MoCAR: Motion-code Coordinate-aware AutoRegression for Continuous Trajectory Forecasting

Yiming Xu[1]* **Hao Cheng**[2] **Monika Sester**[1]
[1]Institute of Cartography and Geoinformatics
Leibniz University Hannover, 30167 Hannover, Germany
[2]Faculty of Geo-Information Science and Earth Observation
University of Twente, 7522 NH Enschede, The Netherlands
{Yiming.Xu,Monika.Sester}@ikg.uni-hannover.de
h.cheng-2@utwente.nl

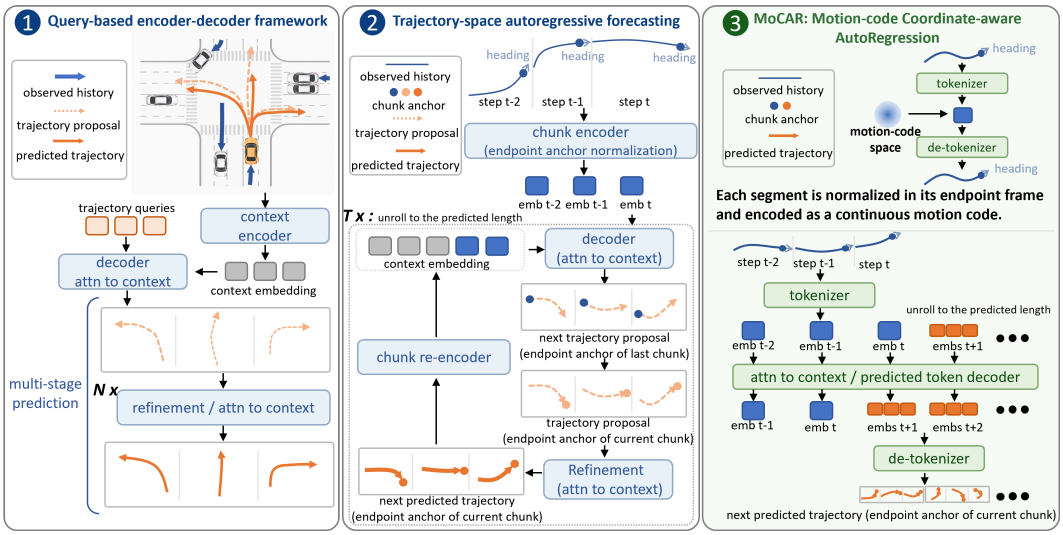


Figure 1: **Paradigm comparison.** ① Query-based encoder–decoder methods generate trajectory proposals and refine them through additional decoding stages. ② Trajectory-space autoregressive methods explicitly decode each sub-trajectory, update the local frame, and re-tokenize the predicted motion for the next step. ③ In contrast, **MoCAR** represents endpoint-normalized trajectory segments as continuous motion codes and performs autoregressive next-code prediction directly in latent space, avoiding explicit trajectory queries and proposal-and-refinement pipelines.

Abstract

Autoregressive generation is natural for language, where predicted tokens can be directly reused as the next prediction state, but trajectory forecasting lacks such a clean token: motion is continuous, multimodal, and expressed in local coordinate frames that evolve with the predicted trajectory. We present **MoCAR** (**Mo**tion-code **C**oordinate-aware **A**uto**R**egression), a *decoder-only* framework that casts trajectory forecasting as next-code prediction in a coordinate-aware continuous latent space. MoCAR learns a continuous motion-code space for endpoint-normalized trajectory segments, where each code jointly captures local trajectory geometry and the reference-frame transition induced by that segment. Historical motion codes are

*Corresponding author: Yiming.Xu@ikg.uni-hannover.de.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

used as a teacher-forced prefix, future codes are generated autoregressively under temporal, map, agent, and mode interactions, and predicted codes persist in latent memory while decoded endpoints update the local scene context. This enables rollout without trajectory-space re-tokenization, trajectory queries, goal candidates, or proposal-and-refinement pipelines. On Argoverse (AV) benchmarks, MoCAR achieves top-tier performance with a simple single-stage architecture, transfers strongly from AV2 to AV1 in zero-shot evaluation, and improves on turn-heavy scenarios. Ablations confirm that the learned continuous motion-code space, latent alignment, weak KL regularization, and joint tokenizer–predictor optimization are essential for stable latent autoregression.

1 Introduction

Trajectory forecasting predicts agents' future motion from their observed histories, road context, and interactions with other agents. As an agent moves, the relevant lanes and neighboring agents change. Autoregressive forecasting can update this context after each predicted segment [1]. The key design choice is what motion representation to pass to the next prediction step. We propose **MoCAR** (**Mo**tion-code **C**oordinate-aware **A**uto**R**egression) to learn continuous motion codes for this purpose. Each code captures local trajectory geometry and the reference-frame transition induced by the segment. Predicted codes directly condition subsequent steps, while decoded endpoints update the local scene context. We refer to this direct reuse of the prediction state as *clean autoregression*.

Learning a reusable motion unit poses three challenges. First, position, velocity, heading, and curvature vary continuously. The representation must preserve the small differences between nearby motions. Second, the same observed history can lead to different futures, such as going straight, turning, or yielding [2, 3, 4]. Third, each predicted segment changes the position and heading used to query nearby lanes and agents [4, 1]. Unlike visual tokens that tile a fixed image grid, a motion token represents a *state transition*. It must describe local trajectory geometry and the relative pose change that establishes the next frame. MoCAR addresses these requirements with coordinate-aware continuous codes and separate rollout branches for multiple possible futures.

Existing forecasting methods use different prediction units and decoding pipelines (Figure 1). Vectorized and query-based encoder–decoder methods achieve strong forecasting performance [3, 5, 6, 7, 4]. Many use goal candidates, motion queries, anchor-informed proposals, or refinement stages [2, 8, 9, 10, 11]. DONUT [1] unifies history encoding and future rollout in a decoder-only model. It updates the local frame along predicted segments, but each step still generates, refines, and re-embeds a trajectory segment. Proposal generation, refinement, and feedback re-encoding introduce additional computation. Task-specific decoding designs can also limit the reuse of learned motion representations across datasets and prediction horizons. MoCAR instead learns continuous motion codes that jointly represent local trajectory shape and the induced frame transition. Each predicted code directly conditions the next prediction, while decoding supplies the trajectory and the next scene anchor. Forecasting therefore follows a shared next-code prediction rule over reusable motion units, without feedback re-encoding or additional refinement.

We learn the motion codes with a variational autoencoder [12]. Each trajectory segment is normalized using the position and heading of its endpoint. The code thus captures local motion geometry and the reference-frame transition induced by the segment. Following DONUT's history-conditioning strategy [1], codes from observed segments form a teacher-forced prefix. A shared causal predictor generates future codes using temporal memory, map context, and interactions among agents and modes. Learned mode embeddings initialize K rollout branches for different future hypotheses. The tokenizer and de-tokenizer are pretrained for reconstruction and then jointly optimized with the forecaster. Weak KL regularization promotes a smooth code space, while latent alignment supervises predicted codes. This yields a single-stage decoder-only forecaster.

The learned code space also provides a reusable vocabulary of local state transitions. Endpoint normalization removes dependence on global position and orientation. This is useful for cross-dataset transfer, where observation and prediction horizons, coordinate conventions, and motion distributions can change [13, 14]. On Argoverse (AV) benchmarks [15, 16], MoCAR achieves top-tier AV2 accuracy, strong AV2-to-AV1 zero-shot transfer, and improved performance on turn-heavy scenes.

Controlled ablations examine code learning and direct code feedback. A matched-stride comparison also shows lower latency than the strong baseline DONUT.

Our contributions are summarized as follows:

- We introduce coordinate-aware continuous motion codes for trajectory segments. Each code represents local motion geometry and the reference-frame transition needed for the next scene update.
- We formulate trajectory forecasting as persistent latent autoregression over these codes. To our knowledge, MoCAR is the first decoder-only trajectory forecaster to directly predict and reuse such codes. It requires no trajectory-space re-tokenization or additional refinement.
- We demonstrate strong AV2 performance, AV2-to-AV1 zero-shot transfer, and gains on turn-heavy scenes. Controlled ablations examine the contributions of motion-code learning and persistent feedback. Matched-stride evaluation shows lower latency than the strong decoder-only baseline DONUT.

Our code is publicly available at `https://github.com/XuYiMing83/MoCAR`.

2 Related Work

Vectorized and query-based motion forecasting. Motion forecasting for autonomous driving has increasingly adopted vectorized scene representations. VectorNet [3], HiVT [5], HDGT [6], Wayformer [7], and QCNet [4] model map-agent interactions using polylines, graphs, and attention [17]. Query-based methods introduce intention localization [2, 18], anchor-informed proposals [8], and decomposed query designs [11]. Other approaches refine trajectory predictions with tube queries [9] or adapt refinement to the scene [10]. These methods use specialized queries, proposals, or refinement modules to generate future trajectories. MoCAR learns reusable local motion units and predicts them sequentially with a shared causal model. Each predicted unit directly conditions the next step as the scene context is updated.

Autoregressive motion forecasting and simulation. MotionLM [19] and Trajeglish [20] model traffic through autoregressive prediction over discrete motion tokens. Both condition their motion decoders on encoded scene context. SMART [21] uses discrete motion tokens with a decoder-only transformer for multi-agent simulation. BehaviorGPT [22] uses a homogeneous autoregressive transformer and predicts trajectory patches. DONUT [1] unifies history encoding and future rollout in a decoder-only forecasting model. Its proposer–refiner module generates trajectory segments. These segments update the local reference frame and are re-embedded as input for the next step. MoCAR directly predicts continuous motion codes and reuses them as the feedback state. Decoded endpoints provide the reference poses for subsequent scene updates. The rollout uses no trajectory-space re-tokenization or additional refinement.

Motion tokenization and latent motion representations. Generative trajectory models include VAE-based latent-variable models [12, 23, 24]. Other approaches use diffusion [25, 26], normalizing flows [27], or flow matching [28]. Many of these models generate a complete future trajectory or use a latent variable for the full prediction horizon. Discrete tokenization offers another way to represent motion [29, 20]. MoCAR learns continuous VAE-based codes for local trajectory segments. Each segment is normalized using the position and heading of its endpoint. Its code captures both local trajectory geometry and the reference-frame transition needed for the next scene update. This continuous representation is learned jointly with the predictor and directly reused during autoregressive rollout.

3 Method

3.1 Problem formulation

We consider a vectorized driving scene with A agents and M map elements. At timestep t, agent i has state $\mathbf{x}_i^t = (\mathbf{p}_i^t, \mathbf{v}_i^t, \theta_i^t, \gamma_i)$. Here, $\mathbf{p}_i^t \in \mathbb{R}^2$ and $\mathbf{v}_i^t \in \mathbb{R}^2$ denote position and velocity. The heading is θ_i^t, and γ_i denotes the agent category. The observed history is $\mathbf{X}_i = \{\mathbf{x}_i^t\}_{t=1}^{T_h}$. The map contains

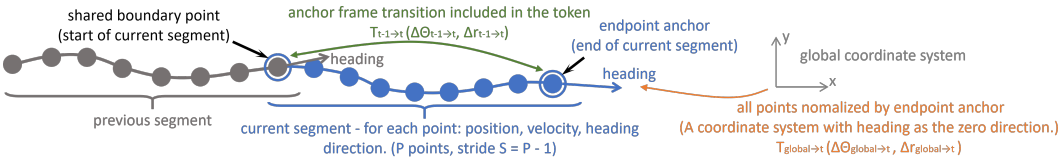


Figure 2: **Coordinate-aware motion segment.** Each local trajectory segment is normalized in the coordinate frame of its endpoint anchor and encoded as a continuous motion code. This code jointly represents local geometry and the induced frame transition, making it the autoregressive prediction unit in MoCAR.

vectorized elements $\mathcal{M} = \{L_j\}_{j=1}^{M}$. Each element contains sampled points and semantic attributes such as lane type or boundary type.

