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

Abstract

Future trajectories of neighboring traffic agents have a significant influence on the path planning and decision-making of autonomous vehicles. While trajectory forecasting is a well-studied field, research mainly focuses on snapshot-based prediction, where each scenario is treated independently of its global temporal context. However, real-world autonomous driving systems need to operate in a continuous setting, requiring real-time processing of data streams with low latency and consistent predictions over successive timesteps. We leverage this continuous setting to propose a lightweight yet highly accurate streaming-based trajectory forecasting approach. We integrate valuable information from previous predictions with a novel endpoint-aware modeling scheme. Our temporal context propagation uses the trajectory endpoints of the previous forecasts as anchors to extract targeted scenario context encodings. Our approach efficiently guides its scene encoder to extract highly relevant context information without needing refinement iterations or segment-wise decoding. Our experiments highlight that our approach effectively relays information across consecutive timesteps. Unlike methods using multi-stage refinement processing, our approach significantly reduces inference latency, making it well-suited for real-world deployment. We achieve state-of-the-art streaming trajectory prediction results on the Argoverse 2 multi-agent and single-agent benchmarks, while requiring substantially fewer resources.

1. Introduction

Trajectory prediction, *i.e.*, forecasting the future movements of nearby traffic participants, is a key component of the autonomy stack in self-driving vehicles. Trajectory prediction is essential for efficient and reliable navigation, as it enables autonomous vehicles to anticipate the behavior of surrounding agents and plan maneuvers proactively in response to predicted motions. The latency of the trajectory prediction model directly contributes to the total delay between sensor observations and vehicle control actions as it is typically performed between the perception and motion planning modules. Thus, for practical deployment in autonomous vehicles, trajectory prediction models must produce highly accurate predictions with minimal latency.

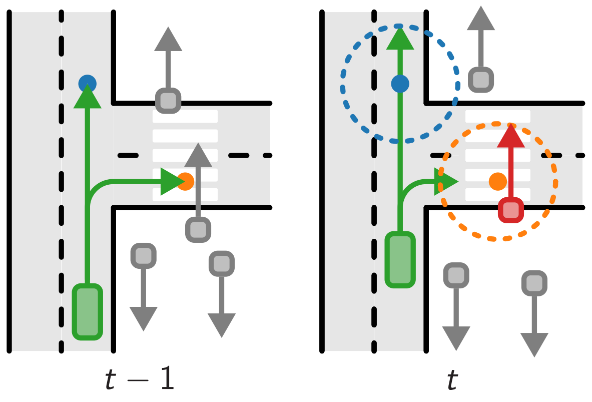


Figure 1. Traffic scenes are constantly changing, which is often neglected in trajectory prediction methods operating on snapshots rather than in a continuous setting. For example, for the trajectory prediction of the turning process of the focal agent (green), a possible interaction with a pedestrian (red) near the endpoint of the previous estimation (at time $t-1$) is important context that needs to be considered in the current prediction (t). We model this temporal context propagation with a novel endpoint streaming mechanism, achieving accurate predictions at minimal latencies.

One of the key challenges in trajectory forecasting is identifying the most influential map elements and the most relevant neighboring agents for motion prediction, especially in complex scenarios with long prediction horizons, often exceeding 100 meters. While the initial agent position provides a strong prior for relevant context elements, scene elements along the future path play a crucial role, as illustrated in Figure 1. Recent trajectory prediction approaches commonly use refinement schemes [28, 42, 43] to incorporate this knowledge: Initial coarse future trajectories are iteratively refined to enhance prediction accuracy. For instance, SmartRefine [42] proposes an adaptive multi-stage refinement scheme with scene context re-encoding to improve predictions of the strong baseline model QCNet [43]. Methods that use such refinement iterations achieve highly accurate results on large-scale benchmarks like Argoverse 2 (AV2) [37]. However, they impose high computational cost and latency.

This limits their applicability in real-world autonomous systems, where fast and efficient decision-making is critical and methods with higher latency can lead to less reliable results due to outdated sensor observations.

