Supplementary Material for Streaming Real-Time Trajectory Prediction Using Endpoint-Aware Modeling

Alexander Prutsch David Schinagl Horst Possegger
Institute of Visual Computing, Graz University of Technology
{alexander.prutsch, david.schinagl, possegger}@tugraz.at

In this supplementary, we first present a comparison of our streaming processing to traditional snapshot-based methods in Section 1. To support reproducibility, in the following, we provide all implementation details of our model (Section 2) and how to apply it to the multi-agent setting (Section 3). We further detail the evaluations (Section 4), present additional results, such as a robustness study on recovering early prediction errors, and provide visualizations of our approach (Section 5). Finally, we provide an overview of the code framework (Section 6) which is also included as supplemental material.

1. Streaming vs. Snapshot-based Processing

Figure 1 compares the standard snapshot-based trajectory prediction paradigm to the streaming-based processing employed by us and related work [10, 12]. Both paradigms use the same scenario splits to separate training and validation data, ensuring a fair comparison. To enable streaming processing without longer context data, the historical input fed into the model is reduced. This allows multiple predictions to be executed within the same set of training data.

The streaming-based processing closely reflects real-world deployment scenarios, where trajectory prediction models operate in a continuous setting. It enables an information flow between successive prediction steps, unlike the snapshot-based paradigm, where each prediction is performed independently. This allows models to leverage temporal information, leading to more robust and consistent forecasting.

For Argoverse 2 [11] we set the model input history to $h = 3\,\mathrm{s}$, which corresponds to $T_h = 30$ input samples and we execute three predictions with a $1\,\mathrm{s}$ time gap in between at $t \in \{3, 4, 5\}\,\mathrm{s}$. Following the benchmark settings, we predict a future of $6\,\mathrm{s}$ which corresponds to $T_f = 60$ output time steps. To fully leverage the available training data during streaming processing, we also predict for $T_a = 20$ additional steps into the future. This leads to a total model output of $8\,\mathrm{s}$ seconds. For all evaluations and also for selecting our endpoint-centric features only the future up to T_f

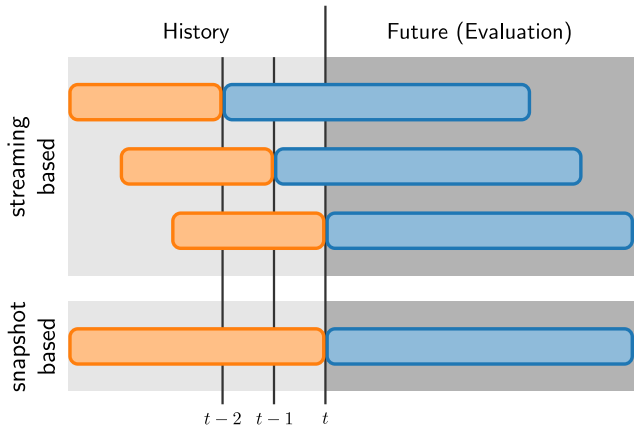

Figure 1. Comparison between snapshot-based and streaming-based trajectory prediction paradigms. Both approaches operate on the same benchmark input data, ensuring a fair comparison without additional data. In the streaming paradigm, past observations are processed using a sliding window, closely resembling practical deployment conditions. To model the challenges of real-world operation information relay mechanisms are established in streaming processing. This enables more consistent and temporally coherent predictions compared to snapshot-based processing, which handles each frame independently. In this example, predictions from the third pass of the streaming model can be directly evaluated against those from the snapshot-based approach.

steps is relevant. During the first and second model pass, the additional future steps T_a allow us to compute the regression losses on a longer future horizon, effectively teaching the model to predict long-term future. For our ablation on Argoverse 1 [1] we use a historic input of $1\,\mathrm{s}$ ($T_h = 10$) and execute three passes at $t \in \{1, 1.5, 2\}\,\mathrm{s}$.

2. Implementation Details

We set the feature dimension of our model to $D = 128$. Our categorical type embeddings distinguish between four agent types (vehicles, pedestrians, cyclists, and others) and three lane types (standard lanes, bike lanes and bus lanes). We use the standard radius of 150 meters to collect scene

elements [2, 9, 10, 12]. To maintain a manageable number of scene tokens, we do not split the lanes into smaller segments as *e.g.* suggested by [3, 6]. We use the xy-positions, xy-velocities and a corresponding *valid* flag to encode agent data, leading to $D_a = 5$. To encode the lane data, we use the xy-positions of each lane point and a *valid* flag to mask parts of lane segments exceeding the region of interest, leading to $D_l = 3$.

