MEMORIZE TO ACCELERATE: BEV ACTION ENGRAM FOR EFFICIENT DIFFUSION-BASED VLA PLANNING IN AUTONOMOUS DRIVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion-based vision-language-action (VLA) planners generate driving trajectories through iterative denoising, but aggressive reduction of the denoising budget degrades planning quality. We observe that each step re-derives two kinds of reusable structure: short-horizon motion primitives that recur across scenes, and refinement dynamics that recur across sampling steps; the Transformer stores only these two in its weights. This redundancy motivates a process-level memory, *BEV Action Engram*, that supplies scene-conditioned action embeddings during decoding. Its two branches address the two redundancies in turn: a Spatial branch maps local motions to BEV displacement n-grams and addresses a learnable embedding table via cross-modal condition hashing for geometric disambiguation, while a CrossStep branch captures the shared refinement dynamics across sampling steps; the resulting features are injected into intermediate layers through gated residuals. In addition, we adapt flow matching group relative policy optimization to this planner and study the memory under reinforcement fine-tuning. Experiments on the NAVSIM dataset show that BEV Action Engram can be learned jointly with the decoder under imitation learning and reused as an action prior during reinforcement fine-tuning. At matched planning quality, the planner uses 40% fewer sampling steps, about 38% lower planner FLOPs, and about 14% lower planner wall-clock, improving the step–compute–accuracy trade-off of the generative decoder. When the decoder is fine-tuned under the same EPDMS reward, removing the memory raises ego progress while the collision-related factors drop and the aggregate score falls below its imitation-learning starting point, whereas retaining the memory restores the safety factors and reaches 88.9 EPDMS on NAVSIMv2, and 89.6 PDMS on NAVSIMv1.

1 INTRODUCTION

End-to-end autonomous driving has increasingly converged on vision-language-action (VLA) systems, where a vision-language model (VLM) perceives the scene and interprets navigation intent, and an action decoder generates future trajectories Jiang et al. (2025b); Hu et al. (2025); Yang et al. (2025). Among decoder designs, generative planners based on diffusion and flow-matching are widely used because multi-step sampling captures the multi-modality and uncertainty of driving behavior Liao et al. (2025); Li et al. (2026b); Jiang et al. (2025a). A common approach is to use the VLM with a flow-matching DiT action decoder Li et al. (2026b;a). In both settings, trajectories are produced by multi-step iterative sampling, and reducing the step budget is a primary lever for faster planning.

Step size reduction and distillation Liao et al. (2025); Zou et al. (2025) trade accuracy for speed; trajectory templates and clustered anchors Jiang et al. (2026b); Yin et al. (2026) improve only the starting point of generation; retrieval from external databases adds noise and latency; and latent or slot-based memories Behrouz et al. (2026) lack a BEV geometric bias for disambiguating motions that share geometry yet differ in semantics. None of these lines explicitly targets the computation that is repeated inside each sampling step.

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.

We observe that the bottleneck is process-level rather than purely schedule-level, and that it has two complementary sources. First, successive steps recompute short-horizon motion structure that recurs across scenes and is therefore reusable. Second, the denoising process follows refinement dynamics that recur across sampling steps and are largely shared regardless of the scene. Standard Transformers fold both kinds of structure only into weights and must re-derive them step by step, leaving limited room to cut the step budget without sacrificing planning quality.

Language-modeling work on Engram-style memory offers a complementary mechanism: static textual n-gram patterns are served by asymptotically $\mathcal{O}(1)$ embedding lookup, relieving the backbone from repeated neural reconstruction Cheng et al. (2026). Migrating this idea to the action modality, we introduce ***BEV Action Engram***, a process-level memory embedded in the action decoder: a learnable lookup table addressed by $\mathcal{O}(1)$ hashing and trained jointly with the decoder, so that recurring structure is served in-weight during denoising rather than retrieved from an external database or summarized in a recurrent state. We refer to this memory as "conditional memory" because cross-modal condition hashing makes retrieval subject to the driving context, so that the same geometric structure retrieves different priors in different scenes. Concretely, continuous trajectories are mapped to position-independent BEV displacement n-grams, and a dual memory serves the two redundancies: a Spatial branch for recurring trajectory geometry and a CrossStep branch for the refinement dynamics shared across sampling steps, both under a consistent train–inference interface with delayed injection.

To evaluate the memory beyond imitation learning, we further adapt Flow-GRPO Liu et al. (2026); He et al. (2025) to our flow-matching planner with BEV Action Engram for reinforcement learning. Reward optimization at this level reveals a further role of the memory: under the same EPDMS reward, fine-tuning without the memory raises ego progress while the sparse safety factors drop and the aggregate score falls below the IL starting point, whereas keeping the Engram active as a frozen prior restores the safety factors and raises the score. On NAVSIM Dauner et al. (2024), the memory-augmented planner attains matched quality with 40% fewer sampling steps and about 38% lower planner FLOPs (about 14% lower planner wall-clock despite a small per-step lookup overhead), improving the step–compute–accuracy trade-off of the decoder, and reaches 89.6 PDMS on NAVSIMv1 and 88.9 EPDMS on NAVSIMv2. Ablations across memory branches, injection depth, and capacity further characterize how the action memory operates.

Our main contributions are summarized as follows:

1. We migrate Engram-style embedded conditional memory to the driving-action modality: recurring local motions are represented as BEV displacement n-grams and served by asymptotically $\mathcal{O}(1)$ embedding lookup, so a generative VLA planner reuses short-horizon motion structure instead of recomputing it at every sampling step.

2. We design a dual memory for Action Engram: a Spatial branch conditioned on scene context via cross-modal condition hashing, and a CrossStep branch for the dynamics shared across sampling steps. Both are written into the decoder through delayed injection and layer-wise gated residuals that keep training and inference consistent.

3. Beyond imitation learning, we adapt Flow-GRPO to flow-matching VLA planners with BEV Action Engram. Under the same EPDMS reward, fine-tuning without the memory degrades the safety factors, whereas retaining the frozen memory restores them and improves the overall score, improving the step–compute–accuracy trade-off.

2 Related Work

2.1 End-to-End Autonomous Driving

End-to-end autonomous driving replaces modular perception–prediction–planning pipelines with a unified model that maps raw sensor inputs to future trajectories Hu et al. (2023); Jiang et al. (2023); Chitta et al. (2022); Jia et al. (2025), yet such models often offer limited interpretability and generalize poorly to long-tail scenarios. VLA methods address these weaknesses by introducing language reasoning, but their large model size raises real-time concerns. Text-based autoregressive VLA planners generate action tokens one by one Tian et al. (2025); Zhou et al. (2026b;a), which is too slow for online deployment. Hybrid designs that attach a diffusion action decoder to a VLM still

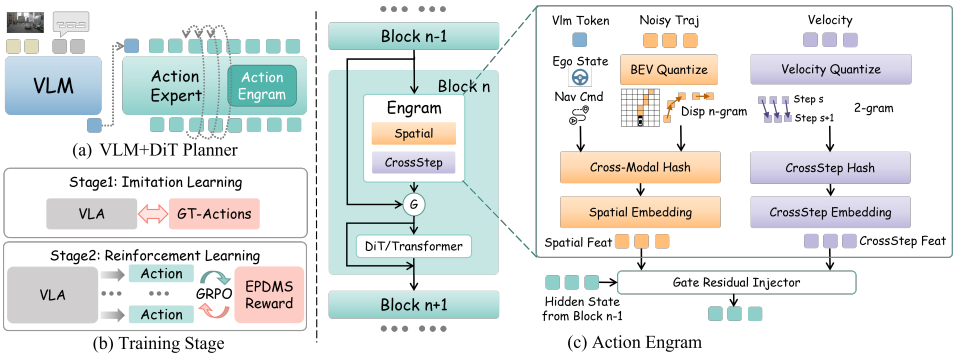


Figure 1: Overview of BEV Action Engram. Selected DiT blocks are augmented with a dual-branch Engram: Spatial memory retrieves scene-conditioned BEV displacement embeddings via cross-modal hashing, and CrossStep memory encodes delayed velocity dynamics. A gate residual injector G writes both embeddings into the block stream.

require multi-step sampling Fu et al. (2025); Li et al. (2026a;b), and recent unified diffusion VLAs decode actions as discrete tokens Li et al. (2025a); Xu et al. (2025); Ma et al. (2026); Xu et al. (2026). Combining VLA perception with generative decoding is a promising route toward real-time autonomous driving, and our work augments existing generative action decoders with a reusable conditional action memory that improves the step–compute trade-off of multi-step sampling.

2.2 Efficient Diffusion and Conditional Memory

In the VLM plus generative decoder paradigm, a semantic gap remains between high-capacity VLM conditioning and low-level trajectory generation. Prior acceleration mainly reshapes the sampling schedule or origin—motion priors, geometric constraints, flow matching, and truncated or staged diffusion Liao et al. (2025); Jiang et al. (2026b; 2025a); Yin et al. (2026); Zou et al. (2025)—but leaves intermediate decoder layers unchanged; we instead inject memory into those layers at each sampling step.

Complementary work attaches memory to reuse history, yet with different storage and interfaces. Robotics VLAs favor implicit banks: MemoryVLA Shi et al. (2025) fuses cross-attention reads of perceptual history into a diffusion action expert, and EvoVLA Liu et al. (2025) keeps Top-K history as context tokens; driving systems more often store explicit knowledge as LLM/VLM context Mei et al. (2024); Ma et al. (2025); Yuan et al. (2024); Chang et al. (2026). Latent compressors such as Titans Behrouz et al. (2026) summarize history with learnable queries but lack a BEV geometric bias for motion. Across these lines, memory mainly supplies scene or expert context to a language backbone, at a retrieval or concatenation cost; in contrast, Engram serves static textual n-grams by asymptotically $\mathcal{O}(1)$ embedding lookup Cheng et al. (2026), and we migrate this mechanism to the action modality, so that recurring BEV motion structure is hashed and injected during multi-step decoding.

3 Method

3.1 Framework Overview

We present *BEV Action Engram*, a conditional action memory embedded in the intermediate layers of a generative action decoder (Fig. 1). At each sampling step, selected DiT blocks are augmented with an Engram module whose Spatial and CrossStep branches produce scene-conditioned and cross-step embeddings, written back through a gated residual injector G.

3.2 Preliminaries: VLA Planner Baseline

Following prior VLM–DiT planners Li et al. (2026b), we use a frozen InternVL3-2B Zhu et al. (2025) VLM backbone and a DiT action decoder that generates trajectories by flow matching Peebles & Xie (2023); Lipman et al. (2023). Given a front-view image, a navigation command, ego states,

and a short history, the VLM produces conditioning features c. The DiT then generates the future trajectory $\tau = \{A_1, \ldots, A_H\}$, $A_t = (x_t, y_t, \theta_t)$, by iterative flow-matching sampling: starting from a noisy trajectory, the decoder predicts a velocity field at each sampling step and integrates it toward the data manifold. At sampling step s (out of K flow-matching sampling steps in total), a stack of L DiT blocks maps the current trajectory state and c to a velocity field $\hat{V}_s \in \mathbb{R}^{H\times 3}$ that updates the trajectory; the trajectory state and c are first encoded into an initial hidden state $h^{(0)}$, which the L blocks then refine. We represent the hidden state at DiT block ℓ as $h^{(\ell)} \in \mathbb{R}^{H\times d}$, where the H rows correspond to the H waypoints $A_1, \ldots, A_H$. Throughout, τ denotes the trajectory; we reserve t for the flow-matching time index used in training.