Given $\{\mathbf{X}_i\}_{i=1}^{A}$ and $\mathcal{M}$, the model predicts K future trajectories $\hat{\mathbf{Y}}_i^{(k)} = \{(\hat{\mathbf{p}}_i^{t,k}, \hat{\theta}_i^{t,k})\}_{t=T_h+1}^{T_h+T_f}$. Their probabilities $\{\pi_i^{(k)}\}_{k=1}^{K}$ satisfy $\sum_{k=1}^{K} \pi_i^{(k)} = 1$. We represent motion as a sequence of local segments and forecast by autoregressive next-code prediction in a continuous latent space.

3.2 Coordinate-aware continuous motion codes

We split each trajectory into segments of P points with stride $S = P - 1$. For segment index c, τ_c denotes the endpoint timestep. The segment of agent i is $\mathcal{S}_{i,c} = \{(\mathbf{p}_i^{\tau_c - S + j}, \theta_i^{\tau_c - S + j}, \mathbf{v}_i^{\tau_c - S + j})\}_{j=0}^{P-1}$. Adjacent segments share one boundary point, as shown in Figure 2. Within each segment, we derive transition features including displacement, velocity direction, and heading change.

Each segment is represented in the local coordinate frame of its endpoint anchor $\mathbf{r}_{i,c} = (\mathbf{p}_i^{\tau_c}, \theta_i^{\tau_c})$:

$$\tilde{\mathbf{p}}_{i,c}^{j} = R(-\theta_i^{\tau_c})(\mathbf{p}_i^{\tau_c - S + j} - \mathbf{p}_i^{\tau_c}), \quad \tilde{\theta}_{i,c}^{j} = \mathrm{wrap}(\theta_i^{\tau_c - S + j} - \theta_i^{\tau_c}). \tag{1}$$

Here, $R(\cdot)$ denotes the 2D rotation matrix and $\mathrm{wrap}(\cdot)$ maps angles to $[-\pi, \pi)$. The endpoint lies at the origin, and the local x-axis follows its heading. The normalized segment retains its start pose relative to the endpoint. This relative pose specifies the translation and rotation between consecutive endpoint frames. The motion code therefore jointly represents local trajectory shape and the associated frame transition. This allows the same local motion pattern to be reused at different scene locations and orientations.

We learn the motion-code space with a variational autoencoder. The encoder maps a normalized segment to a Gaussian distribution $q_\phi(\mathbf{z}_{i,c} \mid \tilde{\mathcal{S}}_{i,c}) = \mathcal{N}(\boldsymbol{\mu}_{i,c}, \mathrm{diag}(\boldsymbol{\sigma}_{i,c}^2))$. The decoder reconstructs the local segment as $\hat{\mathcal{S}}_{i,c} = D_\psi(\mathbf{z}_{i,c})$. We call $\mathbf{z}_{i,c}$ a *motion code*, or *token*. The *motion-code space* comprises the tokenizer, de-tokenizer, and their continuous latent representation. We use the deterministic encoder mean $\boldsymbol{\mu}_{i,c}$ for historical conditioning and ground-truth future targets in latent supervision.

Reconstruction pretraining teaches the tokenizer and de-tokenizer to preserve local motion geometry and frame transitions. During forecasting, the decoder receives predicted codes, which may differ from those produced by the encoder. Latent alignment supervises these predictions toward the encoded future segments, while weak KL regularization encourages a smooth motion-code space. Joint optimization then adapts this representation to forecasting. A smaller learning rate for the tokenizer and de-tokenizer helps preserve their learned reconstruction structure.

3.3 Decoder-only motion-code autoregression

Figure 3 illustrates the rollout. Let C_h and C_f denote the numbers of historical and future motion segments. Historical codes are obtained as $\mathbf{z}_{i,c}^{\mathrm{gt}} = \boldsymbol{\mu}_{i,c}$ for $c = 1, \ldots, C_h$. Following DONUT's history-conditioning strategy [1], these codes form a teacher-forced conditioning prefix. The historical marker in Figure 3 identifies this observed prefix. Teacher forcing applies only to the observed history; future prediction is free-running.

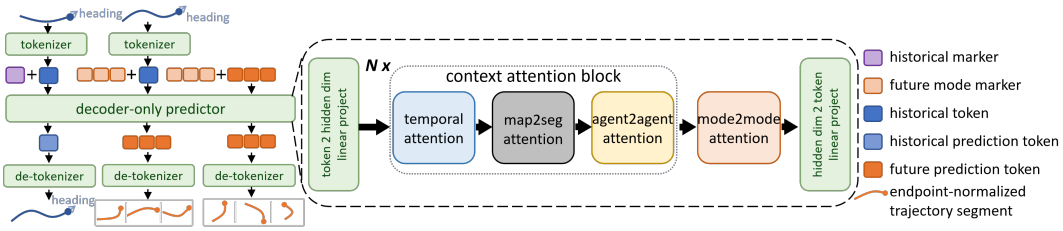


Figure 3: **Decoder-only motion-code autoregression in MoCAR.** Historical segments are tokenized into a teacher-forced motion-code prefix, and future prediction rolls out autoregressively in latent space. Mode embeddings separate latent rollout branches, while the predictor updates each branch using temporal, map, agent, and mode interactions before decoding the next segment and endpoint anchor. MoCAR therefore forecasts directly in code space, without trajectory-space re-tokenization or proposal-and-refinement pipelines.

Future forecasting starts from the last historical code. A mode represents one of the K possible future hypotheses. We initialize a mode-specific hidden state:

$$\mathbf{h}_{i,k,C_h} = W_z \mathbf{z}^{\mathrm{gt}}_{i,C_h} + \mathbf{e}_k, \tag{2}$$

Here, W_z projects a motion code into the predictor's hidden dimension. The learned embedding $\mathbf{e}_k$ separates rollout branches and appears as the future mode marker in Figure 3. At inference, each mode follows a deterministic code rollout conditioned on observed motion and the evolving scene.

At future step $c = C_h + 1, \ldots, C_h + C_f$, the predictor updates the hidden state using scene context and predicts the next motion code. The code is decoded into a segment and projected back to hidden space:

$$\tilde{\mathbf{h}}_{i,k,c} = F_\Theta(\mathbf{h}_{i,k,c-1}, \mathcal{C}_c), \quad \hat{\mathbf{z}}_{i,k,c} = W_o \tilde{\mathbf{h}}_{i,k,c}, \quad \hat{\mathcal{S}}_{i,k,c} = D_\psi(\hat{\mathbf{z}}_{i,k,c}), \quad \mathbf{h}_{i,k,c} = W_z \hat{\mathbf{z}}_{i,k,c} + \mathbf{e}_k. \tag{3}$$

Here, $\hat{\cdot}$ denotes predicted quantities. The scene context $\mathcal{C}_c$ includes temporal memory, map elements, neighboring agents, and other modes.

Persistent latent autoregression. The predicted motion code $\hat{\mathbf{z}}_{i,k,c}$ directly forms the next hidden state through W_z and the mode embedding. The decoded segment supplies the trajectory and endpoint anchor $\hat{\mathbf{r}}_{i,k,c}$ for the next scene update. Using these codes as autoregressive states lets the predictor model successive local motions together with their changes of reference frame. This persistent code feedback avoids trajectory-space re-tokenization. Each code decodes to a motion segment without an additional refinement stage. Temporal attention separately reuses per-layer caches of projected keys and values from earlier hidden states (Appendix A.4).

3.4 Scene-conditioned latent update

Scene context is built around the latest available endpoint anchor. The first future step uses the last historical anchor; later steps use the preceding predicted anchor. The map is encoded into features $\mathbf{G} = \{\mathbf{g}_j\}_{j=1}^{M}$. Each map element is anchored at the midpoint of its entrance boundary, with the entrance direction as its heading. Local interaction graphs are updated around the current anchor during rollout.

Map-to-segment edges connect the current hidden state to nearby map elements. Agent-to-agent edges connect nearby agents within the same mode. Temporal edges connect the current hidden state to the states of historical and previously predicted tokens. Mode-to-mode edges connect the hidden states of different hypotheses for the same agent [4, 11]. These scene-conditioning components are adapted from prior work. MoCAR's contribution is the coordinate-aware continuous motion-code formulation and its persistent autoregressive feedback.

For an edge from source u to target v with anchor poses $(\mathbf{p}_u, \theta_u)$ and $(\mathbf{p}_v, \theta_v)$ [4], we encode

$$\mathbf{r}_{u \to v} = \left[\|\mathbf{p}_u - \mathbf{p}_v\|_2,\ \angle(\mathbf{d}_v, \mathbf{p}_u - \mathbf{p}_v),\ \mathrm{wrap}(\theta_u - \theta_v),\ \Delta c \right], \tag{4}$$

Here, $\mathbf{d}_v = (\cos\theta_v, \sin\theta_v)$ is the heading direction, and $\angle(\cdot,\cdot)$ denotes the signed bearing angle. The segment-time offset Δc is used only for temporal edges. This encoding describes u in v's local frame. Fourier features embed the relative geometry for attention [30, 31].

The predictor applies a stack of context-update blocks to the hidden state. Each block applies temporal, map-to-segment, and agent-to-agent attention in sequence. Mode-to-mode attention is applied once after the final block. Relative geometric encodings modulate the key and value projections in the temporal, map, and agent attention layers.

At the final future prediction step, an MLP produces $\pi_i^{(k)}$ from each mode's updated hidden state. These probabilities score the K complete predicted trajectories. Appendix A.4 provides the detailed computations.

3.5 Training objectives

The motion-code tokenizer and de-tokenizer are trained with a VAE objective

$$\mathcal{L}_{\text{vae}} = \mathcal{L}_{\text{rec}} + \beta_{\text{kl}}\mathcal{L}_{\text{kl}}, \quad \mathcal{L}_{\text{kl}} = \frac{1}{2}\sum_d(\mu_d^2 + \sigma_d^2 - \log\sigma_d^2 - 1), \tag{5}$$

Here, $\mathcal{L}_{\text{rec}}$ supervises local segment positions, headings, and velocities. It also includes auxiliary bridge supervision for segment connection (Appendix A.3).

For future trajectory supervision, we use winner-takes-all. The best mode is $k_i^\star = \arg\min_k \sum_{t=T_h+1}^{T_h+T_f} \|\hat{\mathbf{p}}_i^{t,k} - \mathbf{p}_i^t\|_2$. This mode receives the position and heading losses

$$\mathcal{L}_{\text{traj}} = \sum_{i,t} \|\hat{\mathbf{p}}_i^{t,k_i^\star} - \mathbf{p}_i^t\|_1, \quad \mathcal{L}_{\text{head}} = \sum_{i,t} |\text{wrap}(\hat{\theta}_i^{t,k_i^\star} - \theta_i^t)|, \tag{6}$$

where the time sums cover the future horizon.