2.1. Positional Embeddings

To model the global poses of all elements — agents, lanes, and target-centric coordinate systems — we represent each using its position (x, y) and rotation yaw. For agents, we use their initial position and orientation; for lanes, we use the center point and orientation of the centerline; and for target-centric coordinate systems, we use the trajectory endpoint and compute the orientation based on the curvature at the trajectory's end.

2.2. Target-Centric Coordinate Systems

Figure 2 illustrates the setup of our target-centric coordinate systems. We use the endpoint from trajectory prediction F^{t-1} at the previous time step to create a local reference frame centered and oriented around the target region. All scene features within the target context radius are then encoded w.r.t. the target coordinate system. To capture global spatial relationships, we additionally incorporate positional embeddings from the agent-centric origin to the target-centric frame (blue dashed line).

2.3. Multi-Head Attention Blocks

For our multi-head attention blocks we use the following configuration:

- Number of attention heads: 8
- Dropout rate: 0.2
- Feedforward expansion factor: $4 \cdot D$
- Activation function: GELU [4]
- Norm before attention operation

2.4. Encoder

We utilize four encoder blocks in the agent encoder f_A and four encoding blocks in our scene context encoder f_S. For our target context encoder f_T we utilize two encoder blocks. Table 1 provides an ablation study for the depth of our novel target context encoder. Since the number of tokens N_c' in our target-centric context is smaller than the number of tokens N_c in the agent-centric context, a shallower encoder can be used. To compute the position embeddings for a given pose (x, y, θ) we first transform it to $(x, y, \sin\theta, \cos\theta)$. Then, we apply a two-layer multilayer perceptron (MLP):

- Layer 1: $4 \times D$
- GELU [4] activation function
- Layer 2: $D \times D$

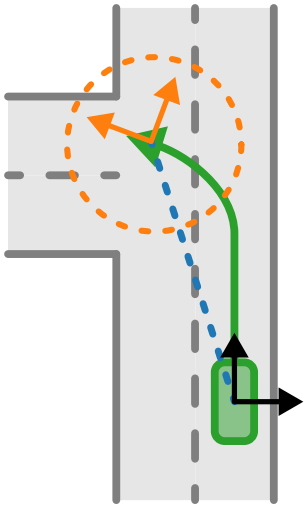

Figure 2. Modeling of target-centric coordinate system (orange) using the prediction (green) from the previous time frame. We also show the target context radius (orange) and the positional embedding (blue) used to model the relationship to the agent-centric origin (black).

2.5. Decoder

Our dual-context decoder consists of three stages ($d = 3$), resulting in a total of six cross-attention blocks. If an agent has no previous prediction, the cross-attention step to the target-centric context is skipped. The MLP to process the decoded queries Q' and output future trajectories F has two layers:

- Layer 1: $D \times 2 \cdot D$
- ReLU activation function
- Layer 2: $2 \cdot D \times 2 \cdot (T_f + T_a)$

Similarly, the MLP for outputting probability scores P:

- Layer 1: $D \times 2 \cdot D$
- ReLU activation function
- Layer 2: $2 \cdot D \times 1$

As described above (Section 1), our decoder predicts T_a additional steps in addition to the T_f steps which are used as actual trajectories in the evaluation.

2.6. Optimization

We employ a common winner-takes-all strategy for training our model [2], where only the best fitting predicted trajectory (*i.e.* with the smallest average displacement error) is used for optimization. We utilize a smooth L1 loss (Huber loss [5]) as regression loss L_{reg} to ensure that the hypothesis fits to the ground truth. Additionally, we use a standard cross-entropy loss L_{cls} to assign the highest confidence score to the best-fitting trajectory. To further enhance learning, we employ an auxiliary loss L_{aux} [2], where a single trajectory is predicted for each non-focal agent present in the scene, and a smooth

Dataset	# Blocks	mADE_6	mFDE_6	b-mFDE_6
AV2 Val	1	**0.66**	1.26	1.86
	2	**0.66**	**1.25**	**1.85**
	3	**0.66**	1.26	**1.85**

Table 1. Ablation study for different number of attention blocks in our target encoder f_T.

L1 loss is applied. To predict the auxiliary future, we utilize a linear layer $D \times 2 \cdot T_f$ for each token in our scene context S that corresponds to an agent excluding the focal agent. The final loss is given as $L = L_{\text{reg}} + L_{\text{cls}} + L_{\text{aux}}$.

We train our model for 80 epochs, with the first 13 epochs serving as warm-up phase, during which the we linearly increase the learning rate from 1e-5 to 1e-2. Afterward, we decrease the learning rate to 1e-5 using a single cosine annealing schedule. Training is executed using a batch size of 32 on a single NVIDIA Quadro RTX 8000. We utilize AdamW [7] as optimizer, employ norm-based gradient clipping with maximum value set to 5 and execute weight decay with 1e-2. We only train on the Argoverse 2 training set without any pre-training or data augmentations.