3.3 BEV Action Discretization and Spatial Engram

Displacement tokens. Instead of absolute waypoints, we represent actions by *relative* displacements, which are position-independent and thus reusable across scenes:

$$\Delta A_t = A_{t+1} - A_t = (\Delta x_t, \Delta y_t, \Delta\theta_t). \tag{1}$$

The displacement sequence of the trajectory τ_s at sampling step s is $\Delta\tau_s = \{\Delta A_t\}_{t=1}^{H-1}$, indexed by $t \in \{1, \ldots, H-1\}$. Each displacement is quantized on a bird's-eye-view (BEV) grid with a heading bin to a discrete identifier:

$$\mathrm{id}_t = \mathrm{Quant}_{\mathrm{BEV}}(\Delta A_t) \in \{1, \ldots, V_d\}, \tag{2}$$

where V_d is the BEV displacement vocabulary size. This yields a sequence of displacement tokens $(\mathrm{id}_1, \ldots, \mathrm{id}_{H-1})$ that captures how the vehicle moves in local geometry.

Displacement n-grams. Following the intuition of language n-grams, we form local motion primitives from consecutive displacements. For each waypoint h, the displacement n-gram

$$g_h^{(n)} = (\mathrm{id}_{h-n+1}, \ldots, \mathrm{id}_h), \quad n \in \{2, 3\}, \tag{3}$$

collects the n displacements leading into h (e.g., "straight-then-slight-right").

Multi-head hashed lookup. The space of n-grams is combinatorially large and extremely sparse, so we avoid an explicit vocabulary and instead map each window to a fixed-size, learnable Spatial memory table set $E^{\mathrm{sp}} = \{E_i^{\mathrm{sp}} \in \mathbb{R}^{N_{\mathrm{slot}}\times d_e}\}_{i=1}^{m_{\mathrm{sp}}}$ via multi-head hashing, where N_{slot} is the number of slots per head, d_e the per-head embedding width, and the m_{sp} heads span the two orders (four heads per order). For head i and waypoint h,

$$\mathrm{slot}_{i,h} = \mathrm{hash}_i\big(g_h^{(n)} \oplus k_c\big) \bmod N_{\mathrm{slot}}, \quad i = 1, \ldots, m_{\mathrm{sp}}, \tag{4}$$

where k_c is the condition key defined by cross-modal condition features, $\oplus$ denotes key composition, and waypoints with fewer than n leading displacements read a dedicated null entry. The Spatial memory feature concatenates the per-head embeddings,

$$z_h^{\mathrm{sp}} = \big[\, E_1^{\mathrm{sp}}[\mathrm{slot}_{1,h}] \,\|\, \cdots \,\|\, E_{m_{\mathrm{sp}}}^{\mathrm{sp}}[\mathrm{slot}_{m_{\mathrm{sp}},h}] \,\big] \in \mathbb{R}^{m_{\mathrm{sp}} d_e}, \tag{5}$$

a length-H sequence over the waypoints, like the CrossStep feature. Multi-head hashing reduces collision ambiguity: two n-grams colliding under one hash are unlikely to collide under all, so their concatenated representations remain distinguishable Cheng et al. (2026). Retrieval is deterministic and asymptotically $\mathcal{O}(1)$.

3.4 Cross-Modal Condition Hashing

A purely geometric displacement is ambiguous: the same leftward motion may correspond to a turn, an obstacle-avoidance maneuver, or a lane change. We therefore condition retrieval on the driving context. We decompose the conditioning features c into four modalities—navigation command c_{nav}, VLM hidden states c_{vlm}, ego state c_{ego}, and history c_{hist}. Continuous features (VLM hidden states, history) are discretized with locality-sensitive hashing (LSH) Datar et al. (2004), while discrete signals (navigation command, ego state) are directly quantized; the ego state is an eight-component

status vector (navigation one-hot, ego velocity, ego acceleration). Each modality is independently hashed or quantized, and the results are fused by multiply–XOR into a single condition key:

$$k_c = \big(q(c_{\text{nav}})\cdot w_0\big) \oplus \big(\text{LSH}(c_{\text{vlm}})\cdot w_1\big) \\ \oplus \Big(\bigoplus_{i=1}^{8} q_i(c_{\text{ego}})\cdot w_{2+(i \bmod 4)}\Big) \oplus \big(\text{LSH}(c_{\text{hist}})\cdot w_6\big), \tag{6}$$

where $\oplus$ denotes bitwise XOR, $w_0, \ldots, w_6$ are fixed random odd multipliers, and $q(\cdot)$, $q_i(\cdot)$ denote direct quantization of the navigation command and of the ego-status components. The condition key k_c is then composed with the displacement n-gram and injected into the Spatial slot computation in Eq. (4), so that identical displacement n-grams under different scenes are routed to different memory slots and thus retrieve different priors. Because LSH maps semantically similar contexts to the same bucket with high probability, related scenes can share memory while dissimilar scenes are disambiguated, which is what makes the memory conditional rather than a static template. Hash bit widths, multipliers, and slot settings are given in App. 2.1.

3.5 CrossStep Engram with Delayed Injection

We introduce a *CrossStep* memory whose keys are formed from the flow-matching velocity fields predicted at previous sampling steps. Like the Spatial branch, it is indexed per position: each row $\hat{V}_s[h] \in \mathbb{R}^3$ of a predicted velocity field is uniformly quantized, and its three digits are packed by mixed radix into a per-waypoint identifier

$$\text{vid}_{s,h} = \text{Quant}_{\text{vel}}\big(\hat{V}_s[h]\big) \in \big\{0, \ldots, B^3 - 1\big\}, \tag{7}$$

where B is the number of quantization bins per velocity component (App. 1.2). For waypoint h and hash head i, the identifiers of the two most recent steps are fused with the discretized flow-matching time by multiply–XOR and reduced modulo N_{slot} into per-waypoint, per-head slot indices of a dedicated table set $E^{\text{cs}} = \{E_i^{\text{cs}} \in \mathbb{R}^{N_{\text{slot}} \times d_e}\}_{i=1}^{m_{\text{cs}}}$; the step order is carried by per-position multipliers, the time term lets the same velocity pair address different slots at different denoising stages, and, unlike the Spatial branch, no cross-modal condition key is used, so the branch captures generation dynamics shared across scenes. The per-head embeddings are concatenated into $z_{s,h}^{\text{cs}} \in \mathbb{R}^{m_{\text{cs}} d_e}$, so the CrossStep feature is a length-H sequence over the waypoints, like the Spatial feature. We use the two most recent steps as a compact key over short-range refinement between consecutive predicted velocities; the table sets E^{sp} and E^{cs} are disjoint and learned independently.

Delayed injection. A naive design would hash the current noisy action, using ground-truth interpolation during training but model outputs at inference, which causes a train–inference mismatch. Instead, our *delayed injection* uses only already-predicted velocity fields from prior steps at both training and inference, obtained from a K-step training rollout (App. 1.2), and falls back to a dedicated null entry of E^{cs} whenever fewer than two prior velocity fields are available (steps $s < 2$). This keeps the memory interface identical across the two regimes.

3.6 Layer-wise Gated Injection

We inject the memory embedding into selected DiT blocks through gated residuals. For an injection layer $\ell \in \mathcal{I}$,

$$h^{(\ell)} \leftarrow h^{(\ell)} + G\big(h^{(\ell)}, z^{\text{sp}}, z^{\text{cs}}\big), \tag{8}$$

where $z^{\text{sp}} = \{z_h^{\text{sp}}\}_{h=1}^{H}$ and $z^{\text{cs}} = \{z_{s,h}^{\text{cs}}\}_{h=1}^{H}$ are the per-waypoint Spatial and CrossStep memory sequences assembled above, both aligned with the H waypoint rows of $h^{(\ell)} \in \mathbb{R}^{H \times d}$. The gate G adaptively (i) balances the Spatial and CrossStep contributions, (ii) modulates per-waypoint injection, and (iii) applies a global scale α that is initialized to a small value, with a soft floor that prevents the gate from closing completely and removing the learning signal, so as to preserve pre-trained representations:

$$G(\cdot) = \alpha \cdot \big(\gamma^{\text{sp}}(h^{(\ell)}) \odot \psi_{\text{sp}}(z^{\text{sp}}) + \gamma^{\text{cs}}(h^{(\ell)}) \odot \psi_{\text{cs}}(z^{\text{cs}})\big), \tag{9}$$

where all terms live in $\mathbb{R}^{H \times d}$: $\alpha \in \mathbb{R}$ is a scalar global scale constrained to stay above a small floor, $\gamma^{\text{sp}}(h^{(\ell)}), \gamma^{\text{cs}}(h^{(\ell)}) \in \mathbb{R}^{H \times d}$ are waypoint-wise gates produced by a small network conditioned on

Algorithm 1 BEV Action Engram: per sampling step

Require: current trajectory τ_s, context c, prev. velocities $\{\hat{V}_{s-2}, \hat{V}_{s-1}\}$, DiT blocks $\{B_\ell\}_{\ell=1}^{L}$, inject set $\mathcal{I}$

$k_c \leftarrow$ condition key {Eq. (6)}

$\Delta\tau_s \leftarrow \{\Delta A_t\}_{t=1}^{H-1}$; $\{g_h^{(n)}\} \leftarrow \mathrm{NGram}(\mathrm{Quant}_{\mathrm{BEV}}(\Delta\tau_s))$ {displacement n-grams}

$z^{\mathrm{sp}} \leftarrow \mathrm{HashLookup}(g^{(n)} \oplus k_c;\, E^{\mathrm{sp}})$ {Spatial memory, Eq. (4)}

if $s \geq 2$ **then**

  $\mathrm{vid}_{s-j,h} \leftarrow \mathrm{Quant}_{\mathrm{vel}}(\hat{V}_{s-j}[h])$, $j \in \{1,2\}$, $h = 1,\ldots,H$; $z^{\mathrm{cs}} \leftarrow \big[\mathrm{HashLookup}\big((\mathrm{vid}_{s-2,h}, \mathrm{vid}_{s-1,h}, \kappa_s);\, E^{\mathrm{cs}}\big)\big]_{h=1}^{H}$ {CrossStep, delayed}

else

  $z^{\mathrm{cs}} \leftarrow$ null entries of E^{cs} {no history yet}

end if

$h^{(0)} \leftarrow \mathrm{Encode}(\tau_s, c)$

for $\ell = 1$ **to** L **do**

  $h^{(\ell)} \leftarrow B_\ell(h^{(\ell-1)}, c)$

  if $\ell \in \mathcal{I}$ **then**

    $h^{(\ell)} \leftarrow h^{(\ell)} + G(h^{(\ell)}, z^{\mathrm{sp}}, z^{\mathrm{cs}})$ {gated injection, Eq. (8)}

  end if

end for

return velocity / trajectory update from $h^{(L)}$

$h^{(\ell)}$, and $\psi_{\mathrm{sp}} : \mathbb{R}^{m_{\mathrm{sp}} d_e} \to \mathbb{R}^d$ and $\psi_{\mathrm{cs}} : \mathbb{R}^{m_{\mathrm{cs}} d_e} \to \mathbb{R}^d$ map the per-waypoint memory features to width d, one waypoint at a time. $\odot$ denotes elementwise multiplication with broadcasting. Multi-layer injection lets the memory contribute differently to trajectory formation, pattern stabilization, and late-stage refinement. The full procedure is summarized in Algorithm 1.

3.7 Two-Stage Training

Imitation learning. The DiT and the Engram are optimized jointly under the flow-matching objective Lipman et al. (2023). Let $t \in [0,1]$ denote the flow-matching time, with t=0 pure noise and t=1 a clean trajectory; for the linear interpolation $x_t = (1-t)\varepsilon + t\,x^\star$, the target velocity is $v^\star = x^\star - \varepsilon$. Each training step executes a K-step rollout from noise and regresses the predicted velocities against $v^\star$ along the chain (Alg. 2, App. 1.2), so the decoder is supervised on its own rollout states, the regime it faces at inference; all self-comparison entries (the memory-free baseline, Equal-FFN, and the single-branch variants) are trained under this identical protocol. The memory learns the mapping from displacement n-grams and scene context to useful action priors, and gated residual injection keeps the write-back compatible with the pretrained decoder.