We also align predicted future codes with encoder means from the corresponding ground-truth segments:

$$\mathcal{L}_{\text{align}} = \sum_{i,c} \|\hat{\mathbf{z}}_{i,k_i^\star,c} - \boldsymbol{\mu}_{i,c}^{\text{gt}}\|_2^2. \tag{7}$$

Here, c indexes future segments. Each target $\boldsymbol{\mu}_{i,c}^{\text{gt}}$ is computed from the ground-truth segment normalized in its own endpoint frame. These encoder means supply training targets; rollout uses predicted codes and anchors.

Mode probabilities are trained with a distance-based soft classification loss

$$\mathcal{L}_{\text{cls}} = -\sum_i \log \sum_{k=1}^{K} \pi_i^{(k)} \exp\left(-\tau \|\hat{\mathbf{p}}_i^{T_h+T_f,k} - \mathbf{p}_i^{T_h+T_f}\|_2\right). \tag{8}$$

Historical segments also receive position and heading supervision. Their weighted loss is $\mathcal{L}_{\text{hist}}$, with coefficients listed in Appendix A.5. The total objective is $\mathcal{L} = \lambda_{\text{vae}}\mathcal{L}_{\text{vae}} + \lambda_{\text{traj}}\mathcal{L}_{\text{traj}} + \lambda_{\text{head}}\mathcal{L}_{\text{head}} + \lambda_{\text{align}}\mathcal{L}_{\text{align}} + \lambda_{\text{cls}}\mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{hist}}$.

4 Experiments

We first evaluate forecasting accuracy, zero-shot transfer, and performance on turn-heavy scenes. We then examine computational cost and motion-code structure, followed by controlled ablations of the representation and autoregressive feedback.

4.1 Experimental setup

Datasets. We evaluate on Argoverse 2 (AV2) [15] and Argoverse 1 (AV1) [16]. AV2 contains 250k scenarios collected across six cities with 5 s of history and 6 s prediction; AV1 has 2 s of observation and 3 s prediction. We use AV2 as the primary benchmark; AV1 is used to evaluate cross-dataset transfer and the reusability of the learned motion-code representation across different observation and prediction horizons.

Evaluation metrics. Following standard protocols [32, 5, 4], we report minADE$_K$, minFDE$_K$, b-minFDE$_K$, and MR$_K$ with $K = 6$ unless otherwise specified. All ablations use the AV2 validation set unless otherwise stated.

Table 1: Comparison with state-of-the-art non-ensemble methods on the AV2 test set. MoCAR reaches DeMo/DONUT-level performance without trajectory queries or explicit refinement. Best results are in bold, and second-best results are underlined.

Method	minFDE$_1\downarrow$	minADE$_1\downarrow$	minFDE$_6\downarrow$	minADE$_6\downarrow$	MR$_6\downarrow$	b-minFDE$_6\downarrow$
FRM [33]	5.93	2.37	1.81	0.89	0.29	2.47
HDGT [6]	5.37	2.08	1.60	0.84	0.21	2.24
MTR [2]	4.39	1.74	1.44	0.73	0.15	1.98
HPTR [34]	4.61	1.84	1.43	0.73	0.19	2.03
HeteroGCN [35]	4.40	1.72	1.34	0.69	0.18	1.90
ProphNet [8]	4.74	1.80	1.33	0.68	0.18	1.88
HiMAP [36]	4.61	1.81	1.33	0.68	0.17	1.96
QCNet [4]	4.30	1.69	1.29	0.65	0.16	1.91
SmartRefine [10]	4.17	1.65	1.23	0.63	0.15	1.86
DeMo [11]	**3.74**	**1.49**	<u>1.17</u>	**0.61**	**0.13**	1.84
DONUT [1]	<u>4.06</u>	1.66	**1.16**	0.63	<u>0.14</u>	**1.79**
MoCAR (Ours)	4.08	<u>1.61</u>	<u>1.17</u>	<u>0.62</u>	**0.13**	<u>1.83</u>

Table 2: Zero-shot transfer to AV1 validation. All methods use AV2-trained checkpoints with identical preprocessing. Full AV1 evaluation appears in Appendix F.

Method (AV2→AV1, zero-shot)	minFDE$_1\downarrow$	minADE$_1\downarrow$	minFDE$_6\downarrow$	minADE$_6\downarrow$	MR$_6\downarrow$	b-minFDE$_6\downarrow$
DeMo [11]	5.70	2.52	2.03	1.08	0.25	2.74
QCNet [4]	5.60	2.51	<u>1.55</u>	<u>0.95</u>	<u>0.19</u>	2.27
DONUT [1]	**5.05**	**2.32**	1.58	0.97	<u>0.19</u>	<u>2.26</u>
MoCAR (Ours)	<u>5.14</u>	<u>2.37</u>	**1.24**	**0.72**	**0.11**	**1.93**

Implementation details. We use $K = 6$ modes, hidden dimension 128, 8 attention heads, and 24-D motion codes. Each segment contains $P = 16$ points with stride $S = P - 1 = 15$ by default, so the 60-step AV2 future is represented by four codes and the 30-step AV1 future by two. After reconstruction pretraining, the tokenizer and de-tokenizer are jointly optimized with the predictor at a smaller learning rate. During rollout, local map and agent interaction graphs are rebuilt around the current predicted anchor. We train with AdamW and cosine decay, without ensembling or test-time augmentation. Appendix A provides the full training configuration, and Appendix F details the shared AV1 preprocessing used by all compared methods.

4.2 Main results

Comparison with state-of-the-art methods. Table 1 compares MoCAR with representative non-ensemble methods on the AV2 test set. MoCAR reaches 1.17 minFDE$_6$, 0.62 minADE$_6$, 0.13 MR$_6$, and 1.83 b-minFDE$_6$. It matches the best MR$_6$, tying or near-tying the remaining multimodal metrics. This performance is achieved with a single-stage decoder-only next-code formulation, without trajectory-space re-tokenization or additional refinement.

Table 3: Performance on the AV2 validation set stratified by future turning angle. The turn-heavy subset contains samples whose ground-truth future trajectory has a heading change of at least 45°.

Method	Subset	minFDE$_6\downarrow$	minADE$_6\downarrow$	MR$_6\downarrow$	b-minFDE$_6\downarrow$
QCNet [4]	Turn-heavy ($\geq 45^\circ$)	2.37	1.17	0.36	2.98
	Lower-turning ($< 45^\circ$)	<u>1.10</u>	<u>0.66</u>	<u>0.13</u>	<u>1.72</u>
	All	1.25	<u>0.72</u>	0.16	1.87
DONUT [1]	Turn-heavy ($\geq 45^\circ$)	<u>2.01</u>	<u>1.10</u>	<u>0.31</u>	**2.64**
	Lower-turning ($< 45^\circ$)	**1.07**	0.67	**0.12**	**1.70**
	All	<u>1.18</u>	0.73	<u>0.14</u>	**1.81**
MoCAR	Turn-heavy ($\geq 45^\circ$)	**1.99**	**1.07**	**0.28**	<u>2.67</u>
	Lower-turning ($< 45^\circ$)	**1.07**	**0.65**	**0.12**	<u>1.72</u>
	All	**1.17**	**0.70**	**0.13**	<u>1.84</u>

Cross-dataset zero-shot transfer to AV1. If a coordinate-aware motion-code space captures reusable local motion primitives rather than dataset-specific trajectory statistics, it should transfer

Table 4: Parameter count and GPU inference latency on a single Nvidia RTX 4090, measured per scenario with batch size $B = 1$. Stride is the number of timesteps advanced per segment and does not apply to QCNet.

Method	Stride	#Params	GPU inference (ms)
QCNet [4]	–	7.7M	35.6
DONUT [1]	10	9.0M	126.7
MoCAR (matched stride)	10	6.3M	97.9
MoCAR (default)	15	6.3M	66.8

across datasets. We test this by directly evaluating AV2-trained checkpoints on AV1 without fine-tuning. This setting is challenging because the datasets differ in observation and prediction horizons, available motion states, and agent-label conventions. All methods use the same preprocessing and masking protocol. Velocity and heading are derived from smoothed historical position sequences, and unavailable historical steps are masked out. Metrics are computed over the original 3 s AV1 future. The shared 10 Hz sampling rate preserves the 1.5 s duration of each 16-point segment. Appendix F details preprocessing and agent-label mapping.

As shown in Table 2, MoCAR achieves the best results on all four multimodal metrics. It reaches 1.24 minFDE_6, compared with 1.55 for QCNet, the strongest external baseline on this metric. This supports our hypothesis that endpoint-normalized continuous motion codes provide a reusable autoregressive vocabulary across different horizons and coordinate conventions. A further ablation shows that training-time positional-noise augmentation benefits zero-shot transfer, consistent with improved robustness to position-derived motion inputs (Appendix F).

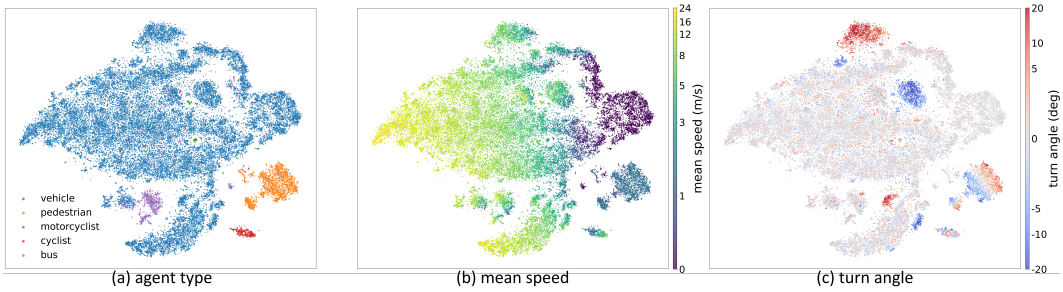
Performance on turn-heavy scenarios. On the AV2 validation set, we evaluate 3,007 cases whose ground-truth future heading change is at least 45°. QCNet and DONUT serve as encoder–decoder and autoregressive baselines. This subset stresses two properties of the forecasting formulation. Large turns require map context and neighboring agents to be updated along the future path, motivating step-wise autoregressive rollout. They also induce large reference-frame changes between consecutive segments, making autoregressive-state consistency important. DONUT updates context during trajectory-space rollout, while MoCAR additionally keeps each generated code as a persistent latent token.

As shown in Table 3, all methods degrade on turn-heavy cases. MoCAR achieves the lowest minFDE_6, minADE_6, and MR_6 among the compared methods and remains comparable on lower-turning cases. These results support coordinate-aware latent autoregression for complex maneuvers. Additional refinement comparisons and qualitative visualizations are provided in Appendices E and G.

4.3 Further analysis

Computational cost. Table 4 reports parameter count and GPU inference latency for representative encoder–decoder and autoregressive baselines, QCNet and DONUT, on a single Nvidia RTX 4090. MoCAR uses per-layer K/V caching and avoids feedback re-encoding, while retaining decoding for trajectories and scene updates. At a matched stride of 10, it takes 97.9 ms per scenario, a 22.7% latency reduction relative to DONUT under the same timing protocol. The default stride of 15 uses fewer rollout steps and takes 66.8 ms. Compared with QCNet, MoCAR incurs extra cost from autoregressive context updates, but retains real-time GPU inference speed ($> 10\,\text{fps}$) and supports step-wise map and interaction reasoning under evolving local frames.