3. Multi-Agent Extension

3.1. Implementation Details

In the AV2 multi-agent settings the goal is to predict trajectories for all scored agents. Our global consistency module employs two transformer blocks for self-attention across all modes of an agent, followed by two transformer blocks for self-attention across agents per mode (world). In the first stage, we do self-attention on our decoded $Q' \in \mathbb{R}^{N_{a,s} \times K \times D}$ across all modes K for each agent, where $N_{a,s}$ is the number of scored agents. Next, we permute the queries to $\mathbb{R}^{K \times N_{a,s} \times D}$ and perform self-attention across all agents $N_{a,s}$ per mode. To better model different agent behavior, we add categorical type embeddings to Q', distinguishing between focal-agents, *scored*-agents which are driving and *scored*-agents that are likely to be parked (based on the marginal predictions). We generate the world predictions $F_w \in \mathbb{R}^{K \times N_a, s \times T_f \times 2}$ using a two-layer MLP:

- Layer 1: $D \times 2 \cdot D$
- ReLU activation function
- Layer 2: $2 \cdot D \times 2 \cdot T_f$

Similarly, the MLP for outputting the associated confidence scores $P_w \in \mathbb{R}^K$:

- Layer 1: $D \times 2 \cdot D$
- ReLU activation function
- Layer 2: $2 \cdot D \times 1$

We use the same streaming-processing setup as in the single-agent setting.

3.2. Optimization

To adapt our approach to the multi-agent setting in Argoverse 2 (AV2) [11], we initialize the model weights using our single-agent model (marginal prediction model). We then jointly train the marginal prediction model and the global consistency module end-to-end on the AV2 multi-agent training set for 35 epochs, without any warm-up phase. To reduce memory consumption, we freeze the agent and lane encoders. During training, we apply a cosine annealing schedule to decay the learning rate from 1e-2 to 1e-5.

Again, we employ a winner-takes-all principle by optimizing only the trajectories of the best world (lowest average minimum displacement error). Following single-agent training, we use a regression loss L_{reg} for each agent prediction in the best world and employ a cross-entropy classification loss L_{cls} to assign the highest confidence score to the best-fitting world. To retain strong single-agent performance and guide multi-agent learning, we also include the single-agent losses L_{marginal} (see above) during training on the multi-agent setting. The final loss is given as $L = L_{\text{reg}} + L_{\text{cls}} + L_{\text{marginal}}$.

4. Evaluation Details

4.1. Metrics

We evaluate our approach using the standard AV2 benchmark metrics. Each metric is evaluated using the top k scoring trajectory hypotheses. The minimum average displacement error (minADE_k) is the mean Euclidean distance between the ground truth and the best-fitting hypotheses across all time steps. The minimum final displacement error (minFDE_k) considers only the distance at the final time step, while the Brier minimum final displacement error (brier-minFDE_k) adds a penalty term $(1-\pi)^2$ to the minFDE_k, where π is the probability score for the best-fitting trajectory. The miss rate (MR_k) evaluates whether any predicted endpoint is within a radius of 2 meters from the ground truth endpoint.

In the multi-agent setting, we evaluate all metrics for the top k scoring worlds. Each world contains one future trajectory for each *scored* agent. The average minimum average displacement error (avgMinADE_k) is computed by averaging the minADE_k across all actors within a world. Analogously, the average minimum final displacement error (avgMinFDE_k) considers the minFDE_k for all actors, and the average brier minimum final displacement error (avgBrierMinFDE_k) averages the brier-minFDE_k across all actors in a world. The actor miss rate actorMR_k denotes the rate of all scored agents which have an endpoint within 2 meters around the ground truth endpoint.

Method	minADE_1	minFDE_1	MR_6	minADE_6	minFDE_6	brier-minFDE_6
RealMotion [10]	1.65	4.10	0.16	0.67	1.30	1.94
DeMo [12]	**1.48**	**3.73**	**0.13**	**0.61**	**1.19**	1.86
SEAM (Ours)	1.60	3.96	0.15	0.66	1.25	**1.85**

Table 2. Comparison of streaming-based methods for single-agent trajectory prediction on the Argoverse 2 validation set. For all models we report results without model ensembling.

Endpoint Noise	mADE_6	mFDE_6	b-mFDE_6
$\mathcal{N}(0, 5)$	0.67	1.28	1.88
$\mathcal{U}(-5, 5)$	0.67	1.27	1.87
$\mathcal{N}(0, 3)$	0.67	1.26	1.87
$\mathcal{U}(-3, 3)$	0.67	1.26	1.86
$\mathcal{N}(0, 1)$	0.67	**1.25**	**1.85**
$\mathcal{U}(-1, 1)$	0.67	**1.25**	**1.85**
None	**0.66**	**1.25**	**1.85**

Table 3. Robustness of our endpoint-aware modeling on the AV2 [11] validation set by perturbing the prediction endpoints at $t \in \{3, 4\}$ s. These endpoints, which define the anchors for extracting our target-centric features, are modified using additive uniform noise ($\mathcal{U}$) or Gaussian noise ($\mathcal{N}$). The shown results correspond to the prediction errors at $t = 5$s.