Flow matching group relative policy optimization. Recent work shows that reinforcement learning can further improve planning accuracy of large driving models Li et al. (2026b); Zhou et al. (2026b). We therefore adapt Flow-GRPO Liu et al. (2026) to the trajectory DiT. Following Flow-GRPO, the deterministic flow ODE is reformulated as a small-noise SDE so that each sampling step becomes a Gaussian policy. With flow step size $\Delta t = 1/K$, a small noise coefficient a, and drift built from the flow velocity v_θ,

$$\pi_\theta(x_{t+\Delta t} \mid x_t, t, c) = \mathcal{N}\big(\mu_\theta(x_t, t, c),\, \sigma_t^2 I\big), \tag{10}$$

where $\sigma_t = a\sqrt{\frac{t}{1-t}}\,\sqrt{\Delta t}$ and μ_θ is the flow-update drift (recovering the ODE update as $a \to 0$) and σ_t is the diffusion scale with a small numerical floor. For every scene we draw a group of N_g trajectories, score each with an EPDMS reward that includes two-frame extended comfort, and compute group-standardized advantages; this aligns the RL objective with the NAVSIM evaluation metric, which the flow-matching MSE of Imitation Learning (IL) does not directly optimize. The DiT is then updated with a PPO-clip policy loss $\mathcal{L}_{\mathrm{policy}}$ on the sampling chain, an analytic KL $\mathcal{L}_{\mathrm{KL}}$ to a frozen IL reference π_{ref}, and a flow-matching imitation term $\mathcal{L}_{\mathrm{IL}}$, weighted by $\lambda_{\mathrm{KL}}, \lambda_{\mathrm{IL}}$:

$$\mathcal{L} = \mathcal{L}_{\mathrm{policy}} + \lambda_{\mathrm{KL}}\,\mathcal{L}_{\mathrm{KL}} + \lambda_{\mathrm{IL}}\,\mathcal{L}_{\mathrm{IL}}. \tag{11}$$

During this stage we freeze the Engram embedding tables, keeping the projection-and-gate module trainable, and keep the memory pathway active in both π_θ and π_{ref}, so the learned hash-to-embedding mapping serves as a stable action prior while policy optimization touches only the decoder and the projection-and-gate module. SDE kernels, advantage weighting, EPDMS reward realization, and hyperparameters are detailed in App. 1.1.

Table 1: Comparison on the NAVSIMv2 navtest split (12,146 scenarios) under EPDMS, including our flow-matching pipeline variants (gray). Best in **bold**, second-best underlined. EC denotes the two-frame extended comfort. Our entries are computed with the updated official implementation after applying the human-behavior filtering fix, under one identical evaluation configuration. The baseline entries use 5 sampling steps and the Engram entries 3, so the Engram entries match or exceed the baselines with 40% fewer sampling steps.

Method	NC↑	DAC↑	DDC↑	TLC↑	EP↑	TTC↑	LK↑	HC↑	EC↑	EPDMS↑
End-to-End planners										
Transfuser Chitta et al. (2022)	96.9	89.9	97.8	99.7	87.1	95.4	92.7	**98.3**	87.2	84.1
DiffusionDrive Liao et al. (2025)	98.2	96.3	99.4	99.8	87.4	97.4	97.0	**98.3**	87.7	88.3
DiffusionDriveV2 Zou et al. (2025)	97.7	96.6	99.2	99.8	88.9	97.2	96.0	97.8	**91.0**	87.5
World-Model / VLA planners										
DriveVLA-W0 Li et al. (2025b)	98.5	**99.1**	98.0	99.7	86.4	**98.1**	93.2	97.9	58.9	86.1
SG-Drive Li et al. (2026a)	**98.6**	94.3	**99.5**	99.9	86.0	97.9	96.1	**98.3**	85.9	86.2
ReCogDrive-IL Li et al. (2026b)	98.1	94.7	99.4	**100.0**	87.1	97.3	91.1	**98.3**	86.8	86.7
ReCogDrive-RL Li et al. (2026b)	98.1	97.7	98.3	**100.0**	89.8	97.3	90.8	97.7	27.3	82.6
WorldRFT Yang et al. (2026)	97.8	96.5	**99.5**	99.8	88.5	97.0	97.4	98.1	69.1	86.7
DriveJEPA Wang et al. (2026c)	98.4	98.6	99.1	99.8	88.4	97.8	**97.6**	97.9	84.8	87.8
baseline-IL (5-step)	98.2	95.4	99.4	**100.0**	86.9	97.3	97.0	97.9	80.7	86.7
+ BEV Action Engram (3-step)	98.1	95.8	99.4	**100.0**	86.7	97.3	97.1	97.9	79.3	86.9
baseline-RL (5-step)	94.1	96.1	98.9	**100.0**	**93.0**	93.0	95.2	**98.3**	84.7	85.0
+ BEV Action Engram (3-step)	98.4	96.9	**99.5**	**100.0**	87.4	97.8	97.1	**98.3**	84.3	**88.9**

Table 2: Comparison on the Navtest v1. "C" denotes camera; "L" denotes LiDAR. Best in **bold**, second-best underlined.

Method	Venue	Input	NC↑	DAC↑	EP↑	TTC↑	C↑	PDMS↑
End-to-End Planners								
TransFuser Chitta et al. (2022)	CVPR'21	C&L	97.7	92.8	79.2	92.8	**100.0**	84.0
PARA-Drive Weng et al. (2024)	CVPR'24	C	97.9	92.4	79.3	93.0	99.8	84.0
DiffusionDrive Liao et al. (2025)	CVPR'25	C&L	98.2	96.2	82.2	94.7	**100.0**	88.1
MeanFuser Wang et al. (2026b)	CVPR'26	C	98.6	97.0	82.8	95.0	**100.0**	89.0
VADv2 Jiang et al. (2026a)	ICLR'26	C	98.3	**97.4**	82.3	95.7	**100.0**	89.3
World-Model / VLA Planners								
Epona Zhang et al. (2025)	ICCV'25	C	97.9	95.1	80.4	93.8	99.9	86.2
WAM-Diff-IL Xu et al. (2025)	Arxiv'25	C	98.0	95.5	**94.2**	**99.4**	80.7	86.6
WorldRFT Yang et al. (2026)	AAAI'26	C	97.8	96.8	81.7	94.0	**100.0**	87.8
VGGDrive Wang et al. (2026a)	CVPR'26	C	98.6	96.3	82.9	95.6	99.9	88.8
DriveLaW Xia et al. (2026)	CVPR'26	C	**99.0**	97.1	81.3	96.7	**100.0**	89.1
AutoVLA Zhou et al. (2026b)	NeurIPS'25	C	98.4	95.6	81.9	98.0	99.9	89.1
BEV Action Engram (ours)	–	C	98.3	96.1	85.1	95.5	**100.0**	**89.6**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We adopt the NAVSIM Dauner et al. (2024) dataset, built upon nuPlan Caesar et al. (2021), with 1192 and 136 real-world scenarios (about 100k frames) for training and test. We evaluate on NAVSIMv1 using PDMS and its sub-metrics, and on NAVSIMv2 over the 12,146-scenario navtest-v2 split using EPDMS and its sub-metrics. The self-comparison entries in Table 1 (the flow-matching baseline and its Engram variant, each under IL and RL) share the same backbone, sampler, reward, and evaluation configuration. The baselines are the same pipeline with the memory module removed but trained under the same K-step rollout (App. 1.2), so the memory is the only added variable; the Engram entries use 3 sampling steps and the baseline entries 5. We set a frozen InternVL3 Zhu et al. (2025) VLM backbone with a DiT action-decoder design under flow matching for IL, then Flow-GRPO Liu et al. (2026) fine-tuning with Engram. The model is trained on Navtrain on 8 NVIDIA A800 GPUs with a total batch size of 512, for 100 epochs of the IL stage and 6 epochs of the GRPO stage. Engram adds 5.6M params (10% of the DiT) with injection layers $\mathcal{I} = \{2, 6, 10\}$, and the efficiency claim is reported at a 3-step inference budget. Further settings are in App. 2.1.

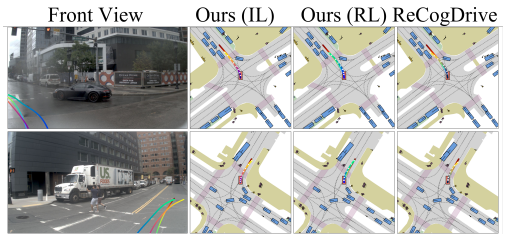


Figure 2: Qualitative comparison at 3-step inference. Red: ground truth; yellow: Engram-IL; cyan: Engram-RL.

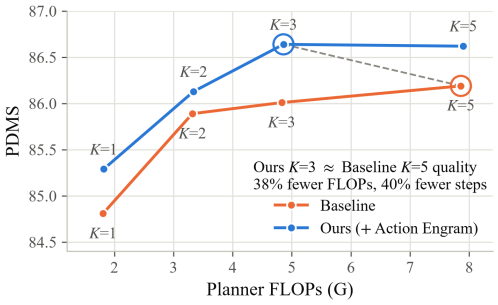


Figure 3: Step–compute–accuracy trade-off on navtest (Flow-IL, NAVSIMv1 protocol). Each marker is a sampling budget K.

4.2 Planning Results Analysis

Tables 1 and 2 report the NAVSIM comparisons; our main entry is Flow-RL+Engram. On NAVSIMv2 (Table 1) it reaches 88.9 EPDMS at 3 steps, the highest among all listed planners. On NAVSIMv1 (Table 2) it reaches 89.6 PDMS, the best among all listed camera and camera+LiDAR planners, with saturated comfort and the second-highest ego progress (85.1 EP); the single higher EP entry (WAM-Diff-IL, 94.2) comes with the lowest comfort (80.7).

4.3 Action Engram under IL and RL

Under the same reward (Table 1), Flow-GRPO without Engram raises ego progress ($86.9 \rightarrow 93.0$ EP) but lowers the collision-related and compliance metrics (NC $98.2 \rightarrow 94.1$, TTC $97.3 \rightarrow 93.0$, LK $97.0 \rightarrow 95.2$), and the aggregate score drops to 85.0, below the IL starting point. Retaining BEV Action Engram as a frozen prior restores the multiplicative safety factors (NC 98.4, DAC 96.9, DDC 99.5, TTC 97.8) and reaches 88.9 EPDMS at 3 sampling steps, a 3.9-point gain over the memory-free run, while ego progress (87.4) stays above both IL models. On the IL side, the 3-step Engram model matches the 5-step baseline (86.9 vs. 86.7), so the IL-side benefit of the memory is reported through the step–quality trade-off in Sec. 4.5 rather than through score differences. The contrast between the two RL runs can also be read as an interaction effect: relative to IL, Flow-GRPO lowers EPDMS by 1.7 without the memory but raises it by 2.0 with the memory, so the learned prior reverses the direction of the reward-optimization effect. The two RL runs are close on the comfort-related terms (HC 98.3 in both, EC 84.7 vs. 84.3), so the overall-score gap is concentrated in the safety factors, consistent with the multiplicative-gate structure of EPDMS.

4.4 Qualitative Trajectory Comparison

Fig. 2 compares 3-step predictions across our Engram-IL, Engram-RL, and ReCogDrive-RL. Under a tight step budget, the ReCogDrive-RL baseline is often jittery or overly conservative, whereas our predictions stay smoother and closer to the ground truth, consistent with the restored safety and compliance factors of Engram-RL in Table 1; more visualizations are in App. 2.6.

4.5 Efficiency Analysis

Fig. 3 plots Flow-IL planning quality on the NAVSIMv1 navtest protocol, under which the step–quality sweep is available. On NAVSIMv2, the 3-step Engram entries in Table 1 match or exceed the 5-step baselines with 40% fewer steps. Engram lifts PDMS at every $K \in \{1, 2, 3, 5\}$ while adding at most 0.04 G planner FLOPs, consistent with $\mathcal{O}(1)$ lookup, so the Engram curve sits above the baseline throughout. The main gain is at matched quality: the 3-step Engram planner (86.64 PDMS) exceeds the 5-step baseline (86.19) with 40% fewer steps, about 38% lower planner FLOPs, and about 14% lower planner wall-clock. The saving comes from spending fewer steps rather than from cheaper ones: at equal K the memory pathway adds planner wall-clock overhead, which the step reduction more than amortizes. Full per-step latency and FLOPs are in App. 2.5.