Structure of the learned motion-code space. To inspect whether the learned codes form a meaningful motion vocabulary, we apply t-SNE [37] to the encoder means of 50,000 AV2 validation segments (Figure 4). The embedding is structured by both agent type and motion dynamics: vehicle segments form the dominant manifold, while pedestrians, cyclists, and other vulnerable road users occupy more peripheral regions. Coloring the same embedding by speed and turn angle reveals smooth local variation rather than fragmented clusters. Pedestrians concentrate in lower-speed regions but exhibit diverse turning directions, whereas vehicles and buses show more regular turn-angle patterns, consistent with stronger constraints from road geometry and traffic rules. Cyclists occupy mixed regions with relatively high speeds and varied turns. These patterns suggest that MoCAR



(a) agent type (b) mean speed (c) turn angle

Figure 4: **Learned motion-code space.** t-SNE visualization of encoder means from 50,000 AV2 validation segments, colored by agent type, mean speed, and turn angle. The learned space organizes segments by agent category and motion dynamics, suggesting that the latent codes capture fine-grained motion primitives.

Table 5: Ablation on motion-code design on the AV2 validation set.

	Variant	minFDE$_6$↓	minADE$_6$↓	MR$_6$↓	b-minFDE$_6$↓
Code dim	z dim = 16	1.23	0.72	0.153	1.88
	z dim = 24	**1.17**	**0.70**	**0.134**	**1.84**
	z dim = 32	1.18	**0.70**	0.138	1.85
Stride	stride = 10	1.18	**0.70**	**0.133**	**1.84**
	stride = 15	**1.17**	**0.70**	0.134	**1.84**
	stride = 20	1.18	**0.70**	0.138	1.85
	stride = 30	1.20	0.71	0.144	1.86

learns a continuous motion vocabulary that preserves fine-grained similarities across agent types, speeds, and curvatures, complementing the quantitative ablations in Section 4.4.

Motion-code reconstruction. Across motion categories on AV2 validation, mean position reconstruction error over local 1.5 s segments ranges from 0.02 m for stationary motion to 0.16 m for very sharp turns. These results indicate that the codes preserve local motion geometry. Appendix D provides the full breakdown of position and heading errors by motion type.

4.4 Ablation studies

Motion-code design. Table 5 examines how code capacity and segment granularity affect forecasting accuracy and the number of autoregressive steps. Increasing the code dimension from 16 to 24 improves minFDE$_6$ from 1.23 to 1.17. A further increase to 32 brings no additional gain. Thus, $z = 24$ strikes a good balance between representational capacity and predictability. Varying the stride at $z = 24$ shows a complementary trade-off. We set $P = S + 1$ so adjacent segments always share one boundary point; longer strides shorten the rollout but gradually degrade performance. Strides 10 and 15 give comparable accuracy, and we choose stride 15 for fewer rollout steps and lower latency (Table 4). Both start and endpoint anchoring use local pose normalization. Endpoint anchoring expresses each segment in the latest pose used for the next scene update and improves over start anchoring (Appendix B.2).

Motion-code learning and feedback. Table 6 examines the motion-code representation, its training objectives, and the feedback state used during rollout. We first compare the full model with two forecasting-only controls: a lightweight MLP encoder–decoder over flattened endpoint-normalized step features, and the same GRU tokenizer–de-tokenizer used by MoCAR. Both retain a learned continuous autoregressive interface, but omit reconstruction pretraining, reconstruction loss, KL regularization, and latent alignment. Both controls perform worse than the full model. With the same GRU architecture, the full motion-code training scheme reduces minFDE$_6$ from 1.27 to 1.17. This comparison demonstrates the benefit of the learning scheme beyond the tokenizer architecture.

Removing $\mathcal{L}_{\text{align}}$ degrades forecasting, showing that predicted codes benefit from supervision toward ground-truth encoder codes. Removing the KL regularizer effectively turns the tokenizer into an autoencoder and causes a more pronounced drop, consistent with the benefit of a smooth latent

Table 6: Ablation on motion-code representation, training, and feedback on AV2 validation. Best results are in bold.

Variant	minFDE$_6$↓	minADE$_6$↓	MR$_6$↓	b-minFDE$_6$↓
Raw geometric MLP encoder–decoder	1.29	0.73	0.17	1.92
Same-architecture GRU (forecasting only)	1.27	0.73	0.16	1.91
w/o latent-alignment loss $\mathcal{L}_{\text{align}}$	1.23	0.72	0.15	1.89
Autoencoder tokenizer ($\beta_{\text{kl}} = 0$)	1.29	0.75	0.16	1.95
Frozen pretrained tokenizer–de-tokenizer	1.25	0.73	0.15	1.90
MoCAR (re-encoding feedback)	1.21	0.72	0.15	1.87
MoCAR (full)	**1.17**	**0.70**	**0.13**	**1.84**

space during rollout. Freezing the pretrained tokenizer–de-tokenizer also hurts, indicating that reconstruction-only pretraining is not fully aligned with forecasting. Together, these results show the importance of latent alignment, weak KL regularization, and joint tokenizer–predictor optimization for latent autoregressive forecasting. A weight comparison in Appendix A.5 also favors weak latent alignment over both no alignment and a stronger penalty.

We also compare direct code reuse with re-encoding decoded segments, keeping the predictor, VAE, losses, data, and training schedule fixed. Both variants decode segments for anchor and scene updates; only the state passed to the next prediction changes. Direct reuse improves all four 6 s metrics, reducing minFDE$_6$ from 1.21 to 1.17. Short-horizon errors are comparable, while the benefit of direct reuse becomes clearer later in the rollout (Appendix B.3). This controlled comparison supports preserving the predicted motion code as a persistent state over repeated rollout.

5 Conclusion

MoCAR formulates trajectory forecasting as next-code prediction over continuous motion codes that jointly represent local trajectory geometry and the induced reference-frame transition. Predicted codes directly condition subsequent predictions, while decoded endpoints update the scene context in a single-stage decoder-only architecture. Experiments show strong AV2 forecasting accuracy and AV2-to-AV1 zero-shot transfer, with improved performance on turn-heavy scenes. MoCAR also achieves lower latency than DONUT at a matched stride. Controlled ablations support joint representation learning and persistent code reuse during rollout. More broadly, MoCAR suggests that endpoint-normalized continuous motion codes can serve as reusable state-transition tokens for autoregressive modeling of physical motion. This abstraction may extend beyond road-agent forecasting, since many robotic, aerial, and embodied systems also require continuous rollout under evolving local coordinate frames.

Limitations. Motion-code and frame-estimation errors can accumulate during rollout, particularly for rare or sharp maneuvers. Transfer has been evaluated only from AV2 to AV1 at a shared 10 Hz sampling rate; other rates require temporal resampling or adaptation of the code duration. Mode scores are evaluated for ranking rather than calibrated uncertainty, and their suitability for downstream planning remains untested.

Acknowledgments and Disclosure of Funding

This work was supported by the German Research Foundation (DFG) as part of the Research Training Group i.c.sens [GRK2159].

References

[1] Markus Knoche, Daan de Geus, and Bastian Leibe. Donut: A decoder-only model for trajectory prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 28903–28912, 2025.

[2] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Motion transformer with global intention localization and local movement refinement. *Advances in Neural Information Processing Systems*, 35:6531–6543, 2022.

[3] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. VectorNet: Encoding HD maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11525–11533, 2020.

[4] Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-centric trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17863–17873, 2023.

[5] Zikang Zhou, Luyao Ye, Jianping Wang, Kui Wu, and Kejie Lu. HiVT: Hierarchical vector transformer for multi-agent motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8823–8833, 2022.

[6] Xiaosong Jia, Penghao Wu, Li Chen, Yu Liu, Hongyang Li, and Junchi Yan. HDGT: Heterogeneous driving graph transformer for multi-agent trajectory prediction via scene encoding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):13860–13875, 2023.

[7] Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S Refaat, and Benjamin Sapp. Wayformer: Motion forecasting via simple and efficient attention networks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2980–2987. IEEE, 2023.

[8] Xishun Wang, Tong Su, Fang Da, and Xiaodong Yang. Prophnet: Efficient agent-centric motion forecasting with anchor-informed proposals. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21995–22003, 2023.

[9] Sehwan Choi, Jungho Kim, Junyong Yun, and Jun Won Choi. R-pred: Two-stage motion prediction via tube-query attention-based trajectory refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8525–8535, 2023.

[10] Yang Zhou, Hao Shao, Letian Wang, Steven L Waslander, Hongsheng Li, and Yu Liu. SmartRefine: A scenario-adaptive refinement framework for efficient motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15281–15290, 2024.

[11] Bozhou Zhang, Nan Song, and Li Zhang. DeMo: Decoupling motion forecasting into directional intentions and dynamic states. *Advances in Neural Information Processing Systems*, 37:106582–106606, 2024.

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[13] Lan Feng, Mohammadhossein Bahari, Kaouther Messaoud Ben Amor, Éloi Zablocki, Matthieu Cord, and Alexandre Alahi. Unitraj: A unified framework for scalable vehicle trajectory prediction. In *European Conference on Computer Vision*, pages 106–123. Springer, 2024.

[14] Yue Yao, Shengchao Yan, Daniel Goehring, Wolfram Burgard, and Joerg Reichardt. Improving out-of-distribution generalization of trajectory prediction for autonomous driving via polynomial representations. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 488–495. IEEE, 2024.

[15] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*, 2021.

[16] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[18] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. MTR++: Multi-agent motion prediction with symmetric scene modeling and guided intention querying. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):3955–3971, 2024.

[19] Ari Seff, Brian Cera, Dian Chen, Mason Ng, Aurick Zhou, Nigamaa Nayakanti, Khaled S Refaat, Rami Al-Rfou, and Benjamin Sapp. MotionLM: Multi-agent motion forecasting as language modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8579–8590, 2023.

[20] Jonah Philion, Xue Bin Peng, and Sanja Fidler. Trajeglish: Traffic modeling as next-token prediction. In *The Twelfth International Conference on Learning Representations*, 2024.

[21] Wei Wu, Xiaoxin Feng, Ziyan Gao, and Yuheng KAN. SMART: Scalable multi-agent real-time motion generation via next-token prediction. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[22] Zikang Zhou, Haibo Hu, Xinhong Chen, Jianping Wang, Nan Guan, Kui Wu, Yung-Hui Li, Yu-Kai Huang, and Chun J Xue. Behaviorgpt: Smart agent simulation for autonomous driving with next-patch prediction. *Advances in Neural Information Processing Systems*, 37:79597–79617, 2024.

[23] Pei Xu, Jean-Bernard Hayet, and Ioannis Karamouzas. SocialVAE: Human trajectory prediction using timewise latents. In *European Conference on Computer Vision*, pages 511–528. Springer, 2022.

[24] Wei Xiang, Haoteng Yin, He Wang, and Xiaogang Jin. SocialCVAE: Predicting pedestrian trajectory via interaction conditioned latents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 6216–6224, 2024.

[25] Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, and Dragomir Anguelov. Motion-Diffuser: Controllable multi-agent motion prediction using diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9644–9653, 2023.