In addition, current research in trajectory prediction focuses on processing and evaluating snapshots of individual scenarios in benchmark datasets [9, 37]. This snapshot-based treatment is in sharp contrast to the application in real autonomous vehicles, where trajectory prediction systems operate in a continuous environment. In this real-world scenario, additional constraints must be considered. In particular, the consistency across successive predictions, which is essential for integration into downstream decision-making systems. On the other hand, continuous streaming-based processing also offers advantages, such as using contextual information across multiple prediction frames to improve the accuracy of future motion estimates. Recently, pioneering streaming-based approaches [24, 30] have emerged, laying the foundation for addressing the challenges of continuous trajectory prediction. RealMotion [30] demonstrates that a simple backbone architecture combined with streaming mechanisms achieve promising results on the AV2 dataset [37]. However, their approach does not use all available temporal information to encode the scene context, for example by not taking into account past predictions.

To advance towards consistent and efficient continuous real-world motion forecasting, we introduce a novel streaming-based trajectory prediction approach that uses temporal information for targeted scene context extraction. In particular, we propose a streaming endpoint-aware modeling (SEAM) architecture to effectively incorporate information across consecutive frames. By leveraging prediction endpoints from the previous prediction frame as prior information, we directly integrate target region information into our modeling without requiring resource-intensive refinement iterations. In-depth evaluations on AV2 [37], both single and multi-agent benchmarks, show that our new streaming mechanism provides significant advantages over previously proposed streaming techniques [30]. We demonstrate that our approach outperforms more complex and resource-demanding models, while also addressing real-world considerations like reducing fluctuations across multiple predictions for the same agent. It combines high accuracy with minimal inference latency, achieving state-of-the-art results for streaming-based trajectory prediction on the AV2 benchmarks. In summary, our main contributions include:

- We propose a new trajectory prediction model, effectively leveraging the consecutive nature of scenarios for target-centric context encoding without any refinement steps.
- Despite achieving minimal inference latency, we demonstrate that our model outputs excellent results on complex trajectory prediction tasks and sets a new state-of-the-art in streaming processing on the AV2 multi-agent benchmark.
- Our ablation study highlights the effectiveness of our endpoint-aware modeling in propagating information from the previous prediction timestep.

2. Related Work

Trajectory prediction methods typically take agent states and map information as input. Map data is predominantly modeled as vectorized data [12, 32] and encoded by PointNet-like architectures [26], while agent time series information is commonly processed by recurrent architectures [21, 32], attention-based modules [4, 17, 25, 30] or state-space models [39]. To capture relationships between agents and map elements, recent methods utilize graph neural networks [6, 12, 16, 18, 31, 38] or attention-based encoders [4, 11, 19, 22, 23, 28, 41]. Trajectory decoding is performed using either multilayer perceptrons (MLPs) [4, 30] or more advanced cross-attention-based designs [28, 29, 39].

2.1. Refinement Modules for Trajectory Prediction

Iterative refinement is a common strategy in state-of-the-art trajectory prediction approaches [11, 28, 34, 42, 43]. Long-horizon benchmarks like Argoverse 2 [37] and WOMD [9] involve predictions spanning over 100 meters, resulting in a large number of scene context elements. While the initial agent position helps to identify the relevant context, detecting critical elements at long distances remains challenging based solely on the given past motion of the agent. To address this, multiple refinement steps can be applied to progressively extract scene context elements that have a strong influence on future agent behavior [28, 42]. Initial predictions provide strong guidance for identifying key scene elements which can be used to iteratively refine the trajectory hypotheses.

MTR [28] and its follow-up works [7, 11, 29] iteratively refine trajectory predictions, using dynamic intention queries for aggregating scene context during decoding. R-Pred [5] proposes a two-stage forecasting approach where initial predictions define scene context tubes which are then utilized by a refinement network to generate improved predictions. QCNet [43] employs a two-stage decoding process, where initial predictions are used as priors in a refinement stage by incorporating them into the mode query of a cross-attention decoder. Additionally, at each inference step, it divides future trajectory prediction into shorter temporal segments, which are predicted recurrently. SmartRefine [42] proposes an adaptive refinement iteration strategy, which is demonstrated for QCNet. At each inference step, an adaptive number of trajectory decoding iterations are performed to progressively refine predictions based on updated scene encoding. Recently, DeMo [39] introduces a decoupled query strategy, first modeling motion state consistency and multi-modal intents separately before decoding the final trajectory output.

Overall, methods like DeMo [39] and SmartRefine [42] provide strong evidence that multi-stage decoding improves trajectory prediction quality. However, the gains in accuracy are often only small, especially considering the significantly higher computing costs required for executing a multi-stage refinement scheme at each prediction time step. This diminishing return underscores the need for novel strategies to balance computational cost and accuracy, when designing trajectory prediction models for real-world applications.