Table 3: Ablations at the 3-step IL budget (NAVSIMv1 protocol), grouped by the varied factor; all variants are retrained, not masked. Δ is the change in PDMS relative to the full model.

Variant	PDMS	Δ
Flow-IL (baseline)	86.0	-0.6
Equal-FFN (param-matched)	85.9	-0.7
Memory branches (dual $\rightarrow$ single)		
Only Spatial	85.5	-1.1
Only CrossStep	85.1	-1.5
Memory addressing		
w/o condition hash	86.1	-0.5
n-gram order $\{2\}$ only	86.2	-0.4
Injection layers		
Inject $\{2\}$ only	86.3	-0.3
Inject $\{6\}$ only	86.0	-0.6
Full (default)	**86.6**	–

4.6 Why Does the Memory Help

Table 3 ablates the 3-step IL configuration on the NAVSIMv1 protocol (Fig. 3), attributing the gain to *structured conditional* memory rather than added capacity: a parameter-matched Equal-FFN (85.9) does not beat Flow-IL (86.0), so extra parameters alone are insufficient. Spatial-only (85.5) and CrossStep-only (85.1) both underperform the baseline and the full dual memory (86.6), confirming that the two branches are synergistic rather than individually sufficient (App. 2.3); restricting the n-gram order to $\{2\}$ costs 0.4 PDMS, and single-layer injection trails the default $\{2, 6, 10\}$ by 0.3–0.6 PDMS. Removing cross-modal condition hashing drops PDMS to 86.1, close to the unconditioned baseline: without scene conditioning, geometrically identical n-grams collapse onto shared slots and the memory degenerates toward a static template. In App. 2.3, capacity sweeps keep PDMS within a narrow band (86.15–86.57), so the gain comes from *how* the memory is addressed, not from table size.

4.7 Portability to a Masked Diffusion VLA

To test whether the memory is tied to continuous flow matching, we attach the same construction to a masked diffusion VLA with discrete action tokens Xu et al. (2025), freezing the backbone and training only Engram. Attaching the memory yields a PDMS gain of $85.9 \rightarrow 86.3$ without changing the unmasking budget, indicating that the per-step prediction quality transfers across generative action interfaces; more detail is given in App. 2.4.

Method	PDMS
Masked diffusion VLA (LLaDA-V-8B)	85.9
+ **BEV Action Engram**	**86.3**

5 Conclusion

We introduced *BEV Action Engram*, a conditional action memory that serves recurring BEV displacement n-grams via $\mathcal{O}(1)$ lookup and injects them into a generative planner through gated residuals. Under a two-stage IL and Flow-GRPO recipe on NAVSIM, it attains matched planning quality with 40% fewer sampling steps; retained as a frozen prior during Flow-GRPO, it restores the safety factors that reward optimization erodes without it and reaches 88.9 EPDMS on NAVSIMv2 and 89.6 PDMS on NAVSIMv1.

Limitations. The memory saves decoder computation but not the once-per-frame VLM forward, and the validation is confined to NAVSIM; extending to closed-loop and multi-view settings is left to future work.

AI USE STATEMENT

In this work, we used generative AI tools (large language models) for English text editing, LaTeX formatting, and assistance with analyzing experimental results. We have not used generative AI tools for producing the core research ideas, the experimental design, or the scientific conclusions. All AI-assisted work was reviewed and verified by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The full training and evaluation setup is described in the paper: Sec. 3 and App. 1 give the memory construction and the two-stage training procedure, including the Flow-GRPO objective and the train–inference consistent CrossStep interface; App. 2.1 lists the BEV discretization, hashing, and table settings together with the training hyperparameters; Sec. 4 specifies the datasets, evaluation protocols, and sampling budgets on NAVSIMv1 and NAVSIMv2.

REFERENCES

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2026.

Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. *arXiv preprint arXiv:2106.11810*, 2021.

Wei Cao, Marcel Hallgarten, Tianyu Li, Daniel Dauner, Xunjiang Gu, Caojun Wang, Yakov Miron, Marco Aiello, Hongyang Li, Igor Gilitschenski, et al. Pseudo-simulation for autonomous driving. In *Proceedings of the Conference on Robot Learning*, 2025.

Cheng Chang, Jingwei Ge, Jiazhe Guo, Zelin Guo, Binghong Jiang, and Li Li. Driving-rag: Driving scenarios embedding, search, and rag applications: C. chang et al. *Automotive Innovation*, pp. 1–13, 2026.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Han Zhang, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4968–4990, 2026.

Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE transactions on pattern analysis and machine intelligence*, 45(11):12878–12895, 2022.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262, 2004.

Daniel Dauner, Marcel Hallgarten, Tianyu Li, Xinshuo Weng, Zhiyu Huang, Zetong Yang, Hongyang Li, Igor Gilitschenski, Boris Ivanovic, Marco Pavone, Andreas Geiger, and Kashyap Chitta. Navsim: Data-driven non-reactive autonomous vehicle simulation and benchmarking. In *Advances in Neural Information Processing Systems*, volume 37, pp. 28706–28719. Curran Associates, Inc., 2024.

Haoyu Fu, Diankun Zhang, Zongchuang Zhao, Jianfeng Cui, Dingkang Liang, Chong Zhang, Dingyuan Zhang, Hongwei Xie, Bing Wang, and Xiang Bai. Orion: A holistic end-to-end autonomous driving framework by vision-language instructed action generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

Xiaoxuan He, Siming Fu, Yuke Zhao, Wanli Li, Jian Yang, Dacheng Yin, Fengyun Rao, and Bo Zhang. Tempflow-grpo: When timing matters for grpo in flow models. *arXiv preprint arXiv:2508.04324*, 2025.

Tianshuai Hu, Xiaolu Liu, Song Wang, Yiyao Zhu, Ao Liang, Lingdong Kong, Guoyang Zhao, Zeying Gong, Jun Cen, Zhiyu Huang, Xiaoshuai Hao, Linfeng Li, Hang Song, Xiangtai Li, Jun Ma, Shaojie Shen, Jianke Zhu, Dacheng Tao, Ziwei Liu, and Junwei Liang. Vision-language-action models for autonomous driving: Past, present, and future. *arXiv preprint arXiv:2512.16760*, 2025.

Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17853–17862, 2023.

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

Bo Jiang, Shaoyu Chen, Qing Xu, Bencheng Liao, Jiajie Chen, Helong Zhou, Qian Zhang, Wenyu Liu, Chang Huang, and Xinggang Wang. Vad: Vectorized scene representation for efficient autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8340–8350, 2023.

Bo Jiang, Shaoyu Chen, Hao Gao, Bencheng Liao, Qian Zhang, Wenyu Liu, and Xinggang Wang. Vadv2: End-to-end vectorized autonomous driving via probabilistic planning. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Hao Jiang, Zhipeng Zhang, Yu Gao, Zhigang Sun, Yiru Wang, Yuwen Heng, Shuo Wang, Jinhao Chai, Zhuo Chen, Hao Zhao, et al. Flowdrive: Energy flow field for end-to-end autonomous driving. *arXiv preprint arXiv:2509.14303*, 2025a.

Sicong Jiang, Zilin Huang, Kangan Qian, Ziang Luo, Tianze Zhu, Yang Zhong, Yihong Tang, Menglin Kong, Yunlong Wang, Siwen Jiao, et al. A survey on vision-language-action models for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4524–4536, 2025b.

Zhulin Jiang, Zetao Li, Cheng Wang, Ziwen Wang, and Chen Xiong. Anchordrive: Llm scenario rollout with anchor-guided diffusion regeneration for safety-critical scenario generation. *arXiv preprint arXiv:2603.02542*, 2026b.

Jingyu Li, Junjie Wu, Dongnan Hu, Xiangkai Huang, Bin Sun, Zhihui Hao, Xianpeng Lang, Xiatian Zhu, and Li Zhang. Sgdrive: Scene-to-goal hierarchical world cognition for autonomous driving. *arXiv preprint arXiv:2601.05640*, 2026a.

Pengxiang Li, Yinan Zheng, Yue Wang, Huimin Wang, Hang Zhao, Jingjing Liu, Xianyuan Zhan, Kun Zhan, and Xianpeng Lang. Discrete diffusion for reflective vision-language-action models in autonomous driving. *arXiv preprint arXiv:2509.20109*, 2025a.

Yingyan Li, Shuyao Shang, Weisong Liu, Bing Zhan, Haochen Wang, Yuqi Wang, Yuntao Chen, Xiaoman Wang, Yasong An, Chufeng Tang, et al. Drivevla-w0: World models amplify data scaling law in autonomous driving. *arXiv preprint arXiv:2510.12796*, 2025b.

Yongkang Li, Kaixin Xiong, Xiangyu Guo, Fang Li, Sixu Yan, Gangwei Xu, Lijun Zhou, Long Chen, Haiyang Sun, Bing Wang, et al. Recogdrive: A reinforced cognitive framework for end-to-end autonomous driving. In *Proceedings of the International Conference on Learning Representations*, 2026b.

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

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *Proceedings of the International Conference on Learning Representations*, 2023.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-grpo: Training flow matching models via online rl. *Advances in neural information processing systems*, 38:40783–40818, 2026.

Zeting Liu, Zida Yang, Zeyu Zhang, and Hao Tang. Evovla: Self-evolving vision-language-action model. *arXiv preprint arXiv:2511.16166*, 2025.

Yingzi Ma, Yulong Cao, Wenhao Ding, Shuibai Zhang, Yan Wang, Boris Ivanovic, Ming Jiang, Marco Pavone, and Chaowei Xiao. dvlm-ad: Enhance diffusion vision-language-model for driving via controllable reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1050–1061, 2026.

Yukai Ma, Tiantian Wei, Naiting Zhong, Jianbiao Mei, Tao Hu, Licheng Wen, Xuemeng Yang, Botian Shi, and Yong Liu. Leapvad: A leap in autonomous driving via cognitive perception and dual-process thinking. *IEEE Transactions on Neural Networks and Learning Systems*, 2025.

Jianbiao Mei, Yukai Ma, Xuemeng Yang, Licheng Wen, Xinyu Cai, Xin Li, Daocheng Fu, Bo Zhang, Pinlong Cai, Min Dou, et al. Continuously learning, adapting, and improving: A dual-process approach to autonomous driving. *arXiv preprint arXiv:2405.15324*, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. *arXiv preprint arXiv:2508.19236*, 2025.

Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Yang Wang, Zhiyong Zhao, Kun Zhan, Peng Jia, XianPeng Lang, and Hang Zhao. Drivevlm: The convergence of autonomous driving and large vision-language models. In *Conference on Robot Learning*, pp. 4698–4726. PMLR, 2025.

Jie Wang, Guang Li, Zhijian Huang, Chenxu Dang, Hangjun Ye, Yahong Han, and Long Chen. Vggdrive: Empowering vision-language models with cross-view geometric grounding for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10954–10964, 2026a.

Junli Wang, Xueyi Liu, Yinan Zheng, Zebing Xing, Pengfei Li, Guang Li, Kun Ma, Guang Chen, Hangjun Ye, Zhongpu Xia, et al. Meanfuser: Fast one-step multi-modal trajectory generation and adaptive reconstruction via meanflow for end-to-end autonomous driving. *arXiv preprint arXiv:2602.20060*, 2026b.

Linhan Wang, Zichong Yang, Chen Bai, Guoxiang Zhang, Xiaotong Liu, Xiaoyin Zheng, Xiao-Xiao Long, Chang-Tien Lu, and Cheng Lu. Drive-jepa: Video jepa meets multimodal trajectory distillation for end-to-end driving. *arXiv preprint arXiv:2601.22032*, 2026c.