[26] Yiming Xu, Hao Cheng, and Monika Sester. Controllable diverse sampling for diffusion based motion behavior forecasting. In *2024 IEEE Intelligent Vehicles Symposium (IV)*, pages 2397–2404. IEEE, 2024.

[27] Rongqin Liang, Yuanman Li, Jiantao Zhou, and Xia Li. STGlow: A flow-based generative framework with dual-graphormer for pedestrian trajectory prediction. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[28] Qi Yan, Brian Zhang, Yutong Zhang, Daniel Yang, Joshua White, Di Chen, Jiachao Liu, Langechuan Liu, Binnan Zhuang, Shaoshuai Shi, et al. TrajFlow: Multi-modal motion prediction via flow matching. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3923–3928. IEEE, 2025.

[29] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[30] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[31] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

[32] Junru Gu, Chen Sun, and Hang Zhao. DenseTNT: End-to-end trajectory prediction from dense goal sets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15303–15312, 2021.

[33] Daehee Park, Hobin Ryu, Yunseo Yang, Jegyeong Cho, Jiwon Kim, and Kuk-Jin Yoon. Leveraging future relationship reasoning for vehicle trajectory prediction. In *The Eleventh International Conference on Learning Representations*, 2023.

[34] Zhejun Zhang, Alexander Liniger, Christos Sakaridis, Fisher Yu, and Luc Van Gool. Real-time motion prediction via heterogeneous polyline transformer with relative pose encoding. *Advances in Neural Information Processing Systems*, 36:57481–57499, 2023.

[35] Xing Gao, Xiaogang Jia, Yikang Li, and Hongkai Xiong. Dynamic scenario representation learning for motion forecasting with heterogeneous graph convolutional recurrent networks. *IEEE Robotics and Automation Letters*, 8(5):2946–2953, 2023.

[36] Yiming Xu, Yi Yang, Hao Cheng, and Monika Sester. HiMAP: History-aware map-occupancy prediction with fallback. *arXiv preprint arXiv:2602.17231*, 2026.

[37] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[38] Hang Zhao, Jiyang Gao, Tian Lan, Chen Sun, Ben Sapp, Balakrishnan Varadarajan, Yue Shen, Yi Shen, Yuning Chai, Cordelia Schmid, et al. TNT: Target-driven trajectory prediction. In *Conference on Robot Learning*, pages 895–904. PMLR, 2021.

[39] Wenyuan Zeng, Ming Liang, Renjie Liao, and Raquel Urtasun. LaneRCNN: Distributed representations for graph-centric motion forecasting. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 532–539. IEEE, 2021.

[40] Yicheng Liu, Jinghuai Zhang, Liangji Fang, Qinhong Jiang, and Bolei Zhou. Multimodal motion prediction with stacked transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7577–7586, 2021.

[41] Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. Learning lane graph representations for motion forecasting. In *European Conference on Computer Vision*, pages 541–556. Springer, 2020.

[42] Prarthana Bhattacharyya, Chengjie Huang, and Krzysztof Czarnecki. SSL-Lanes: Self-supervised learning for motion forecasting in autonomous driving. In *Conference on Robot Learning*, pages 1793–1805. PMLR, 2023.

[43] Görkay Aydemir, Adil Kaan Akan, and Fatma Güney. ADAPT: Efficient multi-agent trajectory prediction with adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8295–8305, 2023.

[44] Mengmeng Liu, Hao Cheng, Lin Chen, Hellward Broszio, Jiangtao Li, Runjiang Zhao, Monika Sester, and Michael Ying Yang. LAformer: Trajectory prediction for autonomous driving with lane-aware scene constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshop*, pages 2039–2049, 2024.

[45] Xiaolong Tang, Meina Kan, Shiguang Shan, Zhilong Ji, Jinfeng Bai, and Xilin Chen. HPNet: Dynamic trajectory forecasting with historical prediction attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15261–15270, 2024.

A Implementation details

A.1 Motion segments and step features

We use $P = 16$ points per segment with a stride $S = 15$, so adjacent segments share one boundary point. With AV2's $T_h = 50$ historical and $T_f = 60$ future steps, this defines four historical segment slots and four future anchors covering the prediction horizon. The last historical slot corresponds to the current state. The number of valid historical motion codes depends on the available observations. When a segment contains fewer than P valid points near sequence boundaries or due to missing observations, invalid positions are padded and excluded by the step mask. Segments with at least three valid points are still encoded by the tokenizer. Its GRU reads only valid steps through the step mask, and reconstruction losses are computed only on valid points. Segments with fewer than three valid points are masked out.

For each segment, we extract per-step transition features in the endpoint-anchor frame: the local position, local heading, step displacement and its direction, heading change, speed magnitude, and velocity direction relative to the anchor heading. These 8-D features provide a redundant but compact description of step-level kinematics that the tokenizer compresses into a single 24-D code.

We additionally supervise a *bridge target*: the local position and heading of the segment's first point in the endpoint-anchor frame. By construction, this equals the previous segment's endpoint pose expressed in the current frame, and is used by an auxiliary head of the de-tokenizer (Section A.3) to enforce inter-segment continuity.

A.2 VAE encoder (tokenizer)

The tokenizer takes the per-step features of a segment together with an agent-type embedding and produces a Gaussian latent $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ with dimension 24. Each 8-D step feature is lifted to a hidden dimension $D = 128$ via a Fourier embedding [30, 31] (64 frequency bands) and a per-dimension MLP. The resulting sequence is processed by a single-layer GRU. For variable-length segments, the step mask determines the valid sequence length; padded transitions are ignored by packed GRU encoding. The final hidden state, combined with the agent-type embedding, is mapped to $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ by two linear heads.

A.3 VAE decoder (de-tokenizer)

The de-tokenizer takes a 24-D code z and reconstructs the segment in the endpoint-anchor frame. It maintains P learned timestep embeddings. Each embedding is concatenated with z and lifted to dimension D by a backbone MLP. A GRU processes the resulting length-P feature sequence. Three heads act on this sequence to predict backward displacements $\Delta\mathbf{p}_j^{\text{back}} \in \mathbb{R}^2$, velocity, and heading. An auxiliary *bridge* head maps z directly to the segment's first-point local pose.

The local relative positions are reconstructed by reverse cumulative summation of the backward displacements,

$$\tilde{\mathbf{p}}^j = -\sum_{j'=j+1}^{P-1} \Delta\mathbf{p}_{j'}^{\text{back}}, \tag{9}$$

which guarantees $\tilde{\mathbf{p}}^{P-1} = \mathbf{0}$ exactly. Errors thus accumulate *toward the past* rather than away from a noisy first point, keeping the endpoint anchor as a precise integration starting point. The bridge head is functionally redundant with $\tilde{\mathbf{p}}^0$ and is used only as an auxiliary training signal to encourage segment-to-segment continuity. It is not used during autoregressive rollout; endpoint-anchor updates are computed from the decoded displacement and heading sequence.

Bridge supervision. For a valid pair of adjacent segments indexed by i, let $(\mathbf{b}_i, \phi_i)$ be the preceding endpoint pose expressed in the current segment's endpoint frame, and let $(\hat{\mathbf{b}}_i, \hat{\phi}_i)$ be the auxiliary head's prediction. The bridge loss is

$$\begin{aligned}
\mathcal{L}_{\text{bridge}} &= \mathcal{L}_{\text{bridge,pos}} + \mathcal{L}_{\text{bridge,heading}}, \\
\mathcal{L}_{\text{bridge,pos}} &= \frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} w_i \|\hat{\mathbf{b}}_i - \mathbf{b}_i\|_2^2, \\
\mathcal{L}_{\text{bridge,heading}} &= \frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} w_i \,\text{wrap}(\hat{\phi}_i - \phi_i)^2,
\end{aligned} \tag{10}$$

where $\mathcal{V}$ contains valid adjacent-segment pairs and w_i is the same turn-dependent sample weight used for segment reconstruction. The wrapping operation maps the angular difference to $[-\pi, \pi)$. The weight is computed from the target bridge heading:

$$w_i = 1 + \alpha \min\left(\frac{\max(|\phi_i| - \theta_0,\, 0)}{\theta_s},\, 2\right), \qquad \theta_0 = \frac{\pi}{20}, \quad \theta_s = 0.35. \tag{11}$$

All angles are in radians, so θ_0 corresponds to 9°. The angular score is clipped to $[0, 2]$. We set $\alpha = 1.0$, giving $w_i \in [1, 3]$. Bridge supervision is included in $\mathcal{L}_{\text{rec}}$ and encourages continuity between adjacent reconstructed segments; it adds no operation to autoregressive inference.

A.4 Decoder-only predictor

The predictor stacks $L = 5$ context-update blocks. Each block consists of three sparse-graph attention layers in order: temporal memory, map-to-segment, and agent-to-agent. After the final context-update block, a mode-to-mode attention layer is applied once to exchange information among the K modes of the same agent. Each temporal, map-to-segment, and agent-to-agent layer is a Pre-LN transformer block with $H = 8$ heads, head dimension 16, an FFN ratio of $4\times$, and dropout 0.1. Relative geometric features modulate keys and values in these three layers via a Fourier embedding [30, 31]. The mode-to-mode layer uses multi-head attention over mode states without relative geometric edge attributes. Map-to-segment and agent-to-agent edges are built by radius search around the current anchor with $r = 50$ m. Temporal edges are causal and connect each token to all earlier tokens of the same agent.

Temporal KV cache. For each temporal attention layer, the predictor caches the projected keys and values of earlier token states. Subsequent rollout steps reuse these projections, while the relative geometric terms are computed from the source and current target anchors. Specifically, the cached terms are $\mathbf{W}^m_{K,\text{temp}}\text{LN}(\mathbf{s}_u)$ and $\mathbf{W}^m_{V,\text{temp}}\text{LN}(\mathbf{s}_u)$ in the attention equations below. The predicted motion code separately determines the next autoregressive input through W_z and the mode embedding (Section 3.3); persistent code feedback and temporal caching therefore serve distinct roles.

Mode initialization. After the last historical token, the predictor branches into $K = 6$ modes by adding a learned mode embedding to the projected historical code. The output projection W_o that maps hidden states back to z is initialized to zero so that early predicted codes start near the latent-space mean, which we found important for stable rollout.