4.2. Latency Measurements

We utilize the official code implementations to measure the latencies of QCNet[1] [13], RealMotion[2] [10], and DeMo[3] [12]. To ensure fair comparison and eliminate influences by different data loading and preprocessing implementations, we measure only the inference latency of the model forward pass.

5. Additional Results

5.1. Streaming Methods on AV2 Validation Set

Table 2 compares streaming-based trajectory prediction approaches on the Argoverse 2 validation set. Our approach again yields the best brier-minFDE_6, which considers both displacement errors and trajectory scoring. By comparing the brier-minFDE_6 to minFDE_6 values of the different models, we can assess that our approach excels in estimating the likelihood of predicted trajectories, which is important for interpreting the results in downstream tasks like ego-motion planning.

5.2. Robustness to Error Propagation

Table 3 evaluates the robustness of our endpoint-aware modeling for information streaming when previous predictions are noisy. To this end, we perturb the endpoint anchors, which are used to obtain our target-centric features, with uniform or Gaussian noise. The results show that performance only slightly deteriorates under small prediction noise. Nevertheless, our dual-context approach maintains robust performance even when past predictions are affected by errors. The target-centric features provide additional information that improves overall prediction accuracy without constraining

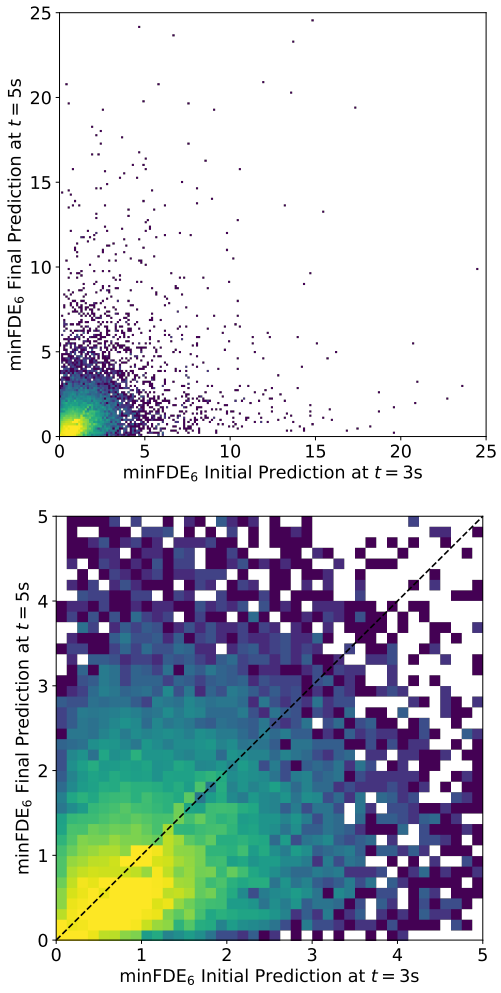

Figure 3. Density plots of prediction errors. Both plots compare the first prediction done at $t = 3$ s (x-axis) versus the final prediction done at at $t = 5$ s (y-axis). The top plot shows that even when the initial error is large, the model can recover and correct its prediction. The bottom plot provides a zoomed-in view near the origin. The higher density of points below the $x = y$ diagonal (dashed line) indicates that predictions generally improve over time.

[1] https://github.com/ZikangZhou/QCNet
[2] https://github.com/fudan-zvg/RealMotion
[3] https://github.com/fudan-zvg/DeMo

Method	Latency ($B = 1$)		Latency ($B = 32$)		Latency ($B = 64$)	
	Offline	Online	Offline	Online	Offline	Online
RealMotion [10]	51 ms	**13 ms**	255 ms	85 ms	512 ms	172 ms
DeMo [12]	73 ms	18 ms	223 ms	71 ms	433 ms	143 ms
SEAM (Ours)	**50 ms**	**13 ms**	**95 ms**	**39 ms**	**185 ms**	**65 ms**

Table 4. Latency analysis for predicting B Argoverse 2 single-agent scenarios using one NVIDIA A10 GPU. We compare streaming-based methods and report the offline and online inference latency. The online latency is relevant for practical application.