Xinshuo Weng, Boris Ivanovic, Yan Wang, Yue Wang, and Marco Pavone. Para-drive: Parallelized architecture for real-time autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15449–15458, 2024.

Tianze Xia, Yongkang Li, Lijun Zhou, Jingfeng Yao, Kaixin Xiong, Haiyang Sun, Bing Wang, Kun Ma, Guang Chen, Hangjun Ye, et al. Drivelaw: Unifying planning and video generation in a latent driving world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 39701–39712, 2026.

Mingwang Xu, Jiahao Cui, Feipeng Cai, Hanlin Shang, Zhihao Zhu, Shan Luan, Yifang Xu, Neng Zhang, Yaoyi Li, Jia Cai, et al. Wam-diff: A masked diffusion vla framework with moe and online reinforcement learning for autonomous driving. *arXiv preprint arXiv:2512.11872*, 2025.

Yifang Xu, Jiahao Cui, Zhihao Zhu, Hanlin Shang, Shan Luan, Mingwang Xu, Feipeng Cai, Neng Zhang, Yaoyi Li, Jia Cai, et al. Wam-flow: Parallel coarse-to-fine motion planning via discrete flow matching for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24918–24928, 2026.

Pengxuan Yang, Ben Lu, Zhongpu Xia, Chao Han, Yinfeng Gao, Teng Zhang, Kun Zhan, XianPeng Lang, Yupeng Zheng, and Qichao Zhang. Worldrft: Latent world model planning with reinforcement fine-tuning for autonomous driving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 11649–11657, 2026.

Yixiang Yang, Chuanrong Han, Runhao Mao, et al. Survey of general end-to-end autonomous driving: A unified perspective. *TechRxiv*, December 2025. doi: 10.36227/techrxiv.176523315.56439138/v1.

Liuhan Yin, Runkun Ju, Guodong Guo, and Erkang Cheng. Diffrefiner: Coarse to fine trajectory planning via diffusion refinement with semantic interaction for end to end autonomous driving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Jianhao Yuan, Shuyang Sun, Daniel Omeiza, Bo Zhao, Paul Newman, Lars Kunze, and Matthew Gadd. Rag-driver: Generalisable driving explanations with retrieval-augmented in-context learning in multi-modal large language model. *arXiv preprint arXiv:2402.10828*, 2024.

Kaiwen Zhang, Zhenyu Tang, Xiaotao Hu, Xingang Pan, Xiaoyang Guo, Yuan Liu, Jingwei Huang, Li Yuan, Qian Zhang, Xiao-Xiao Long, et al. Epona: Autoregressive diffusion world model for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 27220–27230, 2025.

Xingcheng Zhou, Xuyuan Han, Feng Yang, Yunpu Ma, Volker Tresp, and Alois Knoll. Opendrivevla: Towards end-to-end autonomous driving with large vision language action model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 13782–13790, 2026a.

Zewei Zhou, Tianhui Cai, Seth 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. *Advances in Neural Information Processing Systems*, 38: 27920–27956, 2026b.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

Jialv Zou, Shaoyu Chen, Bencheng Liao, Zhiyu Zheng, Yuehao Song, Lefei Zhang, Qian Zhang, Wenyu Liu, and Xinggang Wang. Diffusiondrivev2: Reinforcement learning-constrained truncated diffusion modeling in end-to-end autonomous driving. *arXiv preprint arXiv:2512.07745*, 2025.

APPENDIX

1 MORE DETAIL OF METHOD

1.1 FLOW-GRPO FOR TRAJECTORY DECODER

This appendix details the second-stage Flow-GRPO Liu et al. (2026) fine-tuning used after imitation learning. The goal is to keep IL, RL, and evaluation in the same flow-matching family, while aligning the scalar reward with NAVSIM EPDMS Cao et al. (2025).

Stochastic flow policy. Let $x_t \in \mathbb{R}^{H\times D}, H = 8, D = 3$ denote the (normalized) trajectory at flow time $t \in [0, 1]$, with t=0 pure noise and t=1 a clean trajectory. The DiT predicts a velocity field $v_\theta(x_t, t, c)$. Evaluation uses the deterministic ODE:

$$x_{t+\Delta t} = x_t + \Delta t \cdot v_\theta(x_t, t, c), \qquad \Delta t = 1/K. \tag{12}$$

Following Flow-GRPO, online RL requires a stochastic policy with tractable log-probabilities. With noise level a and step size Δt, we construct a Gaussian transition around the flow update:

$$\begin{aligned}\mu_\theta(x_t, t, c) = x_t\left(1 + \frac{a^2}{2(1-t)}\Delta t\right) \\ + v_\theta(x_t, t, c)\left(1 + \frac{a^2}{2}\right)\Delta t,\end{aligned} \tag{13}$$

and sample:

$$\begin{aligned}x_{t+\Delta t} \sim \pi_\theta(\cdot \mid x_t, t, c) = \mathcal{N}\big(\mu_\theta(x_t, t, c),\, \sigma_t^2 I\big), \\ \sigma_t = a\sqrt{\frac{t}{1-t}}\,\sqrt{\Delta t},\end{aligned} \tag{14}$$

with a numerical floor on σ_t, applied identically in rollout sampling and in the log-probability computation; with $a = 0.005$ the floor is active at every step, so the exploration noise is effectively a small constant. A separate, slightly larger floor is used as the KL denominator (see below). As $a \to 0$, μ_θ recovers the ODE update, so exploration remains close to the IL manifold. The per-step log-probability is the corresponding isotropic Gaussian density averaged over the $H \times D$ action dimensions, following the step-wise, dimension-normalized convention of the Flow-GRPO implementation; log-probabilities are kept per sampling step, and the importance ratios below act on these per-step values. During rollouts, Spatial/CrossStep Engram injection uses the same delayed interface as IL and inference:

$$\begin{aligned}\log \pi_\theta(x_{t+\Delta t} \mid x_t, t, c) = \frac{1}{HD}\sum_{h,d}\Bigg[-\frac{(x'_{hd} - \mu_{\theta,hd})^2}{2\sigma_t^2} \\ - \log \sigma_t - \frac{1}{2}\log(2\pi)\Bigg],\end{aligned} \tag{15}$$

where $x' \equiv x_{t+\Delta t}$ denotes the sampled next state and $x'_{hd}, \mu_{\theta,hd}$ are its per-waypoint per-channel components.

Group-relative advantage and policy objective. During rollouts, transition log-probabilities $\log \pi_{\text{old}}$ are recorded without gradient; the chain is then replayed to recompute $\log \pi_\theta$ with gradient for the policy update. The replay reuses the stored chain states and the detached velocity history as-is (they are not regenerated), so the CrossStep keys are fixed during the update and the DiT forward is the only gradient-carrying path; the reference policy receives the same chain and velocity history. Sampling steps are indexed by $s \in \{0, \ldots, K-1\}$ as in the main text.

After sampling a group of trajectories $\{x^{(i,g)}\}_{g=1}^{N_g}$ for each scene i, we assign a scalar reward $r^{(i,g)}$ and compute the group-standardized advantage:

$$A^{(i,g)} = \text{clip}\left(\frac{r^{(i,g)} - \text{mean}_g(r^{(i,\cdot)})}{\text{std}_g(r^{(i,\cdot)}) + \epsilon},\ -A_{\max},\ A_{\max}\right). \tag{16}$$

The reward is defined per trajectory, so a single scalar $r^{(i,g)}$ is shared by all K sampling steps of that rollout and enters each step only after the discounted broadcast of Eq. (16). Advantages are broadcast to sampling steps with a discount γ,

$$\tilde{A}_s^{(i,g)} = A^{(i,g)}\,\gamma^{K-s-1}, \tag{17}$$

so that later steps nearer the clean trajectory receive larger weight when $\gamma < 1$. With importance ratio:

$$\begin{aligned}\rho_s^{(i,g)} = \exp\Big(\log \pi_\theta(x_{s+1} \mid x_s, t_s, c) \\ - \log \pi_{\text{old}}(x_{s+1} \mid x_s, t_s, c)\Big),\end{aligned} \tag{18}$$

the policy loss is the PPO-clip surrogate:

$$\begin{aligned}\mathcal{L}_{\text{policy}} = \mathbb{E}_{i,g,s}\Big[\max\Big(-\tilde{A}_s\,\rho_s, \\ -\tilde{A}_s \cdot \text{clip}(\rho_s,\, 1-\epsilon_{\text{clip}},\, 1+\epsilon_{\text{clip}})\Big)\Big].\end{aligned} \tag{19}$$

To stabilize updates, we add an analytic Gaussian KL between the transition means of π_θ and a frozen IL reference π_{ref} that shares the same SDE variance structure,

$$\mathcal{L}_{\text{KL}} = \mathbb{E}\left[\frac{\|\mu_\theta(x_t,t,c) - \mu_{\text{ref}}(x_t,t,c)\|_2^2}{2\,\sigma_{\text{ref}}^2}\right], \sigma_{\text{ref}} \geq \sigma_{\text{KL}}^{\min}, \tag{20}$$

where the denominator is clamped to a small floor $\sigma_{\text{KL}}^{\min}$ for numerical stability, and the first sampling step is excluded from the KL sum since σ_t is ill-conditioned near $t = 0$. To prevent the reinforcement learning trajectories from deviating too much and to improve training stability, we introduced an imitation learning loss function to regularize the output:

$$\mathcal{L}_{\text{IL}} = \mathbb{E}_{t,\varepsilon}\left\|v_\theta(x_t,t,c) - v^\star\right\|_2^2. \tag{21}$$

The total objective is

$$\mathcal{L} = \mathcal{L}_{\text{policy}} + \lambda_{\text{KL}}\,\mathcal{L}_{\text{KL}} + \lambda_{\text{IL}}\,\mathcal{L}_{\text{IL}}. \tag{22}$$

The Engram embedding tables have $\text{requires_grad} = \text{False}$; the DiT and the projection-and-gate module are the only trainable parameters.

EPDMS as the reward. We use EPDMS as the RL reward so that policy optimization targets the same evaluation metric reported at test time. Following the official multiplicative-safety $\times$ weighted-submetric structure, the score aggregates no-collision, drivable-area, traffic-light, and directionality multiplicative gates with weighted terms (progress, TTC, lane keeping, history comfort, and the two-frame extended comfort EC):

$$r_{\text{single}} = M \cdot \frac{\sum_{j\neq\text{EC}} w_j\, m_j}{\sum_{j\neq\text{EC}} w_j}, M = \prod_j m_j^{\text{multi}}, \tag{23}$$

where the single-frame aggregator masks EC since EC requires cross-frame information. For the full reward, the worker caches per-log ego states and fuses the two-frame EC term when adjacent frames fall within a time window $\Delta T \in (0,\, 0.55]\,\text{s}$; otherwise EC is dropped and its weight is set to zero:

$$\begin{aligned} r_{\text{full}} &= M \cdot \frac{\sum_j w_j\, \tilde{m}_j}{\sum_j w_j}, \\ \tilde{m}_j &= \begin{cases} m_{\text{EC}} & j = \text{EC and EC is available,} \\ 0\ (w_{\text{EC}} \leftarrow 0) & j = \text{EC and EC is unavailable,} \\ m_j & \text{otherwise.} \end{cases} \end{aligned} \tag{24}$$

The default submetric weights follow the official scorer: progress 5.0, TTC 5.0, lane keeping 2.0, history comfort 2.0, and extended comfort 2.0. The scalar reward assigned to each rollout trajectory in Eq. (16) is this EPDMS score, $r^{(i,g)} = r_{\text{full}}(x^{(i,g)})$, computed per trajectory; the single-frame aggregator r_{single} is used as a fallback only when no adjacent frame is available to form the two-frame EC term. This trajectory-level reward is the sole signal entering $\mathcal{L}_{\text{policy}}$ through the group-standardized advantage of Eq. (16), while $\mathcal{L}_{\text{KL}}$ and $\mathcal{L}_{\text{IL}}$ act as reward-independent regularizers. This stateful realization is intended to be isomorphic to the NAVSIMv2 evaluation protocol. Reward scoring runs asynchronously on CPU and overlaps with GPU log-probability recomputation, without changing the mathematical objective.