Attention layer details. The temporal, map-to-segment, and agent-to-agent layers use the same sparse relative-attention form, with different source sets. For a target token v with hidden state $\mathbf{h}_v$, let $\mathcal{N}_a(v)$ denote its sparse source neighborhood under attention type $a \in \{\text{temp}, \text{map}, \text{agent}\}$. A source element $u \in \mathcal{N}_a(v)$ has feature $\mathbf{s}_u$. For temporal attention, this is a previous token's hidden state. For map-to-segment and agent-to-agent attention, it is a map-element feature and a neighboring-agent hidden state, respectively. Each edge (u, v) carries a relative geometric encoding $\mathbf{r}^a_{u\to v}$, defined from the source and target anchor poses as in Section 3.4. This encoding is embedded by Fourier features and a relation-specific MLP:

$$\boldsymbol{\rho}^a_{u\to v} = \mathrm{MLP}_a\left(\mathrm{Fourier}(\mathbf{r}^a_{u\to v})\right). \tag{12}$$

We use $\mathrm{LN}(\cdot)$ to denote layer normalization. For attention type a and head m, $\mathbf{W}^m_{Q,a}$, $\mathbf{W}^m_{K,a}$, and $\mathbf{W}^m_{V,a}$ are learnable query, key, and value projection matrices, respectively. The matrices $\mathbf{U}^m_{K,a}$ and $\mathbf{U}^m_{V,a}$ project the relative edge embedding into the key and value spaces. Thus, the query is computed from the target token, while the key and value are computed from the source feature and modulated by the relative geometry:

$$\begin{aligned}
\mathbf{q}^{m,a}_v &= \mathbf{W}^m_{Q,a}\,\mathrm{LN}(\mathbf{h}_v),\\
\mathbf{k}^{m,a}_{u\to v} &= \mathbf{W}^m_{K,a}\,\mathrm{LN}(\mathbf{s}_u) + \mathbf{U}^m_{K,a}\boldsymbol{\rho}^a_{u\to v},\\
\mathbf{v}^{m,a}_{u\to v} &= \mathbf{W}^m_{V,a}\,\mathrm{LN}(\mathbf{s}_u) + \mathbf{U}^m_{V,a}\boldsymbol{\rho}^a_{u\to v}.
\end{aligned} \tag{13}$$

Here, $\mathbf{q}^{m,a}_v$ is the query of target v, while $\mathbf{k}^{m,a}_{u\to v}$ and $\mathbf{v}^{m,a}_{u\to v}$ are the edge-specific key and value passed from source u to target v.

The attention weights are normalized over the sparse source neighborhood of v, and the head output is the weighted sum of the modulated values:

$$\begin{aligned}
\alpha^{m,a}_{u\to v} &= \mathrm{softmax}_{u\in\mathcal{N}_a(v)}\left(\frac{(\mathbf{q}^{m,a}_v)^\top \mathbf{k}^{m,a}_{u\to v}}{\sqrt{d_h}}\right),\\
\mathbf{o}^{m,a}_v &= \sum_{u\in\mathcal{N}_a(v)} \alpha^{m,a}_{u\to v}\mathbf{v}^{m,a}_{u\to v}.
\end{aligned} \tag{14}$$

Here, $\mathbf{o}^{m,a}_v$ denotes the output of attention head m for target token v under attention type a, obtained as the weighted sum of edge-specific value vectors from its sparse source neighborhood, and d_h is the dimension of each attention head.

The outputs of all H heads are concatenated, projected by the learnable output matrix $\mathbf{W}_{O,a}$, and added back to the target token through a residual connection. The result is then passed through a feed-forward network FFN_a with another residual connection:

$$\begin{aligned}
\bar{\mathbf{h}}_v &= \mathbf{h}_v + \mathbf{W}_{O,a}[\mathbf{o}^{1,a}_v; \ldots; \mathbf{o}^{H,a}_v],\\
\mathbf{h}^+_v &= \bar{\mathbf{h}}_v + \mathrm{FFN}_a(\mathrm{LN}(\bar{\mathbf{h}}_v)).
\end{aligned} \tag{15}$$

Thus, relative geometry does not form an additional token; instead, each sparse edge contributes an edge embedding that modulates the key and value projections. The final mode-to-mode layer uses the same multi-head attention structure over mode states of the same agent, but without relative geometric modulation.

A.5 Loss weights

Loss weights. Concrete weights are listed in Table 7. Future-trajectory L1 regression receives the largest weight, while latent alignment has a much smaller coefficient. Despite this small weight, removing alignment increases forecasting error (Table 6), supporting its role as auxiliary supervision for the predicted motion codes.

Table 7: Loss weight settings.

Term	Value
VAE objective λ_{vae}	0.1
KL coefficient β_{kl}	10^{-4}
Future-trajectory L1 (winner) λ_{traj}	5.0
Historical-trajectory L1 $\lambda_{\text{traj hist}}$	1.0
Heading-trajectory L1 (hist / fut)	1.0 / 1.0
Latent alignment λ_{align}	0.01
Soft classification λ_{cls}	1.0
Soft-cls temperature τ	1.0

Representation-loss weights. Table 8 complements the removal ablations in Table 6 with a stronger latent-alignment weight. The architecture, reconstruction objective, pretraining, and joint optimization are retained. Weak KL regularization improves all four forecasting metrics relative to the autoencoder tokenizer. For latent alignment, the default weight of 0.01 outperforms both no alignment and the stronger weight of 1.0. These results support weak latent supervision as an auxiliary training signal. All other loss weights remain at their default values.

Table 8: Representation-loss weight comparisons on the AV2 validation set. Each group changes one coefficient while keeping the others at their defaults. The zero-weight and default results are repeated from Table 6.

Term	Weight	minFDE$_6\downarrow$	minADE$_6\downarrow$	MR$_6\downarrow$	b-minFDE$_6\downarrow$
KL β_{kl}	0	1.29	0.75	0.16	1.95
	10^{-4} (default)	1.17	0.70	0.13	1.84
Alignment λ_{align}	0	1.23	0.72	0.15	1.89
	0.01 (default)	1.17	0.70	0.13	1.84
	1.0	1.19	0.71	0.14	1.86

A.6 Training schedule

Tokenizer pretraining. The tokenizer and de-tokenizer are first pretrained for segment reconstruction before being loaded into the full MoCAR model. This stage optimizes only the VAE objective in Eq. (5), including segment reconstruction, bridge reconstruction, and the weak KL regularizer. We use AdamW with weight decay 10^{-4}, learning rate 10^{-4}, batch size 256, and cosine annealing for 100 epochs. The KL coefficient is set to $\beta_{\text{kl}} = 10^{-4}$. After pretraining, the encoder mean $\boldsymbol{\mu}$ is used as the deterministic motion code for ground-truth segments.

Joint forecasting training. The pretrained tokenizer–de-tokenizer is then loaded into the full MoCAR model and jointly optimized with the predictor and map encoder. We use a total batch size of 16 across both GPUs. Training uses AdamW with weight decay 10^{-4}, an initial learning rate of 2×10^{-4}, and cosine annealing to 0 over 64 epochs. The tokenizer–de-tokenizer parameters are placed in a separate parameter group whose learning rate is scaled by $0.1\times$, so that the motion-code space can adapt to forecasting while preserving its reconstruction structure. The full objective combines trajectory regression, heading supervision, soft mode classification, latent alignment, and the VAE loss as described in Section 3.5.

Positional-noise augmentation. During joint training we add Gaussian noise with $\sigma = 0.2$ to the positional channels of the input step features for 20% of agents in each mini-batch. This augmentation is applied only during training and is disabled during validation and testing. Its measured benefit appears in AV2-to-AV1 zero-shot transfer, while AV2 validation metrics are unchanged at the reported precision when it is removed (Section F). We do not use ensembling or test-time augmentation.

Hardware. Training was conducted on $2\times$ Nvidia RTX Pro 6000 GPUs for approximately 40 hours per full forecasting run.

Inference timing. All methods in Table 4 are timed on a single NVIDIA RTX 4090 with batch size 1. The matched-stride and default-stride measurements use the same timing protocol. After warming

up on 10 scenarios, we measure GPU execution time over the test set. Data loading and preprocessing are excluded. The reported latency is the total measured GPU time divided by the number of test scenarios, in milliseconds per scenario.

B Additional ablations

B.1 Predictor depth sweep

Table 9 reports an ablation on the number of context-update blocks in the decoder-only predictor. Each block consists of temporal, map-to-segment, and agent-to-agent attention, with a single mode-to-mode attention layer applied after the final block. Increasing depth from three to five consistently improves all reported metrics. The gains largely saturate at five blocks, with little further improvement from a sixth. We therefore use five blocks as the default.

Table 9: Predictor depth sweep on the AV2 validation set.

Variant	minFDE$_6$↓	minADE$_6$↓	MR$_6$↓	b-minFDE$_6$↓
3 context-update blocks	1.27	0.74	0.156	1.91
4 context-update blocks	1.22	0.72	0.142	1.89
5 context-update blocks (MoCAR)	**1.17**	**0.70**	0.134	**1.84**
6 context-update blocks	**1.17**	**0.70**	**0.133**	**1.84**

B.2 Choice of coordinate anchor

Table 10 compares the choice of local reference pose while retaining local pose normalization and the motion-code formulation. Endpoint anchoring achieves lower minFDE$_6$ than start anchoring on both AV2 validation and AV1 zero-shot evaluation. It also expresses each code in the latest predicted pose used for the next scene update. A start anchor instead refers to the pose 1.5 s earlier at the default stride.

Table 10: Choice of segment anchor with local coordinate normalization. Both variants are trained on AV2 and evaluated on AV2 validation and AV1 validation without fine-tuning.

	AV2 validation				AV1 zero-shot
Anchor	minFDE$_6$↓	minADE$_6$↓	MR$_6$↓	b-minFDE$_6$↓	minFDE$_6$↓
Segment start	1.19	0.71	0.14	1.85	1.25
Segment endpoint	1.17	0.70	0.13	1.84	1.24

B.3 Autoregressive feedback across prediction horizons

Table 11 complements the full-horizon metrics in Table 6. The variants use the same predictor, VAE, losses, data, and training schedule; only the state passed to the next prediction step changes. Both decode segments to update the anchor and scene context. Re-encoding gives slightly lower error at 1.5 s, and the variants tie at 3 s. Persistent code reuse has lower error at 4.5 s and 6 s, showing the benefit of persistent code reuse over longer rollouts.

Table 11: Autoregressive feedback ablation on AV2 validation, reporting minFDE$_6$ by prediction horizon.

Feedback state	1.5 s	3.0 s	4.5 s	6.0 s
Trajectory: decode and re-encode	0.19	0.44	0.80	1.21
Persistent motion code (MoCAR)	0.20	0.44	0.78	1.17

B.4 Discrete motion-code control

As a supplementary representation control, we jointly train a VQ-VAE tokenizer–de-tokenizer with the predictor. The latent-alignment target is the codebook entry selected by the future encoder, while

Table 12: Winner-mode distribution and entropy on the AV2 validation set.

	$k=1$	**2**	**3**	**4**	**5**	**6**	**Entropy** H
MoCAR	0.177	0.134	0.119	0.172	0.200	0.199	1.774

Table 13: VAE reconstruction error on AV2 validation segments, grouped by motion type. Stationary segments are grouped by mean speed, while non-stationary segments are grouped by segment-level heading change. Position error is measured by the mean L2 distance over segment points, and heading error is measured by the mean absolute wrapped angle error. Turn categories are defined over local $P = 16$ segments, about 1.5s on AV2; thus $|\Delta\theta| \geq 22.5^\circ$ already corresponds to a sharp local turn.

Motion type	**#Segments**	**Pos. err. (m)**$\downarrow$	**Head. err. (rad)**$\downarrow$
Stationary ($v < 0.1\,\mathrm{m/s}$)	397488	0.02	0.00
Straight ($0^\circ \leq \lvert\Delta\theta\rvert < 5^\circ$)	238393	0.08	0.01
Mild turn ($5^\circ \leq \lvert\Delta\theta\rvert < 22.5^\circ$)	33713	0.10	0.03
Sharp turn ($22.5^\circ \leq \lvert\Delta\theta\rvert < 45^\circ$)	11514	0.12	0.06
Very sharp turn (turn-heavy) ($\lvert\Delta\theta\rvert \geq 45^\circ$)	860	0.16	0.19

the predictor remains free to select another entry. We use 16,384 24-D codes with commitment weight 0.25 and codebook decay 0.99. Initialization uses 20 k-means iterations (`vq_init_ratio=128`); code maintenance uses `dead_threshold=1.0` and `reset_interval=200`.