2.2. Streaming Trajectory Prediction

Current trajectory prediction research primarily focuses on snapshot-based prediction, where individual scenarios are processed without considering global temporal context across evolving scenes. In practice, however, autonomous systems operate in a continuous environment and process data in a streaming manner, introducing additional requirements like temporal consistency, as well as opportunities for improvement by leveraging additional temporal information.

Pang *et al*. [24] present an early contribution to motion forecasting in continuous, streaming scenarios. Their predictive streamer extends snapshot-based trajectory prediction models with a custom occlusion reasoning module and a differentiable filter to enhance temporal consistency. RealMotion [30] shows that incorporating context and predictions from previous frames enables accurate results even with a simple backbone [4]. They introduce a context stream applying cross-attention on ego-motion-aligned context from current and previous frames, and a trajectory relay mechanism that refines current predictions via offsets from embedded past predictions. DeMo [39] also integrates both mechanisms into their approach. HPNet [31] includes the exploration of a basic historical prediction attention module on the comparatively simple AV1 dataset [3]. QCNet [43] explores a key-value cache for past observations, touching on concepts related to streaming processing. However, it does not consider past predictions or ensure alignment across prediction steps, and its compute-intensive decoder poses significant challenges for real-time deployment.

Overall, existing streaming-based methods incorporate mechanisms to maintain the consistency of successive predictions [24, 30] and leverage the propagation of agent-centric scene context across timesteps [30] to address the continuous nature of real-world autonomous driving. However, they do not explore the use of the previously predicted trajectories as prior knowledge for context encoding. Given the importance of these regions for extracting relevant contextual features [28, 42], we propose an efficient strategy that includes this information in a single decoding step, thereby avoiding the high computational overhead of refinement modules.

3. Streaming Trajectory Prediction Using Endpoint-Aware Modeling

We propose SEAM, an accurate yet lightweight trajectory prediction method based on a streaming-based processing paradigm. Unlike refinement-based approaches [28, 42], which use multi-stage decoding to iteratively refine predictions by aggregating relevant scene context, SEAM introduces a novel mechanism that propagates predictions across timesteps to extract endpoint-centered target region features. This design enables SEAM to achieve high accuracy with low latency by avoiding the computational overhead of refinement iterations. Furthermore, by leveraging past predictions to identify potential target regions, SEAM addresses a key limitation of prior streaming-based methods [24, 30], which ignore this strong temporal prior. Our approach aligns well with the continuous nature of real-world autonomous driving: as future prediction timesteps unfold, interactions with other agents may evolve and must be dynamically considered. For example, recall Figure 1: when predicting a turn, a potentially crossing pedestrian on the sidewalk must be considered. As illustrated in overview Figure 2, we leverage past prediction endpoints to guide the model towards relevant regions, allowing us to achieve highly accurate future trajectories in a single decoding step, eliminating the need for computationally expensive refinement iterations.

The core component of our approach is the streaming-based dual-region encoding mechanism that uses the endpoints from past predictions to enhance scene context aggregation. In addition to conventional agent-centric context encoding, which is centered at the current position of the focal agent, we introduce a second set of context features: We extract *target-centric* features by retrieving scene elements closely around the endpoints of the previously predicted trajectory as these regions are likely to have high relevance for the current frame's prediction. We encode them using a shallow target encoder with the trajectory endpoints as coordinate system origins. During decoding, we execute attention to both agent-centric and target-centric features.

We describe the baseline encoding in Sections 3.1 and 3.2. Next, we introduce our *endpoint-aware modeling (EAM)*, which includes the novel target-centric feature encoding (Section 3.3) and its integration into the proposed dual-context decoder (Section 3.4). Section 3.5 outlines the integration of baseline streaming mechanisms [30]. We present our extension to the multi-agent setting in Section 3.6.