Optimization protocol and hyperparameters. Stage 1 (IL) jointly trains the DiT and Engram under flow matching. Stage 2 (Flow-GRPO) freezes the Engram embedding tables, together with the IL reference π_{ref}, and fine-tunes the DiT and the projection-and-gate module with Eq. (22). Unless otherwise noted, Flow-GRPO uses $K{=}5$ sampling steps, group size $N_g{=}8$, a very small SDE noise $a{=}0.005$ (near-ODE exploration), PPO clip $\epsilon_{\text{clip}}{=}0.05$, $\lambda_{\text{KL}}{=}10^{-5}$, $\lambda_{\text{IL}}{=}0.32$, advantage clip $A_{\max}{=}5$, and denoising discount $\gamma{=}0.8$. The log-probability standard deviation is floored at $\sigma_{\min}{=}0.01$ and the KL denominator at $\sigma_{\text{KL}}^{\min}{=}0.02$, with the first sampling step skipped in the KL sum. The imitation term uses a one-step rollout on ground-truth trajectories. The reference policy is the IL checkpoint of the corresponding branch (baseline-IL for the memory-free run, Engram-IL for the memory-augmented run), with all of its parameters frozen. Optimization uses AdamW with a learning rate of 1×10^{-5} and a per-GPU batch size of 8. Evaluation reports PDMS / EPDMS under deterministic flow-ODE sampling with Engram enabled; the main paper uses the 3-step inference budget for the efficiency claim, while RL training uses $K{=}5$. Adaptive step schedules are disabled during GRPO so that the sampling chain length remains fixed for log-probability accounting.

Role of the Engram during RL. We freeze the Engram embedding tables during RL (the projection-and-gate module stays trainable) but keep the memory pathway active in both π_θ and π_{ref}. This design is deliberate: the SDE noise injected for exploration would otherwise corrupt the learned hash-to-embedding mapping, so the IL-stage mapping serves as a stable action prior while policy optimization touches only the decoder and the projection-and-gate module. Keeping the reference policy Engram-active ensures the analytic KL is measured in the same prior space. Under the same EPDMS reward, the memory-free run raises ego progress while the multiplicative safety factors drop and the aggregate score falls below its IL starting point, whereas the memory-augmented run keeps the safety factors high and improves the overall score (Table 1 of the main text); the frozen tables thus anchor policy optimization to action patterns already validated during IL.

1.2 CrossStep Train–Inference Consistent Interface

A central question is how the two prior velocity fields used by the CrossStep memory are obtained *during training*, since standard random-time flow-matching training does not naturally produce them. This appendix gives a reproducible answer and explains why the interface is identical at train and inference time. All symbols used here follow the main text.

Velocity identifier. The CrossStep key is formed per waypoint: each row $\hat{V}_s[h] \in \mathbb{R}^3$ of a predicted velocity field yields one identifier. We quantize each component with a quantum $\Delta = 0.25$ over the symmetric range $[-v_{\max}, v_{\max}]$ with $v_{\max} = 4$, giving $B = 2\,v_{\max}/\Delta + 1 = 33$ bins; the range is chosen in normalized units to cover the target velocity $v^\star = x^\star - \varepsilon$ with wide margin. For waypoint $h \in \{1, \dots, H\}$ and component $d \in \{1, 2, 3\}$ (corresponding to x, y, θ), the raw index is

$$r_{h,d} = \text{clip}\left(\text{round}\left(\frac{\hat{V}_s[h,d]}{\Delta}\right),\ -\frac{v_{\max}}{\Delta},\ \frac{v_{\max}}{\Delta}\right), \tag{25}$$

shifted to a non-negative digit $b_{h,d} = r_{h,d} + v_{\max}/\Delta \in \{0, \dots, B-1\}$, and the three digits of waypoint h are packed by mixed radix into

$$\text{vid}_{s,h} = \text{Quant}_{\text{vel}}\big(\hat{V}_s[h]\big) = b_{h,1}B^2 + b_{h,2}B + b_{h,3} \in \big\{0, \dots, B^3-1\big\}, \tag{26}$$

used in Algorithms 1, 2 and 3.

Cross-step hashing. For hash head i and waypoint h, the identifiers of the two most recent steps are aggregated with the discretized flow time by a multilinear (multiply–XOR) hash. Let $\kappa_s = \lfloor t_s N_t \rfloor$ denote the flow-matching time discretized into $N_t = 1000$ buckets, where t_s is the current flow time (sampled continuously during IL training and equal to s/K at inference). With fixed random odd multipliers $\omega_{i,1}, \omega_{i,2}, \omega^{\text{t}}$,

$$\text{slot}^{\text{cs}}_{s,h,i} = \big(\text{vid}_{s-2,h}\cdot\omega_{i,1} \ \oplus\ \text{vid}_{s-1,h}\cdot\omega_{i,2} \ \oplus\ \kappa_s\cdot\omega^{\text{t}}\big) \bmod N_{\text{slot}}, \tag{27}$$

where $\oplus$ denotes bitwise XOR, so the step order is carried by the per-position multipliers rather than by digit weights, and the time term lets the same velocity pair address different slots at different denoising stages. All intermediates are bounded by $B^3 \cdot \max\omega < 2^{42}$ and XOR does not increase bit width, so Eq. (27) is exact in fixed-width integers. Collisions are not avoided but diluted: each (waypoint, head) pair is reduced to one of N_{slot} slots, and the m_{cs} heads use independent multiplier tables, so two keys colliding under one head are unlikely to collide under all, as in the Spatial branch (Sec. 3.3). When fewer than two prior velocity fields are available ($s < 2$), the lookup of every head falls back to a dedicated null entry of E^{cs}; a legitimate hash that reduces to the null slot under all heads has negligible probability ($N_{\text{slot}}^{-m_{\text{cs}}}$).

The mismatch problem. A naive CrossStep design would hash the current noisy action: during training the noisy state is a known interpolation of ground truth and noise, whereas at inference it is the model's own prediction. The two distributions differ, so a CrossStep memory trained on interpolated inputs would not transfer stably to inference. Our *delayed injection* avoids this by using only already-predicted velocity fields from prior sampling steps at *both* training and inference, and falling back to the null entry when fewer than two prior velocities are available ($s < 2$).

Algorithm 2 CrossStep delayed injection: training rollout (one scene)

Require: context c, DiT blocks $\{B_\ell\}$, inject set $\mathcal{I}$, rollout steps K
Sample noise ε; init $x_0 = \varepsilon$; vel_history $\leftarrow [\,]$
for $s = 0$ **to** $K - 1$ **do**
  $k_c \leftarrow$ condition key
  $\{g_t^{(n)}\} \leftarrow \mathrm{NGram}(\mathrm{Quant}_{\mathrm{BEV}}(\Delta\tau_s))$ {displacement n-grams}
  $z^{\mathrm{sp}} \leftarrow$ HashLookup$(g^{(n)} \oplus k_c;\, E^{\mathrm{sp}})$ {Spatial, from x_s}
  if $|\mathrm{vel_history}| \geq 2$ **then**
    $\mathrm{vid}_{s-j,h} \leftarrow \mathrm{Quant}_{\mathrm{vel}}(\hat{V}_{s-j}[h])$, $j \in \{1,2\}$, $h = 1, \ldots, H$
    $z^{\mathrm{cs}} \leftarrow \big[\mathrm{HashLookup}\big((\mathrm{vid}_{s-2,h}, \mathrm{vid}_{s-1,h}, \kappa_s);\, E^{\mathrm{cs}}\big)\big]_{h=1}^{H}$
  else
    $z^{\mathrm{cs}} \leftarrow$ null entries {delayed: no history yet}
  end if
  $\hat{V}_s \leftarrow \mathrm{DiT}(x_s, c, z^{\mathrm{sp}}, z^{\mathrm{cs}})$ with gated injection at $\ell \in \mathcal{I}$
  $x_{s+1} \leftarrow$ flow update using $\hat{V}_s$
  $\mathrm{vel_history.append}(\hat{V}_s.\mathrm{detach}())$ {no grad through keys}
end for
Compute flow-matching loss against $v^\star$ on the chain (with gradient through $\hat{V}_s$ but not through the buffer)

Training-time K-step rollout. As shown in Alg. 2, during IL training we execute a K-step rollout along the flow-matching chain. At each step s, the per-waypoint CrossStep keys are built from the two most recent entries of the velocity history buffer, $\mathrm{vel_history}[-2:] = \{\hat{V}_{s-2}, \hat{V}_{s-1}\}$, exactly as at inference; the model then predicts a velocity $\hat{V}_s$, which we detach from the graph and append to the buffer. Because the keys are always sourced from the model's own (detached) predictions and never from the ground-truth interpolation, they are constructed by the same rule at training and inference. Gradients flow through the DiT and the Engram injection path but not through the velocity history used as keys.

Inference procedure. Inference follows the identical mechanism: at each step the model predicts a velocity, the per-waypoint CrossStep keys are built from the two most recent buffer entries, and the branch falls back to the null entry for $s < 2$. The only difference from training is that no loss is computed, no gradient is propagated, and the appended velocity is not detached. Algorithm 3 makes this explicit.

Algorithm 3 CrossStep delayed injection: inference (one scene)

Require: context c, rollout steps K
Init $x_0 = \varepsilon$; vel_history $\leftarrow [\,]$
for $s = 0$ **to** $K - 1$ **do**
  $\{g_t^{(n)}\} \leftarrow \mathrm{NGram}(\mathrm{Quant}_{\mathrm{BEV}}(\Delta\tau_s))$
  $z^{\mathrm{sp}} \leftarrow$ HashLookup$(g^{(n)} \oplus k_c;\, E^{\mathrm{sp}})$
  if $|\mathrm{vel_history}| \geq 2$ **then**
    $\mathrm{vid}_{s-j,h} \leftarrow \mathrm{Quant}_{\mathrm{vel}}(\hat{V}_{s-j}[h])$, $j \in \{1,2\}$, $h = 1, \ldots, H$
    $z^{\mathrm{cs}} \leftarrow \big[\mathrm{HashLookup}\big((\mathrm{vid}_{s-2,h}, \mathrm{vid}_{s-1,h}, \kappa_s);\, E^{\mathrm{cs}}\big)\big]_{h=1}^{H}$
  else
    $z^{\mathrm{cs}} \leftarrow$ null entries
  end if
  $\hat{V}_s \leftarrow \mathrm{DiT}(x_s, c, z^{\mathrm{sp}}, z^{\mathrm{cs}})$
  $x_{s+1} \leftarrow$ flow update using $\hat{V}_s$
  $\mathrm{vel_history.append}(\hat{V}_s)$
end for
return x_K

Train–inference alignment. Table 4 enumerates the CrossStep interface on both sides; every component is identical, which is the precise content of the "train–inference consistent" claim in the main text.

Table 4: CrossStep interface is identical at training and inference.

Component	Training	Inference
Velocity history source	$\hat{V}_s$ (model pred., detached)	$\hat{V}_s$ (model pred.)
Key composition	Per-waypoint multiply–XOR of the two most recent velocity identifiers, without the cross-modal condition key (unlike Spatial)	same
Time term	discretized flow time κ_s times a multiplier, XOR'd into the key	same
Null-entry condition	$s < 2$	$s < 2$
Quantization	$\mathrm{Quant}_{\mathrm{vel}}$	same

History truncation and the $K{=}1$ gain. The CrossStep keys cover the two most recent predictions, so only a two-step effective history is needed and a K-step rollout provides at most that much regardless of K. Table 5 reports the number of CrossStep-active steps for each inference budget K used in the paper. At $K{=}1$ the CrossStep branch reads only the null entry, so the $K{=}1$ gain over the baseline ($84.81 \rightarrow 85.29$ PDMS) is attributable solely to the Spatial branch. CrossStep only begins to contribute at $K \geq 3$, consistent with its role as an auxiliary convergence signal rather than a primary geometric prior, and with the ablation in the main text.