Model	Training Data	Global Consistency Module	actorMR$_6$	avgMinADE$_6$	avgMinFDE$_6$	avgBrierMinFDE$_6$
RealMotion [10]	Single-Agent Data	✗	0.727	1.413	4.060	4.716
SEAM (Ours)	Single-Agent Data	✗	0.693	1.324	3.643	4.256
SEAM (Ours)	Finetuned on Multi-Agent Data	✗	0.228	0.720	1.619	2.126
SEAM (Ours)	Finetuned on Multi-Agent Data	✓	**0.155**	**0.600**	**1.180**	**1.814**

Table 5. Ablation study on adapting single-agent methods to the AV2 multi-agent validation set. The first two rows show that, without any model adaptations or finetuning, both our approach and related work [10] perform poorly in the multi-agent setting. Finetuning on the multi-agent dataset significantly improves performance, and incorporating a global consistency module for scene-level scoring yields the best results.

predictions, allowing recovery when agent-centric features at the current timestep indicate a different future movement. As the noise magnitude increases, performance declines further; however, the model remains resilient even under substantial noise offsets.

Furthermore, we analyze the evolution of errors in Figure 3. The plots compare the minFDE$_6$ for the first prediction, made at $t = 3\,\text{s}$ into the scenario, with the final prediction at $t = 5\,\text{s}$, each evaluated over a future horizon of 6 s. The results show that our dual-context decoding approach can recover from early prediction errors: even when the initial prediction is inaccurate (high value on the x-axis), the final prediction can still be highly accurate (low value on the y-axis). There are also some cases where the initial prediction is accurate but the final prediction degrades, for example when an unexpected maneuver, *e.g.* a sudden stop, is not apparent in the first prediction window but becomes relevant in the second. Overall, the model achieves improved prediction quality over time, as reflected by the higher point density below the $x = y$ diagonal.

5.3. Latency on NVIDIA A10 GPU

We provide additional latency results using one NVIDIA A10 GPU, comparing streaming-based methods in Table 4. Also on this newer GPU architecture our approach achieves the best latency results, obtaining the lowest online and offline latency across all batch sizes.

5.4. Multi-Agent Ablation Study

We present an ablation study on extending single-agent approaches to the multi-agent setting in Table 5. The first two rows show that naively applying a single-agent model to the multi-agent benchmark yields poor performance. This is primarily because the multi-agent dataset contains motion patterns that are not well-represented in the single-agent data. Moreover, these approaches ignore global consistency across agent predictions. Without a dedicated global consistency module, we generate multiple plausible future worlds by combining the most likely trajectories of each agent into one world, the second most likely into another, and so on. Finetuning the model on multi-agent data helps capture a broader range of behaviors (row 3), and incorporating an explicit global consistency module further improves performance, achieving the best results (row 4).

Method	Fluctuation
RealMotion [10]	0.347
SEAM (Ours)	**0.341**

Table 6. We compare the trajectory fluctuation of our model to RealMotion [10]. The fluctuation metric defined by [8] gives an indication on the consistency of trajectories across multiple timesteps.

5.5. Trajectory Fluctuation

Table 6 compares the trajectory fluctuation of our approach with that of RealMotion [10]. The fluctuation metric, as defined by [8] measures the consistency of trajectories across multiple prediction frames. Our approach achieves a lower fluctuation score than RealMotion, indicating more consistent predictions.

5.6. Result Visualizations

We present additional qualitative results on scenarios from the Argoverse 2 validation set in Figure 4 and Figure 5.

5.7. Failure Cases

We present failure cases, where our approach fails to correctly predict future trajectories in Figure 6. Commonly,

failures are introduced by agent movements which cannot be anticipated at the prediction time, often also due to inadequate map data, *i.e*. missing modeling of driveways.

6. Code Implementation

For better clarity and to facilitate reproducibility, we also provide our code implementation. The accompanying *ReadMe* file outlines how to setup a working environment and execute training, validation and visualization for the single-agent task. The codebase also includes all implementations for the multi-agent setting (files marked with `ma` prefix or suffix). We will release the source code and pretrained models upon paper acceptance. Our code is based on the implementation of RealMotion[2].

References

[1] 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 *CVPR*, 2019. 1

[2] Jie Cheng, Xiaodong Mei, and Ming Liu. Forecast-MAE: Self-supervised Pre-training for Motion Forecasting with Masked Autoencoders. In *ICCV*, 2023. 2

[3] Yiqian Gan, Hao Xiao, Yizhe Zhao, Ethan Zhang, Zhe Huang, Xin Ye, and Lingting Ge. MGTR: Multi-Granular Transformer for Motion Prediction with LiDAR. In *ICRA*, 2024. 2

[4] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *arXiv*, 2016. 2

[5] Peter J Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 1964. 2

[6] Zhiqian Lan, Yuxuan Jiang, Yao Mu, Chen Chen, Shengbo Eben Li, Hang Zhao, and Keqiang Li. SEPT: Towards Efficient Scene Representation Learning for Motion Prediction. In *ICLR*, 2023. 2