Input Representation: Trajectory prediction refers to forecasting future trajectories F given historical agent states A and lane data L. We follow the common input representation of state-of-the-art work [4, 25, 30, 39] and initially sample all agents and map elements within a fixed radius around the focal agent. A tensor $A \in \mathbb{R}^{N_a \times T_h \times D_a}$ represents the historical trajectories, where N_a is the number of agents, T_h represents the number of historical timesteps and D_a the feature dimension of the motion states, *i.e.* positions and velocity. Each lane is represented by sampling P_l points along its centerline. We define lane data as a tensor $L \in \mathbb{R}^{N_l \times P_l \times D_l}$, where N_l denotes the number of lane

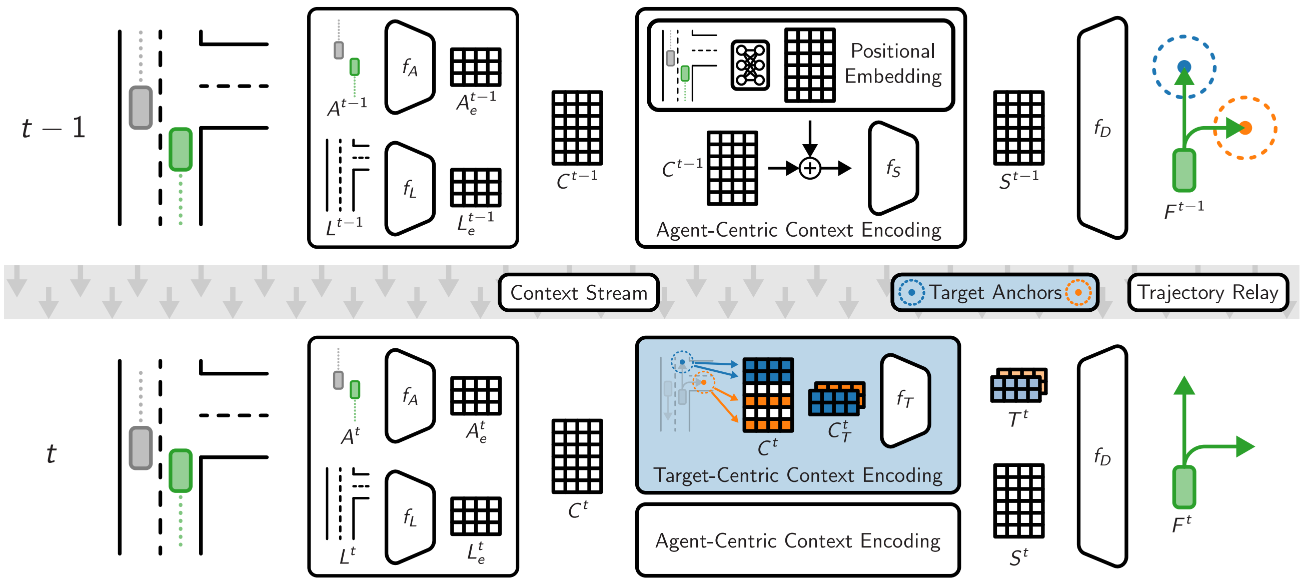


Figure 2. Overview of our streaming-based trajectory prediction architecture SEAM. We assume that for frame $t-1$ no previous prediction exists for our focal agent. Thus, only a standard trajectory prediction model pass is executed. In the next frame t, we incorporate the endpoints from the previous predictions to aggregate target-centric context information. We encode them using a second encoder path and provide it as additional input to our novel dual-context decoder.

segments and D_l represents the feature dimension, *i.e*. the xy-coordinates of the centerline. To facilitate efficient encoding of historical movements and lane data, we first transform each element in a local coordinate system [4]. We normalize the historical trajectories w.r.t. their initial pose while coordinates of lane centerlines are given w.r.t. their center position. This enables the model to learn motion patterns and lane segment shapes independently of global positions.

Output Representation: Our model predicts for a focal agent the multi-modal future trajectory hypotheses $F \in \mathbb{R}^{K \times T_f \times 2}$, where T_f denotes the number of future timesteps and K the number of motion modes, along with associated probability scores $P \in \mathbb{R}^K$. We represent future trajectories in xy-coordinates. Thus, trajectory prediction can be formulated as $(F, P) = f(A, L)$, where f is our trajectory prediction model. More specifically, our model consists of an encoder f_E generating the scene context $S = f_E(A, L)$ and a trajectory decoder f_D using the scene context $S \in \mathbb{R}^{N_a+N_l \times D}$ to predict trajectories F and probability scores P as $(F, P) = f_D(S)$.

Streaming Processing: To align with real-world autonomous driving applications, we adopt a streaming processing scheme to predict the future trajectories at the current timestep t. We leverage previously encoded context information S^{t-1} and past trajectory predictions F^{t-1} to generate highly accurate predictions and ensure temporally consistent trajectories. Consequently, our trajectory prediction at time t can be formulated as $(F^t, P^t) = f(A^t, L^t, S^{t-1}, F^{t-1})$.