Table 5: Number of sampling steps with an active CrossStep lookup (velocity history available), for each inference budget K.

K	1	2	3	5
CrossStep-active steps	0	0	1	3

2 More Detail of Experiments

2.1 Settings and Hyperparameters

BEV displacement discretization. We represent actions by relative displacements $\Delta A_t = (\Delta x_t, \Delta y_t, \Delta\theta_t)$, which are position-independent and thus reusable across scenes. Each displacement is quantized on a 61×61 BEV grid with 37 heading bins to a discrete identifier $\mathrm{id}_t \in \{1, \ldots, V_d\}$ with $V_d = 61 \cdot 61 \cdot 37 = 137{,}197$, covering the per-step displacement range of urban driving.

Cross-modal condition hashing. To disambiguate geometrically identical motions across scenes, each conditioning modality is hashed *independently* and fused by multiply–XOR rather than concatenated and hashed jointly, so that related scenes can share memory while dissimilar scenes are separated. Continuous features (VLM hidden states, history) use locality-sensitive hashing with non-trainable projection matrices; discrete signals (navigation command, ego state) are directly quantized. Bit widths and settings are listed in Table 7.

Multi-head hashing and table capacity. Spatial memory uses $m_{\mathrm{sp}} = 8$ hash heads of width $d_e = 32$ (four heads per n-gram order), and CrossStep uses $m_{\mathrm{cs}} = 4$ heads of width $d_e = 64$; each head addresses a table of $N_{\mathrm{slot}} = 10{,}007$ slots plus a null row, so both branches produce 256-dimensional concatenated per-waypoint features. The Spatial slot computation follows the same multiply–XOR–mod pattern as the CrossStep hash of Eq. (27): per-position multipliers over the window's displacement identifiers, the condition key k_c folded in with a per-head multiplier, and a single modular reduction. Multi-head hashing reduces collision ambiguity: two n-grams colliding under one hash are unlikely to collide under all. The Engram adds about 5.6M parameters, roughly 10% of the DiT, split across the two table sets and a small projection-and-gate module:

Gate residual injection. The global scale α is a single learnable scalar shared across the injection layers $\mathcal{I} = \{2, 6, 10\}$, initialized small so that the effective gate scale is about 0.10, with a floor that keeps the gradient path to the embedding tables open. The per-waypoint gates $\gamma^{\mathrm{sp}}, \gamma^{\mathrm{cs}}$ and the

Table 6: Engram parameter breakdown.

Module	Params (approx.)	Role
Spatial embedding ($8 \times 10{,}007 \times 32$)	$\sim$2.56M	BEV action primitive bank
CrossStep embedding ($4 \times 10{,}007 \times 64$)	$\sim$2.56M	denoising velocity bank
Projection + Gate	$\sim$0.5M	conditioning fusion
Engram total	$\sim$5.6M	$\sim$10% of DiT

Table 7: Memory addressing configuration. All hash multipliers are fixed-seed random odd integers in $[10^4, 10^7)$, drawn as $\texttt{randint}(10^4, 10^7) \,|\, 1$ with NumPy `RandomState` seeds 242 (Spatial, 20 values), 342 (CrossStep, 16), and 142 (condition key, 16).

Spatial branch	
Displacement quantization	61×61 grid, 0.25 m/bin (± 7.5 m), 37 heading levels
n-gram orders	$\{2, 3\}$; $m_{\text{sp}} = 8$ heads (four per order)
Table per head	$N_{\text{slot}} = 10{,}007$ slots + null row, $d_e = 32$
Window alignment	per waypoint; fewer than n leading displacements $\rightarrow$ null entry
CrossStep branch	
Velocity quantization	$B = 33$ bins/component, $\Delta = 0.25$, ± 4 (normalized units)
Key span	two most recent steps, per waypoint; time mixing with $N_t = 1000$ buckets
Table per head	$m_{\text{cs}} = 4$; 10,007 slots + null row, $d_e = 64$
Condition key k_c	
Navigation command	4 classes, directly quantized
VLM hidden states	8-bit LSH, mean-pooled last hidden state, fixed random projection
Ego status	8 components (navigation one-hot, velocity, acceleration), 33 bins/component
History	4-bit LSH over the four past ego poses, fixed random projection

projections $\psi_{\text{sp}}, \psi_{\text{cs}}$ are small MLPs conditioned on the block hidden state. This design preserves the pretrained decoder while still allowing the memory to be learned end-to-end.

Training protocol. Stage 1 jointly trains the DiT and the Engram under flow matching with the InternVL3-2B VLM backbone frozen and cached hidden states reused. IL runs for 100 epochs on 8 NVIDIA A800 GPUs with a per-GPU batch of 64 (effective batch 512) and a learning rate of 1×10^{-4}; each step executes a $K = 3$-step flow rollout so that the CrossStep delayed-injection interface is learned under the same chain used at inference; the memory-free baselines are trained under the identical rollout with the Engram module removed. At inference the sampling budget is varied independently of this training rollout (Fig. 3 of the main text). Stage 2 (Flow-GRPO) fine-tunes the DiT and the projection-and-gate module with the objective of Eq. (22), freezing the Engram embedding tables and the IL reference π_{ref}; its SDE, PPO, KL, and IL hyperparameters are detailed in App. 1.1, and it runs for 6 epochs on the same hardware.

2.2 Other Baselines

For completeness we characterize the baselines in Tables 1 and 2. They fall into two families: end-to-end planners that map sensors directly to trajectories, and world-model / VLA planners that condition a generative decoder on a vision-language or latent-world backbone. Our entry uses the same camera-only input and InternVL3-2B backbone as the VLA group; the controlled IL/RL comparison in Table 1 holds the backbone, sampler, reward, and evaluation fixed, isolating the Engram contribution from backbone and supervision differences.

End-to-end planners. TransFuser Chitta et al. (2022) fuses camera and LiDAR with multi-resolution transformers, then predicts waypoints autoregressively. PARA-Drive Weng et al. (2024) co-trains perception, prediction, and planning in parallel without serial dependencies, improving runtime while staying interpretable. DiffusionDrive Liao et al. (2025) denoises from anchored Gaussians to multi-mode trajectories with a cascade decoder, and DiffusionDriveV2 Zou et al. (2025) adds intra-/inter-anchor GRPO to suppress low-quality modes while preserving multimodality. MeanFuser Wang et al. (2026b) replaces discrete anchors with Gaussian-mixture noise and a MeanFlow identity for one-step generation, refined by an adaptive reconstruction module. VADv2 Jiang

et al. (2026a) discretizes the planning space into a trajectory vocabulary and samples actions from a predicted token distribution.

World-model / VLA planners. Epona Zhang et al. (2025) is an autoregressive diffusion world model that jointly predicts future video and trajectories through decoupled spatiotemporal factorization. WAM-Diff Xu et al. (2025) is a masked-diffusion VLA that refines discrete trajectory tokens with a sparse MoE backbone and online GSPO. WorldRFT Yang et al. (2026) is a latent world model with hierarchical planning and local-aware refinement, fine-tuned by GRPO. DriveVLA-W0 Li et al. (2025b) counters VLA supervision sparsity by predicting future images as dense world-model self-supervision. SG-Drive Li et al. (2026a) organizes a pretrained VLM around a scene–agent–goal hierarchy before planning. ReCogDrive Li et al. (2026b) couples a cognitive VLM with a diffusion planner reinforced by DiffGRPO (IL/RL variants in Table 1). DriveJEPA Wang et al. (2026c) pretrains a video JEPA encoder and plans by distilling human and simulator proposals. VGGDrive Wang et al. (2026a) injects cross-view 3D geometric features from a frozen 3D foundation model into a VLM. DriveLaW Xia et al. (2026) feeds shared video latents from a world model into a diffusion planner. AutoVLA Zhou et al. (2026b) unifies reasoning and discrete trajectory actions in one autoregressive VLA with fast/slow modes, then applies GRPO to cut unnecessary reasoning.

2.3 Results Analysis

2.3.1 Gate Analysis

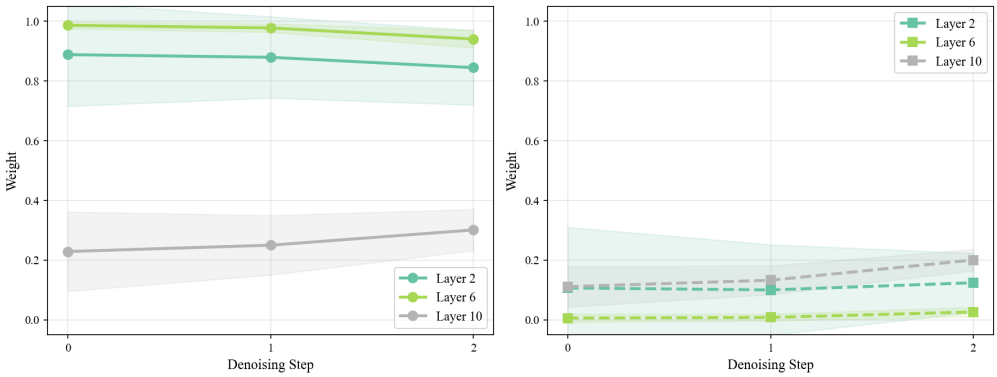


Figure 4: Layer-wise and step-wise gate weights of the dual-branch Action Engram. Spatial dominates at early and mid layers (L2 $\approx$0.88, L6 $\approx$0.97) and decreases at the late layer (L10 $\approx$0.30) where CrossStep rises; the effective gate scale is about 0.10, and CrossStep increases slightly after the first two steps. The pattern matches the multi-depth injection design $\mathcal{I} = \{2, 6, 10\}$ and the single-layer ablation in the main text.

Layer-wise gate statistics show (Fig. 4) Spatial dominance at early and mid layers (L2 $\approx$0.88, L6 $\approx$0.97) and a relative rise of CrossStep at the late layer (L10 Spatial $\approx$0.30), with an effective gate scale of about 0.10. Across denoising steps the Spatial weight is stable while CrossStep rises slightly, consistent with its role as an auxiliary convergence signal that becomes relevant only after the first two steps. The layer-wise pattern, with Spatial strong where trajectory geometry is formed and CrossStep relatively stronger at late-stage refinement, matches the multi-depth injection design ($\mathcal{I} = \{2, 6, 10\}$) and the single-layer ablation ($\{6\}$ only, 86.0 vs. full 86.6) in the ablation section of the main text.

2.3.2 Scaling Analysis

We sweep Engram capacity along two orthogonal axes, vocabulary size (1,009/5,003/10,007/20,011) and embedding dimension (64/128/256/384/512), holding the aligned 8-GPU IL recipe fixed (App. 2.1). PDMS varies within a narrow band (86.15–86.57) and peaks near the default configuration ($\sim$6M parameters), with no monotonic gain from further scaling. At matched parameter budgets, the vocabulary and embedding-dimension axes yield comparable performance, suggesting that structured memory capacity rather than a specific axis

choice governs the ceiling. Together with the parameter-matched Equal-FFN control in Table 3, this supports that the gain comes from *how* the memory is addressed, not from table size or parameter count.

2.3.3 SINGLE-BRANCH INTERACTION

Both single-branch variants are retrained and fall below the Flow-IL baseline (86.0): Spatial-only reaches 85.5 and CrossStep-only 85.1, while the full dual memory reaches 86.6. The quantity of interest is the interaction: the two single-branch effects sum to -1.4 PDMS, yet the joint effect is $+0.6$, an interaction of about $+2.0$ PDMS. Neither branch is individually sufficient; the gain comes from their composition.