This configuration obtains 1.35 minFDE$_6$, 0.75 minADE$_6$, 0.18 MR$_6$, and 2.02 b-minFDE$_6$ on AV2 validation, with 99.8% code utilization. The continuous VAE achieves lower forecasting error, supporting its use as the motion-code representation for MoCAR.

C Mode utilization

To examine how frequently each mode is selected as the best match to the ground truth, Table 12 reports the empirical winner-mode distribution and its entropy $H = -\sum_k p_k \log p_k$ over the AV2 validation set. A uniform distribution gives $H = \ln 6 \approx 1.79$; assigning every sample's winner to the same mode gives $H = 0$. The observed entropy of 1.774 is close to the maximum. All six modes are selected across the validation set, and no single mode dominates the winner distribution.

D Motion-code space analysis

We evaluate the reconstruction quality of the learned VAE motion codes on 681,968 AV2 validation segments across different motion types. The categories in Table 13 are defined at the segment level: stationary segments are grouped by mean speed, while non-stationary segments are grouped by heading change over a local $P = 16$ segment, corresponding to 1.5 s on AV2. These local categories differ from the turn-heavy forecasting subset, which uses heading change over the full 6 s future.

As shown in Table 13, stationary and straight segments can be reconstructed with very small errors, while the reconstruction error gradually increases as the local turning angle grows. This trend is expected since turning segments contain stronger nonlinear motion and larger heading variation. Nevertheless, mean position error remains low even for sharp and very sharp local turns, at or below 0.16 m in every category. These results suggest that the continuous VAE latent space can represent diverse motion primitives from lane-following motion to challenging turning behavior. Heading error reaches 0.19 rad for very sharp turns. The segment counts also highlight the long-tailed nature of local motion: only 860 segments turn by at least 45° within 1.5 s.

E Additional turn-heavy and refinement analysis

Table 14 further analyzes the role of method-specific refinement stages on the same AV2 validation split used in Table 3. All methods are evaluated on the same 24,988 candidates, with 3,007 turn-heavy samples and 21,981 lower-turning samples. For baselines with refinement stages, we report both the pre-refinement output and the full refined output. Refinement generally improves QCNet and DONUT, confirming that post-hoc trajectory correction is beneficial for these architectures.

MoCAR does not use any refinement stage: it directly autoregresses motion codes and decodes trajectories from the latent rollout. On the turn-heavy subset, it achieves lower minFDE_6, minADE_6, and MR_6 than both the pre-refinement and fully refined outputs of QCNet and DONUT. These results demonstrate that MoCAR achieves strong turn-heavy forecasting accuracy with a single-stage coordinate-aware latent rollout.

Table 14: Effect of method-specific refinement stages on the AV2 validation set stratified by future turning angle. All methods are evaluated on the same 24,988 candidates and the same ground-truth turn split as Table 3, with 3,007 turn-heavy and 21,981 lower-turning samples. Best values are in bold, and second-best values are underlined; ties are marked consistently.

Method	Stage	Subset	$\text{minFDE}_6\downarrow$	$\text{minADE}_6\downarrow$	$\text{MR}_6\downarrow$	b-$\text{minFDE}_6\downarrow$
QCNet [4]	w/o refinement	Turn-heavy ($\geq 45°$)	2.40	1.17	0.38	2.99
		Lower-turning ($< 45°$)	1.13	**0.65**	0.14	1.73
		All	1.28	<u>0.72</u>	0.17	1.88
	w/ refinement	Turn-heavy ($\geq 45°$)	2.37	1.17	0.36	2.98
		Lower-turning ($< 45°$)	1.10	<u>0.66</u>	<u>0.13</u>	1.72
		All	1.25	<u>0.72</u>	0.16	1.87
DONUT [1]	w/o refinement	Turn-heavy ($\geq 45°$)	2.06	1.11	0.32	2.68
		Lower-turning ($< 45°$)	<u>1.08</u>	0.68	**0.12**	<u>1.71</u>
		All	1.20	0.73	0.15	<u>1.82</u>
	w/ refinement	Turn-heavy ($\geq 45°$)	<u>2.01</u>	<u>1.10</u>	<u>0.31</u>	**2.64**
		Lower-turning ($< 45°$)	**1.07**	0.67	**0.12**	**1.70**
		All	<u>1.18</u>	0.73	<u>0.14</u>	**1.81**
MoCAR (Ours)	refinement-free	Turn-heavy ($\geq 45°$)	**1.99**	**1.07**	**0.28**	<u>2.67</u>
		Lower-turning ($< 45°$)	**1.07**	**0.65**	**0.12**	1.72
		All	**1.17**	**0.70**	**0.13**	1.84

F Argoverse 1: preprocessing and full evaluation

Zero-shot transfer protocol. Table 15 summarizes the changes encountered by an AV2-trained model when evaluated on AV1. The observation and prediction horizons, available motion states, and agent-label conventions differ, while the sampling rate is shared. This keeps the physical duration of a motion code fixed, allowing the unchanged AV2 checkpoint to be evaluated without temporal adaptation. For MoCAR, the shorter observed history yields fewer valid historical codes. The 3 s future is generated in two prediction steps at the default stride. All methods use the same preprocessing and masking protocol.

Table 15: AV2 training and AV1 zero-shot evaluation settings. A segment contains 16 points and spans 1.5 s at the shared sampling rate.

Property	AV2 training	AV1 zero-shot evaluation
Observation / prediction	5 s / 6 s	2 s / 3 s
Available motion states	Position, velocity, heading	Position; velocity and heading derived
Agent labels	Native AV2 categories	Ego and AGENT mapped to vehicle; others to other/unknown
Model weights	Trained on AV2	Unchanged AV2 checkpoint; no fine-tuning
Sampling / code duration	10 Hz / 1.5 s	10 Hz / 1.5 s

AV1 preprocessing. AV1 only provides agent positions; velocity and heading must be derived from position sequences. Direct finite differences on raw AV1 positions produce noisy heading estimates that perturb the endpoint-anchor frames MoCAR relies on. To reduce this noise, we apply local polynomial smoothing (Savitzky–Golay style) to each agent's observed position history,

using a window of 13 and polynomial order 2. Each smoothed point is constrained to stay within $0.25\,\mathrm{m}$ of its raw position. Heading and velocity are then recomputed by finite differences on the smoothed positions. The same preprocessing is applied to all methods, including the baseline zero-shot evaluations in Table 2. All AV1 metrics are computed against the original raw future positions. Smoothing is used only to derive velocity and heading inputs. For AV1 zero-shot evaluation, the autonomous-vehicle (ego) category and AGENT category are mapped to the vehicle embedding. Other observed agents are mapped to the other/unknown embedding.

Effect of positional-noise augmentation. We train a control without the positional-noise augmentation described in Section A.6, keeping all other settings unchanged. AV2 validation metrics remain unchanged at the reported precision. On AV1 zero-shot evaluation, augmentation reduces minFDE_6 from 1.27 to 1.24 and b-minFDE_6 from 1.97 to 1.93 (Table 16). Both variants use the same AV1 smoothing and state-derivation procedure. Position perturbations change displacement magnitudes and directions, affecting the derived velocity and heading estimates. The transfer gain is consistent with improved robustness to these motion inputs.

Table 16: Effect of training-time positional-noise augmentation on AV2-to-AV1 zero-shot evaluation. Both models use the same AV1 preprocessing and are evaluated without noise augmentation.

Noise augmentation	**minFDE$_1$↓**	**minADE$_1$↓**	**minFDE$_6$↓**	**minADE$_6$↓**	**MR$_6$↓**	**b-minFDE$_6$↓**
Without	5.21	2.40	1.27	0.73	0.11	1.97
With (default)	5.14	2.37	1.24	0.72	0.11	1.93

Full AV1 evaluation. Beyond the zero-shot comparison in the main text, we evaluate two additional MoCAR configurations on AV1: short fine-tuning of the AV2-trained model for 10 epochs, and full training on AV1 from scratch. Table 17 reports the complete results alongside published AV1-trained baselines. Fine-tuning on all agents reaches 1.03 minFDE_6, fine-tuning on focal and AV agents reaches 0.92, and training from scratch reaches 0.91, competitive with AV1-specialized methods and within 0.04 minFDE of the best listed result. Together with the zero-shot results, this shows that the AV2-trained MoCAR model is usable directly, substantially improves with brief fine-tuning, and remains competitive when trained on AV1 from scratch.

Table 17: Full Argoverse 1 validation evaluation. Top block: representative methods originally trained on AV1 (numbers from their original papers). Middle block: zero-shot transfer of AV2-trained models, repeated from Table 2 for completeness. Bottom block: MoCAR under fine-tuning and full AV1 training.

Method	**minFDE$_6$↓**	**minADE$_6$↓**	**MR$_6$↓**
TNT [38]	1.29	0.73	0.09
LaneRCNN [39]	1.19	0.77	0.08
mmTransformer [40]	1.15	0.71	0.11
LaneGCN [41]	1.08	0.71	–
DenseTNT [32]	1.05	0.73	0.10
SSL-Lanes [42]	1.01	0.70	0.09
FRM [33]	0.99	0.68	–
HiVT [5]	0.96	0.66	0.09
ADAPT [43]	0.95	0.67	0.08
LAformer [44]	0.92	0.64	–
DeMo [11]	0.90	**0.59**	0.07
HPNet [45]	**0.87**	0.64	**0.07**
Zero-shot transfer of AV2-trained models:			
DeMo (zero-shot)	2.03	1.08	0.25
DONUT (zero-shot)	1.58	0.97	0.19
QCNet (zero-shot)	1.55	0.95	0.19
MoCAR (zero-shot)	1.24	0.72	0.11
MoCAR: fine-tuning and AV1 training:			
MoCAR fine-tuned, all agents (10 ep.)	1.03	0.64	0.10
MoCAR fine-tuned, focal+AV (10 ep.)	0.92	0.62	0.08
MoCAR trained on AV1 (from scratch)	0.91	0.62	0.08

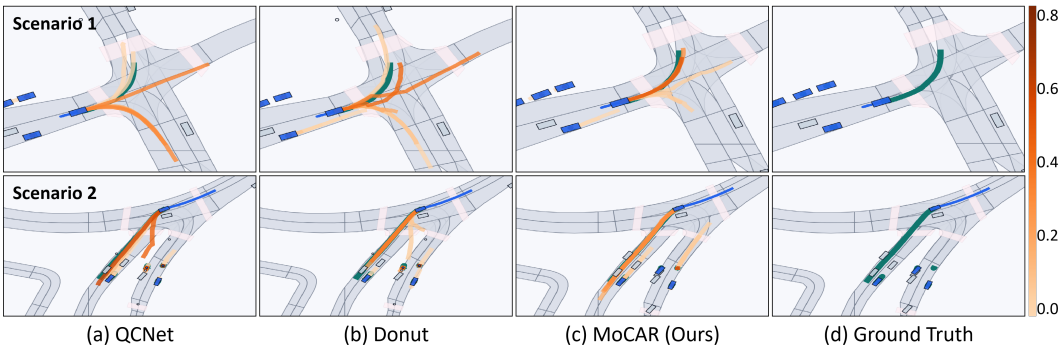


Figure 5: **Qualitative comparison on challenging AV2 scenarios.** From left to right, we show predictions from QCNet, DONUT, MoCAR, and the ground truth. Blue denotes the observed history, orange denotes predicted future trajectories, and green denotes the ground-truth future. While all methods are generally map-aligned, MoCAR produces more coherent and temporally consistent predictions in complex intersections and interaction-heavy scenes.