3.1. Agent and Lane Encoding

We follow [17, 25] to encode the historical agent movement by projecting the agent data A^t into the D-dimensional feature space $\mathbb{R}^{N_a \times T_h \times D}$ using a linear layer. Next, we apply multi-head self-attention [33] along the temporal dimension to capture the agent motion dynamics and use a pooling operation to aggregate temporal information. This yields an agent representation $A_e^t \in \mathbb{R}^{N_a \times D}$ where the movement of each agent is encoded in a feature vector $\mathbb{R}^{1 \times D}$. To encode the lane data, we follow the default state-of-the-art approach [4, 25, 28–30, 39] of using a mini-PointNet-like [26] network. This results in a lane context representation $L_e^t \in \mathbb{R}^{N_l \times D}$. Thus, our encoders f_A and f_L generate agent feature tokens $A_e^t = f_A(A^t)$ and lane feature tokens $L_e^t = f_L(L^t)$ for further encoding.

3.2. Agent-Centric Context Encoding

After encoding each scene context element individually, the relationships among all scene elements must be learned. To achieve this, we concatenate the agent A_e^t and lane L_e^t tokens to form a scene context representation $C^t \in \mathbb{R}^{N_c \times D}$, where $N_c = N_a + N_l$. To incorporate categorical information, like different agent and lane types, we add learnable type embeddings [4] to the scene tokens. A common way for encoding relations among map elements and agents is self-attention across scene tokens [4, 17, 25, 39, 42]. To this end, a global positional encoding must be established, which is commonly done using a focal agent-centric coordinate system. We encode the global poses using the position (x, y) and the rotation yaw of each local token coordinate system,

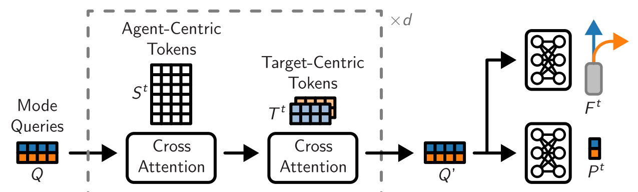


Figure 3. Architecture of our novel dual-context attention decoder, leveraging both agent-centric S^t and target-centric T^t features.

i.e. agent track origin and lane segment center. We apply a sine and cosine transformation to the yaw and process the pose information $\mathbb{R}^{N_c \times 4}$ using a two-layer multilayer perceptron (MLP). Following, we add the positional embeddings to our context tokens $C^t \in \mathbb{R}^{N_c \times D}$ and apply multi-head attention in our scene encoder f_S. This yields the encoded scene context $S^t = f_S(C^t)$ with $S^t \in \mathbb{R}^{N_c \times D}$.

3.3. Target-Centric Context Encoding

The agent-centric encoding implicitly guides the model to focus on the surroundings of the focal agent, where nearby context elements are highly relevant for predicting future motion. However, as highlighted in Figure 1, also the areas where the agent is heading to are highly important for accurate predictions. Since we operate in a streaming-based setting, each agent that was visible in the previous frame has multi-modal past trajectory predictions, denoted as $F^{t-1} \in \mathbb{R}^{K \times T_f \times 2}$. These previous predictions F^{t-1} provide a strong prior for scenario regions that are also crucial for context extraction. We use the endpoints of these trajectories as anchors to define K target regions. Next, we gather tokens from our scene context C^t within a region-of-interest radius r around each target anchor, forming K individual target context token sets $C^t_T \in \mathbb{R}^{K \times N'_c \times D}$. In practice, N'_c is significantly smaller than N_c making our target encoding computationally less expensive than the main agent-centric encoding.

Following the same process as for the agent-centric scene context S, we encode each target-centric context C^t_T individually in our target encoder f_T, obtaining $T^t \in \mathbb{R}^{K \times N'_c \times D}$. To capture the relationship between the global coordinate root and each target anchor coordinate frame, we also add positional embeddings for the transformation to the target anchor of each region (see our supplementary for details).

3.4. Decoding with Dual-Context Attention

Our decoder f_D generates future trajectories F^t and probability scores P^t based on the agent-centric scene context S^t and our novel target-centric scene context T^t, formulated as $(F^t, P^t) = f_D(Q, S^t, T^t)$. We employ learnable queries $Q \in \mathbb{R}^{K \times D}$ to obtain multi-modal outputs and implement a DETR-like [2] decoding approach. Figure 3 illustrates the architecture of our dual-context attention decoder.