We read this as follows. The two branches inject priors on different axes: the Spatial branch provides scene-conditioned geometry but is time-invariant, whereas the CrossStep branch is time-aware (Eq. (27)) but carries no scene context. Injected alone, each is a partial prior whose bias is not counterbalanced: Spatial-only pulls toward average motion patterns at stages that call for data-specific refinement, and CrossStep-only injects geometrically ungrounded corrections. With both branches present, the gate composes the two signals and their biases constrain each other. This reading is consistent with the gate statistics (App. 2.3.1: Spatial dominant at early and mid layers, CrossStep rising at the late layer) and with the parameter-matched Equal-FFN control (85.9), which rules out capacity as the driver.

The negative single-branch scores reflect single-branch training rather than a harmful pathway. At $K{=}1$, where the CrossStep branch reads only the null entry and the Spatial pathway acts alone, the dual-trained model still improves over the baseline ($84.81 \rightarrow 85.29$ PDMS, App. 2.5). We also note that differences of this magnitude under IL are near the noise level of the benchmark; the consistent direction, with both branches negative alone and the dual memory positive, is what supports the interaction reading.

2.3.4 LEARNED TABLE DIAGNOSTICS

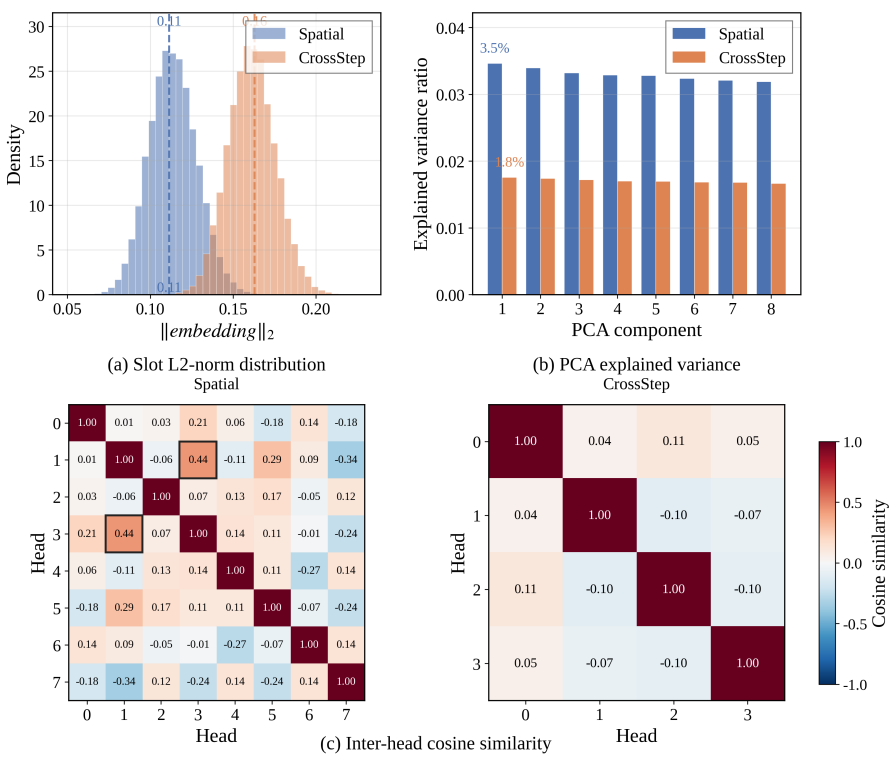
Learned table diagnostics. We probe the learned tables (Fig. 5) to rule out degenerate solutions. Spatial head norms peak around 0.11 and CrossStep around 0.16 in overlapping bell-shaped distributions, so inter-head differences are mainly in direction rather than magnitude. Inter-head cosine similarity is low (CrossStep off-diagonal $\approx \pm 0.1$; Spatial mostly < 0.2, one pair at ~ 0.44), ruling out heads copying the same embedding. Together with the PCA separation between the two branches (Spatial first component $\sim 3.4\%$ of variance vs. CrossStep $\sim 1.7\%$), this is consistent with Spatial being the more structured primary memory and CrossStep a more diffuse dynamics auxiliary, matching the ablation (only-Spatial 85.5, only-CrossStep 85.1, full 86.6).

2.4 TRANSFER TO LLADA-V-8B

Attaching the same memory construction to a masked diffusion VLA Xu et al. (2025) (LLaDA-V-8B; backbone frozen, only Engram trained) yields the PDMS gain reported in Sec. 4.7 of the main text without changing the unmasking budget. The masked-diffusion interface differs from flow matching in two respects: the action is a sequence of discrete tokens unmasked iteratively rather than a continuous trajectory integrated by a velocity field, and there is no cross-step velocity field. We therefore reuse the Spatial branch unchanged, computing BEV displacement n-grams and cross-modal condition hashing from the currently unmasked tokens, and replace the CrossStep branch and the injection adapter with a residual write-back matched to the masked-diffusion block width.

2.5 FULL STEP–COMPUTE–ACCURACY MEASUREMENTS

Fig. 3 in the main text plots PDMS against planner FLOPs for the Flow-IL step ablation. Table 8 here gives the complete per-step measurements behind that figure, including planner and end-to-end latency as well as planner and end-to-end FLOPs. This sweep follows the NAVSIMv1 protocol; the NAVSIMv2 comparison in Table 1 of the main text contrasts the 3-step Engram entries with 5-step baselines. Adding the memory raises PDMS at every sampling budget while planner FLOPs increase by at most 0.04 G, consistent with the asymptotically $\mathcal{O}(1)$ lookup. The 3-step Engram



(a) Slot L2-norm distribution (b) PCA explained variance

(c) Inter-head cosine similarity

Figure 5: Learned table diagnostics on the IL checkpoint. **(a)** Slot ℓ_2-norm distributions for Spatial and CrossStep embeddings peak near 0.11 and 0.16 in overlapping bells, so inter-head differences are mainly directional. **(b)** PCA explained-variance ratios: Spatial PC1 $\approx 3.5\%$ vs. CrossStep $\approx 1.8\%$, indicating a more structured Spatial table and a more diffuse CrossStep auxiliary. **(c)** Inter-head cosine similarity (mean slot prototypes). CrossStep off-diagonals stay near ± 0.1; Spatial entries are mostly < 0.2 (boxed pair ≈ 0.44), ruling out duplicated head embeddings.

Table 8: Step ablation on the NAVSIMv1 navtest split (Flow IL), the full measurements behind Fig. 3 in the main text. Planner latency: single-GPU planner-only wall-clock (CUDA sync), excluding InternVL. E2E latency: online InternVL ($\sim$168 ms at 9 tiles, fixed scene) plus planner; composed from separate VLM-only and planner-only measurements.

	PDMS ↑		Planner (ms) ↓		E2E (ms) ↓		Planner FLOPs (G)		E2E FLOPs (G)	
K	Base.	Ours	Base.	Ours	Base.	Ours	Base.	Ours	Base.	Ours
1	84.81	85.29	18.4	25.1	186.1	192.8	1.81	1.82	2001.8	2001.8
2	85.89	86.13	35.5	48.5	203.2	216.1	3.32	3.34	2003.3	2003.3
3	86.01	**86.64**	54.3	72.3	222.0	240.0	4.83	4.86	2004.8	2004.9
5	**86.19**	86.62	84.4	122.0	252.1	289.7	7.86	7.90	2007.9	2007.9

model (86.64 PDMS) surpasses the 5-step baseline (86.19) at about 38% lower planner FLOPs. End-to-end latency and FLOPs are dominated by the once-per-frame InternVL forward, so the relative planner savings are compressed at the full-stack level.

2.6 More Visualizations

We provide additional qualitative comparisons under a tight sampling budget, covering both generative interfaces used in the paper. In each figure the left column shows the front-view camera with the predicted trajectory overlaid, and the two right columns show the corresponding BEV planning result for the baseline and for the same model with BEV Action Engram; the trajectory color runs from the ego position to the planning horizon.

Fig. 6 compares our flow-matching baseline with the Engram variant across right turn at a busy intersection, and a straight cruise. Without the memory, the baseline trajectory tends to wander between

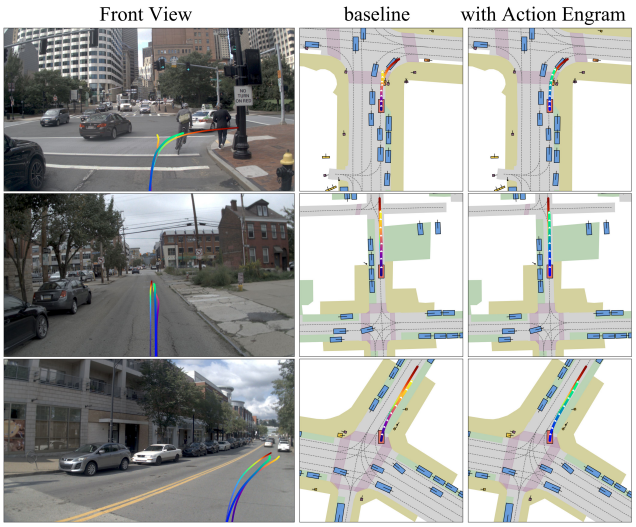


Figure 6: Qualitative comparison on the flow-matching planner at a 3-step budget. Left: front-view camera with the predicted trajectory. Middle: baseline. Right: baseline with BEV Action Engram. The memory yields smoother trajectories that stay closer to the lane center and the drivable area.

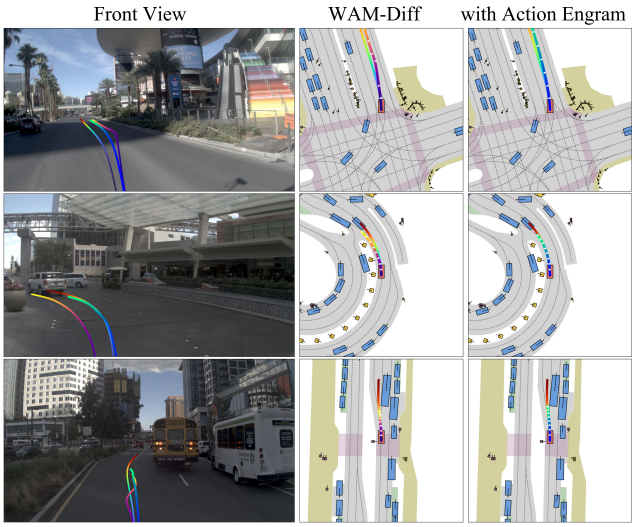


Figure 7: Qualitative comparison on the masked diffusion VLA (LLaDA-V-8B). Left: front-view camera with the predicted trajectory. Middle: baseline. Right: baseline with BEV Action Engram. The memory reduces jitter and keeps the trajectory aligned with the lane, matching the effect observed on the flow-matching planner.

lanes and bends unevenly along the path, and at the intersection it drifts toward the oncoming lane. With BEV Action Engram, the same scenes produce smoother trajectories that stay centered in the drivable corridor and track the road geometry more closely.

Fig. 7 repeats the comparison on the masked diffusion VLA called WAM-Diff Xu et al. (2025) with LLaDA-V-8B, where the action is a sequence of discrete tokens rather than a continuous trajectory. The baseline predictions are jagged and occasionally split across neighboring lanes, whereas attaching the same memory construction produces more coherent and better-aligned trajectories. The improvement transfers across the two generative interfaces, consistent with the interface-agnostic design of the memory.

2.7 Memory Footprint on Vehicle-Grade Hardware

Since the method trades sampling steps for a fixed lookup table, a natural question is whether the added storage is affordable on an in-vehicle platform. The footprint is small and static: the two embedding tables and the projection-and-gate module total about 5.6M parameters (Table 6), roughly 11 MB in half precision and about 5.6 MB in 8-bit, which is about 0.3% of the InternVL3-2B backbone that already runs on the platform. The tables are allocated once and read through an $\mathcal{O}(1)$ gather of a few kilobytes per step, so what the memory removes, a full decoder forward at every saved step, far outweighs what it adds. This is a footprint and complexity analysis rather than an on-device benchmark, and profiling on a specific automotive SoC is left to future work.