[7] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019. 3

[8] Ziqi Pang, Deva Ramanan, Mengtian Li, and Yu-Xiong Wang. Streaming Motion Forecasting for Autonomous Driving. In *IROS*, 2023. 5

[9] Alexander Prutsch, Horst Bischof, and Horst Possegger. Efficient Motion Prediction: A Lightweight & Accurate Trajectory Prediction Model With Fast Training and Inference Speed. In *IROS*, 2024. 2

[10] Nan Song, Bozhou Zhang, Xiatian Zhu, and Li Zhang. Motion Forecasting in Continuous Driving. In *NeurIPS*, 2024. 1, 2, 4, 5, 7, 8, 9

[11] 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 *NeurIPS Datasets and Benchmarks*, 2021. 1, 3, 4

[12] Bozhou Zhang, Nan Song, and Li Zhang. DeMo: Decoupling Motion Forecasting into Directional Intentions and Dynamic States. In *NeurIPS*, 2024. 1, 2, 4, 5

[13] Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-Centric Trajectory Prediction. In *CVPR*, 2023. 4

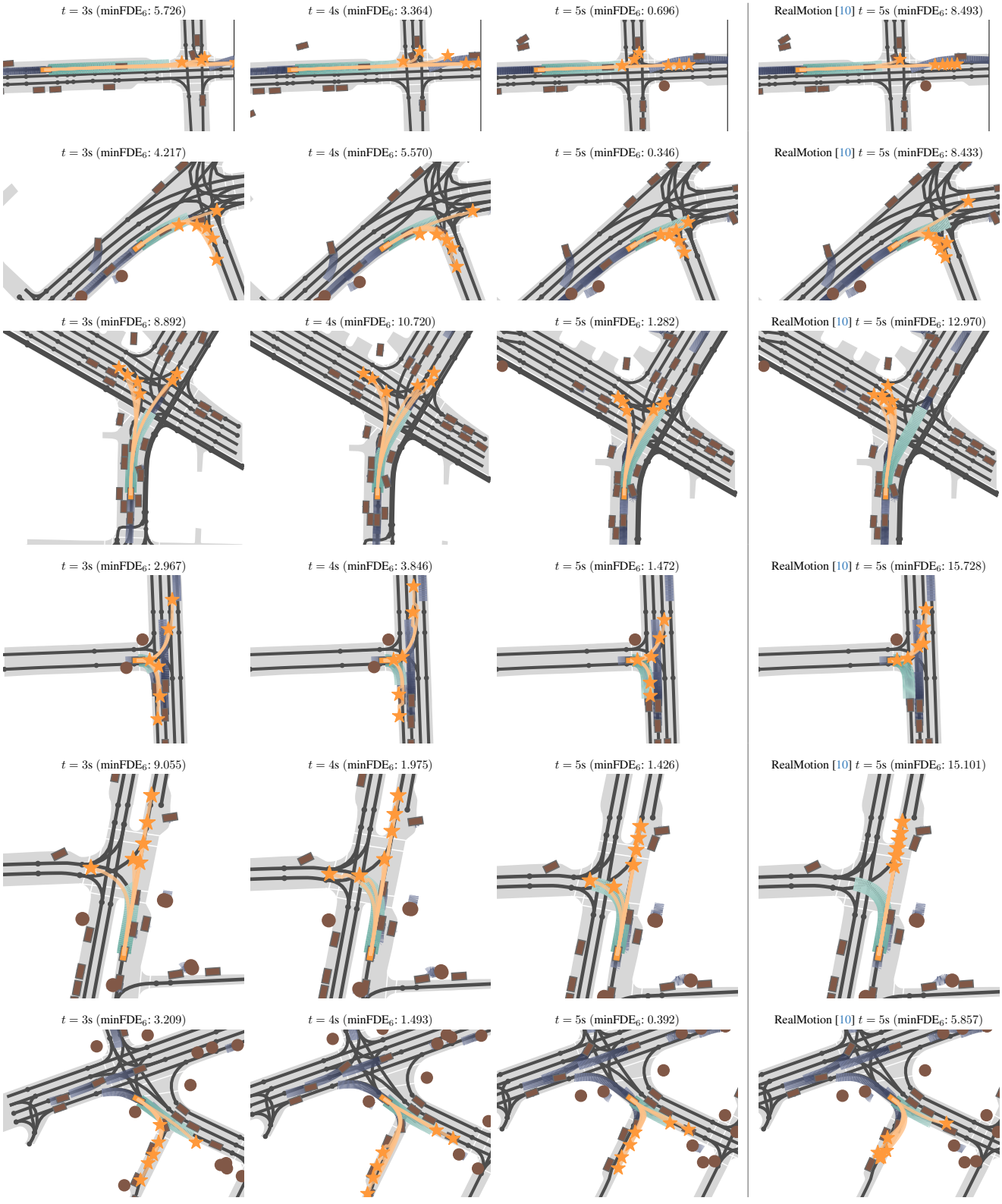

Figure 4. Qualitative results of our approach on scenarios from the Argoverse 2 validation set. We show the **predictions** of our streaming-based method at $t \in \{3, 4, 5\}$s. The visualizations also show **ground truth future**, **agent histories**, and **neighboring agents**. The right column shows the final predictions at $t = 5$s for using RealMotion [10] in the streaming setting.