We iteratively execute cross-attention d-times to our scene context $S \in \mathbb{R}^{N_c \times D}$, and the target-specific tokens $T \in \mathbb{R}^{K \times N'_c \times D}$. Thus, information from both feature sets is available in each decoder stage and integrated into our final output. The agent-centric context S provides local information, while our target context T gives the model fine-grained information on long-horizon dependencies with other agents and map elements. Each mode query only attends to one associated set of target tokens $T \in \mathbb{R}^{1 \times N'_c \times D}$. Finally, we project the decoded queries Q' to future trajectories $F^t \in \mathbb{R}^{K \times T_f \times 2}$ and probability scores $P^t \in \mathbb{R}^K$ using a dedicated two-layer MLP for each output.

3.5. Context Referencing and Trajectory Relaying

In addition to our novel endpoint-aware modeling, we leverage context referencing and trajectory relaying [30] to model the streaming setting through information flow from the previous frame $t-1$. We follow the setup of [30] for both mechanisms to enable fair comparison: To account for focal agent motion between $t-1$ and t, we apply motion-aware layer normalization (MLN) [35] to the scene context S^{t-1} before integrating it into the current encoding C^t via agent-to-scene and map-to-map cross-attention. Trajectory relay employs a cross-attention-based interaction module to refine the current predictions F^t using past predictions F^{t-1}.

3.6. Extension to Multi-Agent Settings

To predict accurate trajectories for all agents N_a in a scene, we first generate marginal predictions for each agent individually. To this end, we treat each agent as focal agent once for our single-agent model, stack the input data and perform an inference pass with a batch size of N_a. Next, we introduce a lightweight global consistency module, to fuse the marginal predictions to joint, globally consistent world predictions.

Our consistency module takes the mode features $Q' \in \mathbb{R}^{N_a \times K \times D}$ for each agent and their global positions $R \in \mathbb{R}^{N_a \times 4}$ as input. A shallow MLP encodes the global positions into positional embeddings. We then apply self-attention across all modes per agent (over K) as well as self-attention across all agents per mode (over N_a) to capture intra- and inter-agent relationships. Following our single agent decoder design, we decode the final world trajectory predictions using two shallow MLPs. As output, we obtain $F_w \in \mathbb{R}^{K \times N_a \times T_f \times 2}$ world predictions with associated confidence scores $P_w \in \mathbb{R}^K$.

4. Experiments

We present experimental results on the single-agent and multi-agent benchmarks of the Argoverse 2 (AV2) [37] dataset, along with a latency analysis for both settings to assess the practical applicability of our approach. Additionally, we provide an ablation study for our endpoint-aware modeling on both AV2 and Argoverse (AV1) [3], with further results included in the supplementary material.

Method	Streaming	minADE_1	minFDE_1	MR_6	minADE_6	minFDE_6	brier-minFDE_6
SIMPL [40]	✗	2.03	5.50	0.19	0.72	1.43	2.05
HPTR [41]	✗	1.84	4.61	0.19	0.73	1.43	2.03
Forecast-MAE [4]	✗	1.74	4.36	0.17	0.71	1.39	2.03
MTR [28]	✗	1.74	4.39	0.15	0.73	1.44	1.98
EMP-D [25]	✗	1.75	4.35	0.17	0.71	1.37	1.98
GANet [34]	✗	1.77	4.48	0.17	0.72	1.34	1.96
ProIn [8]	✗	1.77	4.50	0.18	0.73	1.35	1.93
RealMotion [30] *	✓	1.65	4.12	0.15	0.67	1.29	1.93
QCNet [43]	✗	1.69	4.30	0.16	0.65	1.29	1.91
RealMotion [30] (Train/Val)*	✓	<u>1.59</u>	<u>3.93</u>	0.15	0.66	1.24	1.89
Tamba [14]	✗	1.66	4.24	0.17	0.64	1.24	1.89
ProphNet [36]	✗	1.80	4.74	0.18	0.68	1.33	1.88
MTR++ [29]	✗	1.64	4.12	<u>0.14</u>	0.71	1.37	1.88
DyMap [10]	✗	-	-	-	0.71	1.29	1.87
SmartRefine [42]	✗	1.65	4.17	0.15	<u>0.63</u>	<u>1.23</u>	<u>1.86</u>
DeMo [39]	✓	**1.49**	**3.74**	**0.13**	**0.61**	**1.17**	**1.84**
SEAM (Ours)	✓	1.60	3.95	0.15	0.66	1.24	**1.84**

Table 1. Single-agent trajectory prediction results on the Argoverse 2 test set. For all metrics lower indicate better, table sorted in descending order by brier-minFDE_6. We report results for published works on the official leaderboard without model ensembling. The streaming approach of Pang *et al*. [24] is not included (their custom dataset prevents direct comparison). *For RealMotion [30] we report the results for their official checkpoint, as well as the results from their paper, which is trained on the train and validation set.

