydra-NeXt's trajectory branch outputs six points, and SwiftWM outputs ten points using its single-step planner. Native point counts and horizons are retained. The predictions do not control the recorded vehicle, and Hydra-NeXt's trajectory branch is not the executed output of its combined control branches. The comparison characterizes planning outputs on common recordings with each method's native horizon.

C.4 Solver Steps

Table 9 reports results across the predefined driving conditions. Reduced-step settings satisfy the ADE and speed-CE screening thresholds of 1.05 times the corresponding 20-step route mean plus 10^{-4}, and preserve the stop decisions. These thresholds serve as engineering screening criteria; they do not establish statistical equivalence. Identical speed logits follow from the separate speed head.

Table 9: Recorded-input errors across solver budgets. Parentheses give the nominal publication interval.

Cohort	ADE_{20}	ADE_5	ADE_1	Shared CE
Stop hold (250 ms)	0.0489	0.0440	0.0323	0.0097
Normal (250 ms)	0.0214	0.0216	0.0173	0.4814
Normal (500 ms)	0.0353	0.0347	0.0281	0.4902
Normal (1000 ms)	0.1473	0.1435	0.1380	0.5978
Brake onset (250 ms)	0.0228	0.0227	0.0199	0.1906
Brake onset (500 ms)	0.0409	0.0390	0.0323	0.4334
Brake onset (1000 ms)	0.2384	0.2265	0.2057	0.5916
Release (250 ms)	0.0498	0.0467	0.0380	0.5563
Route-uniform (250 ms)	0.0264	0.0256	0.0219	0.2573

Table 10 reports GPU timings on the nominal route-uniform panel. Solver order rotates per frame, with two warmup sweeps excluded and one timed call per budget. GPU clocks are not locked. Stage timings are nested, not sensor-to-action latency. Outputs were checked against the stored 20-step reference.

Table 10: Mean GPU stage durations (ms) by solver budget.

Steps	Context encoding	Query transformer	Trajectory head
20	0.495	3.877	132.909
5	0.509	3.893	35.422
1	0.505	3.885	9.402



(a) Go straight (b) Turning (c) Intersection

Figure 7: Single-step path predictions for straight driving, turning, and intersections. Each column contains eight examples. Red dots denote predicted spatial waypoints, and each BEV view includes a 10 m scale bar.

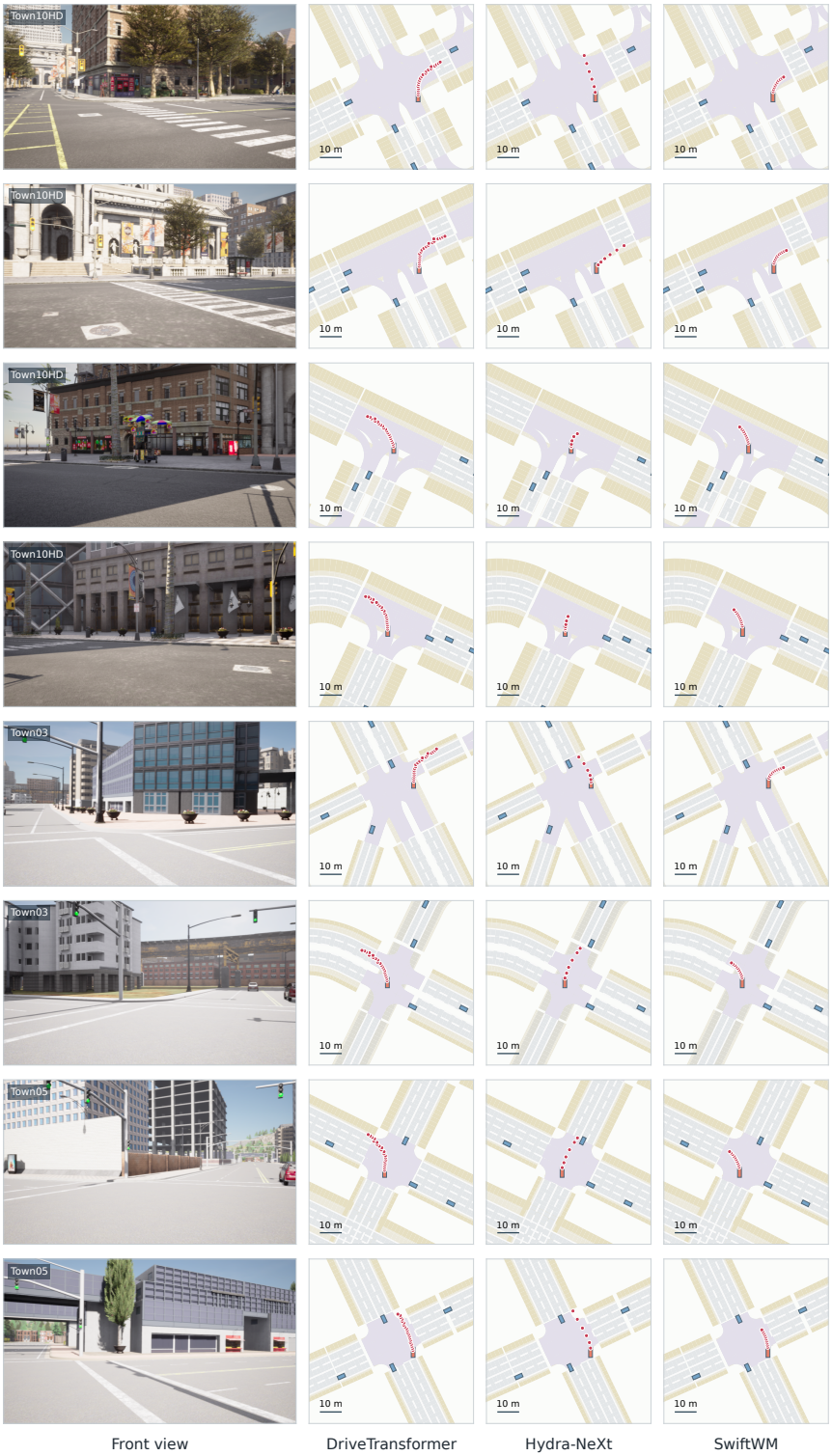

Figure 8: Predictions at eight turning junctions. Columns show the recorded front view, DriveTransformer, Hydra-NeXt's trajectory branch, and SwiftWM. Red dots retain each model's native points and horizon. Each BEV view includes a 10 m scale bar.