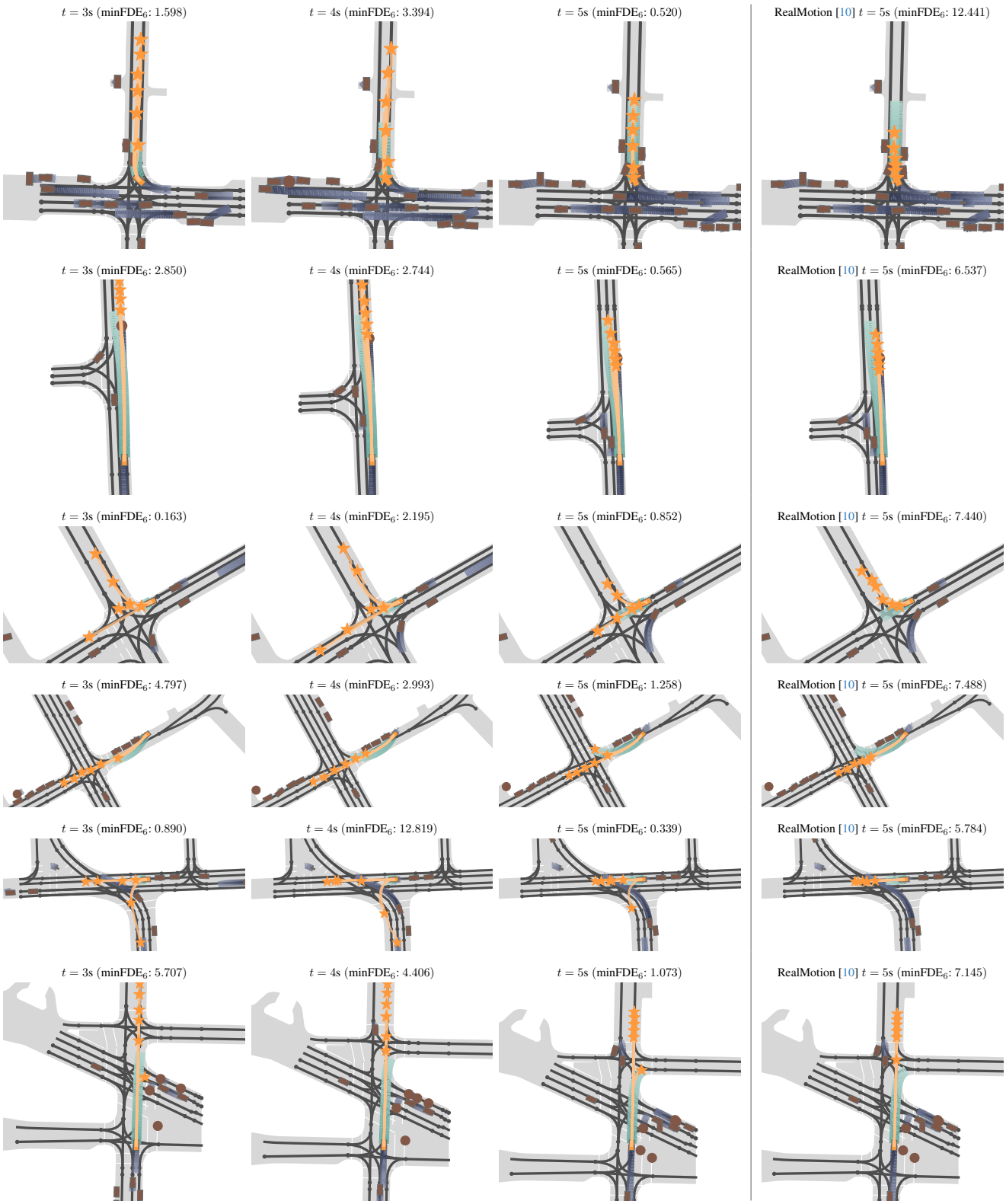

Figure 5. Qualitative results of our approach on scenarios from the Argoverse 2 validation set. We show the **predictions** of our streaming-based method at $t \in \{3, 4, 5\}$s. The visualizations also show **ground truth future**, **agent histories**, and **neighboring agents**. The right column shows the final predictions at $t = 5$s for using RealMotion [10] in the streaming setting.