G Qualitative results

Comparison with other state-of-the-art methods. Figure 5 compares QCNet, DONUT, and MoCAR on challenging AV2 scenarios. All methods remain broadly map-aligned in regular cases. The differences emerge in large intersections and interaction-heavy scenes, where forecasting requires resolving maneuver ambiguity while preserving temporal consistency. In these scenarios, QCNet and DONUT become more dispersed or less coherent after the turn, whereas MoCAR follows the feasible path more consistently. This pattern is consistent with latent rollout preserving cross-segment consistency without trajectory-space re-tokenization.

Additional success cases. Figure 6 shows additional qualitative examples on the AV2 validation set. Across diverse scenarios, the model produces predictions that remain well aligned with the local road geometry and closely match the ground-truth future. In addition, the supplementary material provides dynamic **video visualizations** that show the continuous forecasting results over time, further demonstrating the temporal consistency and stability of the proposed model.

Failure case analysis. We manually inspected representative cases from the AV2 validation set where MoCAR produces large displacement errors. As shown in Figure 7, the illustrated failures involve a mismatch between map-consistent prediction and the realized driving behavior. MoCAR tends to favor trajectories aligned with local lane geometry and scene topology. This preference generally helps, but can lead to errors when an agent makes a late directional commitment or executes a less common maneuver. The model can also miss a continuation weakly supported by the local map structure. In these cases, it predicts a topologically plausible continuation, while the ground-truth future reflects a rarer or less predictable behavioral choice. These examples highlight the challenge of handling behavioral deviations and rare decisions that are difficult to infer from history and map context alone.

H Broader Impacts

MoCAR is evaluated as an offline trajectory-forecasting component on public autonomous-driving benchmarks. More accurate and efficient forecasting may benefit downstream planning research, especially in complex maneuvers and cross-dataset transfer settings. Beyond road-agent forecasting, the decoder-only motion-code formulation may also inform related 2D and 3D motion prediction problems where agents move under evolving local coordinate frames, such as maritime navigation, UAV trajectory prediction, and robotic motion forecasting. However, incorrect forecasts could be unsafe if used directly in deployed systems. Real-world deployment would require extensive safety validation, uncertainty handling, monitoring under distribution shift, and integration with fail-safe planning and control modules.

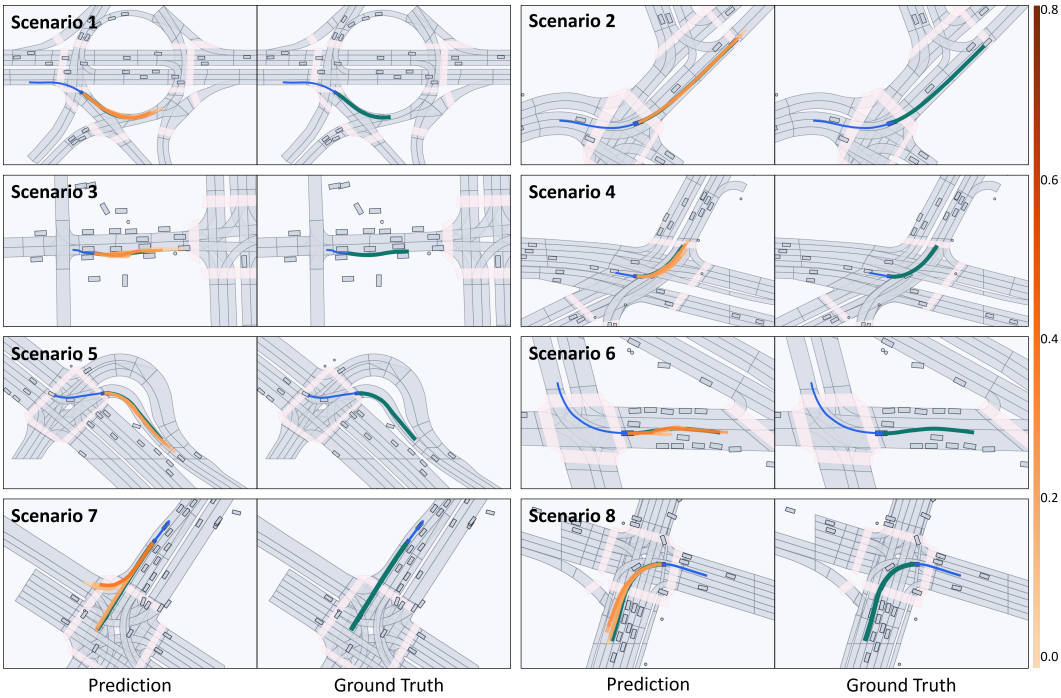


Figure 6: **Additional qualitative success cases on the AV2 validation set.** Eight representative examples covering roundabouts, straight driving, curved-road following, and turning maneuvers at intersections. Blue shows the observed history, green the ground-truth future, and orange the predicted futures, with darker shades indicating higher predicted probability. The predictions remain consistent with the local map structure and closely follow the ground-truth motion.

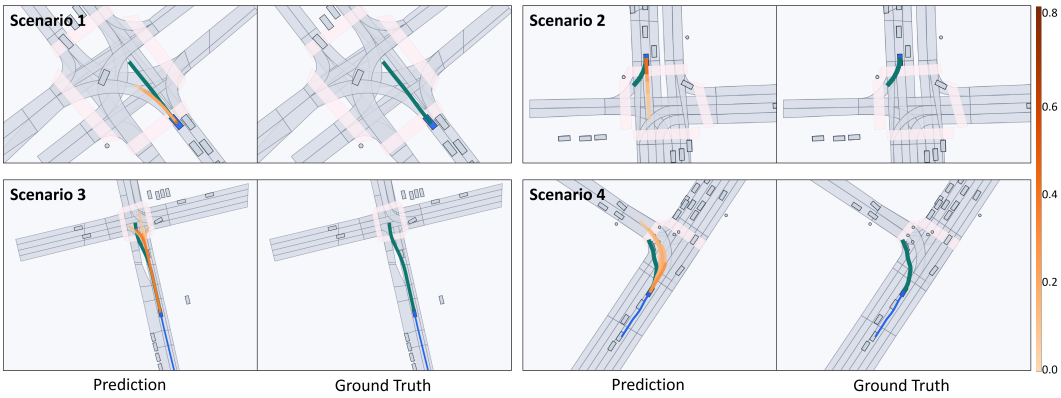


Figure 7: **Representative failure cases on the AV2 validation set.** Blue shows the observed history, green the ground-truth future, and orange the predicted futures, with darker shades indicating higher predicted probability. Most errors occur when the model follows the most map-consistent continuation, but the realized behavior reflects a later or less common maneuver.

I Asset usage and license compliance

Datasets and benchmarks. We use Argoverse 2 [15] and Argoverse 1 [16] following their official terms of use. Both datasets are public autonomous-driving forecasting benchmarks released under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International license (CC BY-NC-SA 4.0). The associated Argoverse code and APIs are released under the MIT license.

External code and model assets. For baseline evaluation, we use publicly released implementations and pretrained checkpoints of QCNet [4], DONUT [1], and DeMo [11]. The official QCNet repository is released under the Apache-2.0 license and provides pretrained checkpoints for Argoverse 2 evaluation. The official DONUT repository is also released under the Apache-2.0 license, builds upon QCNet, and provides a pretrained checkpoint for evaluation. The official DeMo repository provides publicly available checkpoints; however, we did not find an explicit repository-level license for its code or checkpoints. We therefore use DeMo only for academic benchmarking and comparison, with proper citation, and do not redistribute its code, checkpoints, or derived model assets.

Compliance. All external assets are used only for academic research, benchmarking, and reproducibility. The supplementary material does not include third-party datasets, external codebases, pretrained checkpoints, or non-redistributable assets. The MoCAR implementation and AV2 preprocessing, training, validation, and visualization scripts are publicly available at `https://github.com/XuYiMing83/MoCAR`. The repository includes installation and usage instructions and an MIT license. Model checkpoints will be released upon publication. The release will preserve the required attribution and will not redistribute external assets beyond their original license terms.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state three contributions: a coordinate-aware continuous motion-code space, a decoder-only autoregressive formulation without proposal-and-refinement pipelines, and competitive performance on Argoverse benchmarks. These claims are directly supported by Sections 3 and 4, and the ablations in Section 4.4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes. The paper includes an explicit "Limitations" paragraph and further analyzes representative failure cases; see Section 5 and Appendix G.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.

- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results requiring theorem statements and proofs; its main contributions are methodological and empirical.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Yes. The paper specifies the model formulation, datasets, evaluation setup, ablations, preprocessing, implementation details, and training schedule needed to reproduce the main results; see Sections 3– 4 and Appendix A–A.6.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed

instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: MoCAR source code is publicly available at `https://github.com/XuYiMing83/MoCAR`, including preprocessing, training, validation, and visualization scripts. The repository README provides installation and usage instructions. Experiments use public Argoverse 1 and Argoverse 2 data; the paper and appendix document the model, training settings, and evaluation protocols.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

Justification: Yes. The paper specifies the datasets, prediction horizons, architecture dimensions, optimizer, schedule, segment construction, and other training details in the main paper and appendix; see Section 4 and Appendix A.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: The tables report single-run point estimates rather than multi-seed summary statistics. This version does not report error bars, confidence intervals, or statistical significance tests. Evaluation settings and controlled ablations are described in Section 4 and Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Training compute is reported in Appendix A.6 ($2\times$ Nvidia RTX Pro 6000, $\sim$40 hours per run). Inference latency and parameter count for our model and baselines on a single Nvidia RTX 4090 are reported in Section 4.3 (Table 4).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and confirm that our research conforms to it. The work uses only publicly available driving datasets (Argoverse 1 and Argoverse 2) and does not involve human subjects.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix H discusses the potential positive impact of more accurate and efficient trajectory forecasting for autonomous-driving research, as well as possible negative impacts from incorrect forecasts if used directly in deployed systems without uncertainty handling, safety validation, monitoring under distribution shift, and fail-safe planning and control.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release pre-trained generative models, scraped datasets, or other assets with high risk for misuse. Trained checkpoints, if released, would be specific to motion forecasting on Argoverse.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Argoverse 1 [16] and Argoverse 2 [15] are properly cited and used under their public dataset license, CC BY-NC-SA 4.0; the associated Argoverse APIs are released under the MIT license. For baseline evaluation, we use publicly released implementations and checkpoints of QCNet [4], DONUT [1], and DeMo [11]. QCNet and DONUT are released under the Apache-2.0 license. For DeMo, we did not find an explicit repository-level license, so we use it only for academic benchmarking and do not redistribute its code, checkpoints, or derived model assets. Full details are provided in Appendix I.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not introduce or release new assets at submission time, such as new datasets, benchmark suites, source code packages, model checkpoints, or pretrained models. The experiments use existing public datasets, Argoverse 1 and Argoverse 2.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing experiments or research with human subjects, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not part of the paper's core method, experiments, or scientific claims.

    Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.