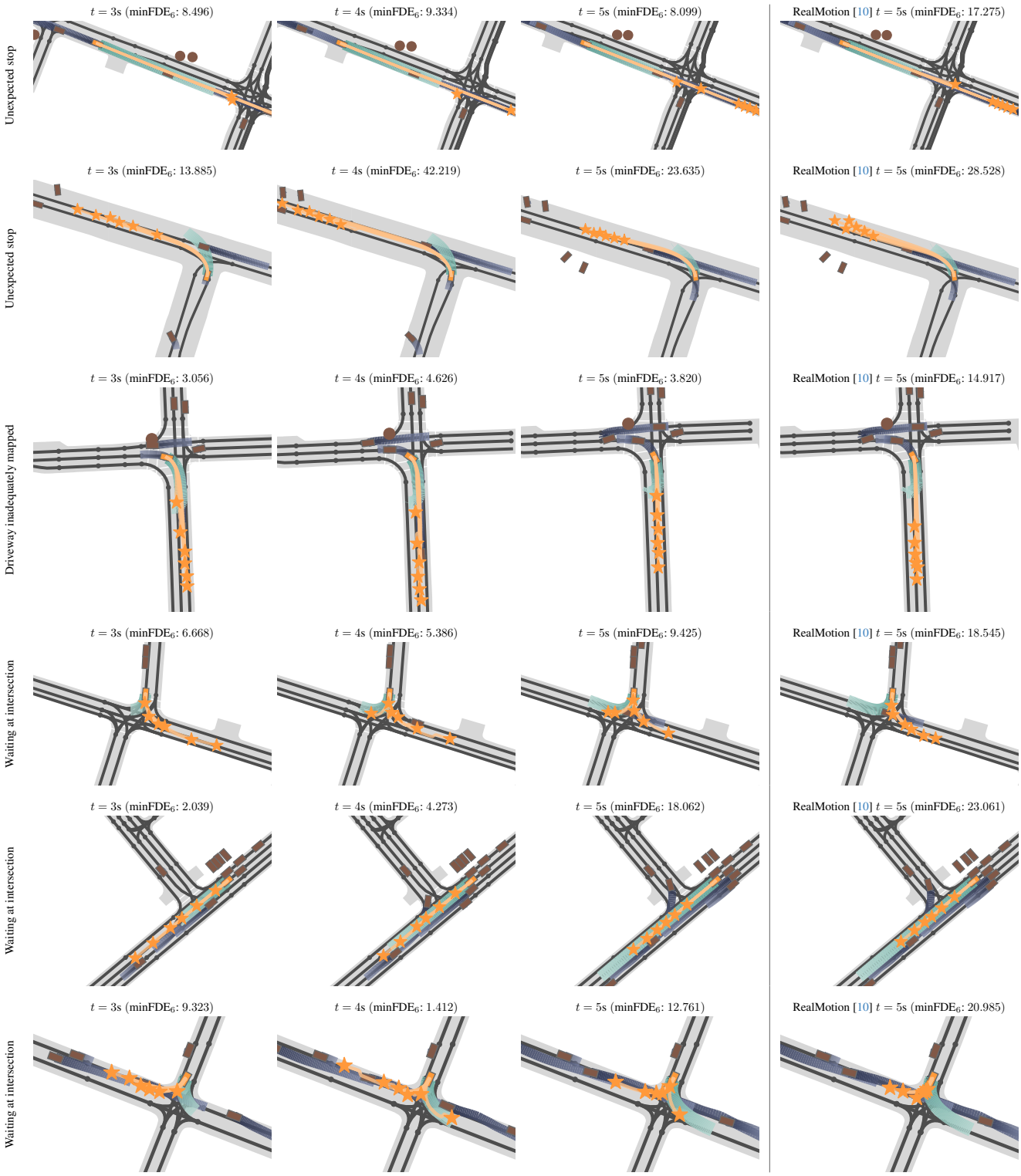

Figure 6. Failure cases of our approach on scenarios from the Argoverse 2 validation set. We show the **predictions** of our streaming-based method at $t \in \{3, 4, 5\}$s. The visualizations also show **ground truth future**, **agent histories**, and **neighboring agents**. The right column shows the final predictions at $t = 5$s for using RealMotion [10] in the streaming setting.

In the first scenario, a vehicle stops before an intersection that is fairly far away for a currently unknown reason. In the second scenario, a vehicle stops for a reason that is not detected at the moment. In the third scenario, the vehicle begins turning into a driveway that is not modeled in the lane data. The fourth, fifth, and sixth scenarios depict vehicles waiting at an intersection where their future path is unclear.