4.1. Experimental Settings

Dataset: We evaluate our model on the Argoverse 2 (AV2) [37] trajectory prediction dataset. It has a sampling rate of 10 Hz and contains 199,908 training, 24,988 validation, and 24,984 test scenarios collected in six U.S. cities. In each scenario the first 5 s serve as historical context and the goal is to predict the trajectories for the subsequent 6 s. The long historical context makes AV2 ideal for evaluation in streaming processing, as it allows a reasonable sliding-window processing (see below). In contrast, WOMD [9] provides only 1 s of history and nuScenes [1] offers 2 s but at only 2 Hz. Further reducing this few historical data points to multiple windows limits the agent encoder's ability to learn motion patterns. Both are also not considered for streaming processing by related work [24, 30, 39]. We evaluate on the single-agent (predict trajectories for one agent per scenario) and multi-agent (predict globally consistent trajectories for multiple agents per scenario) benchmarks of AV2. Additionally, we provide an ablation study on Argoverse 1 (AV1) [3] which features 2 s history and considers a future of 3 s.

Metrics: We evaluate our approach using the standard AV2 [37] benchmark metrics. For the single-agent case, we use the miss rate (MR_k), minimum average displacement error (minADE_k), minimum final displacement error (minFDE_k), and Brier minimum final displacement error (brier-minFDE_k). Following the official leaderboard, we evaluate using the top $k \in \{1, 6\}$ scoring trajectories. For the multi-agent case we utilize the actorMR_k, avgMinADE_k, avgMinFDE_k and avgBrierMinFDE_k which account for the mean errors across all actors in each world.

Streaming Processing: To enable streaming processing using existing benchmarks, we follow [30, 39] by limiting the historical model input T_h to a fraction of the available history. We then operate using a sliding window and conduct multiple model executions. In particular, for AV2 we use a historic input of $h = 3$ s and execute three predictions at $t \in \{3, 4, 5\}$s. The final prediction is made at the same time point as in snapshot-based methods, allowing a fair comparison. We provide a detailed comparison of streaming and snapshot-based processing in our supplementary material.

4.2. Argoverse 2 Single-Agent Results

Table 1 shows the results for the Argoverse 2 single-agent test set. Our SEAM, along with DeMo [39], achieves the best score on the main metric (brier-minFDE_6), outperforming recent work like Tamba [14] and even more resource-intensive methods, such as QCNet [43] and SmartRefine [42]. This strong performance without any refinement iterations highlights the effectiveness of our endpoint-aware modeling. We also outperform RealMotion [30], a prior streaming-based prediction method, while also achieving lower inference latency (Sec. 4.4). DeMo [39] achieves slightly lower displacement errors since their decoupled decoding yields more diverse trajectories, but at a significantly higher latency (Sec. 4.4). The small gap between minFDE_6 to brier-minFDE_6 (induced by the main metric's probability penalty) demonstrates our model's strong confidence estimation, which is also crucial for downstream integration into the planning module. This highlights that directly integrating target-centric features helps assess maneuver likelihood.

Method	Streaming	avgMinADE$_1$	avgMinFDE$_1$	actorMR$_6$	avgMinADE$_6$	avgMinFDE$_6$	avgBrierMinFDE$_6$
FJMP [27]	✗	1.52	4.00	0.23	0.81	1.89	2.59
Forecast-MAE [4]	✗	1.30	3.33	0.19	0.69	1.55	2.24
RealMotion [30]	✓	1.14	2.97	0.18	0.62	1.32	2.01
DeMo [39]	✓	1.12	2.78	0.16	0.58	1.24	1.93
SEAM (Ours)	✓	1.05	2.57	0.16	0.62	1.21	1.85
QCNeXt [44]	✗	**1.03**	**2.55**	**0.14**	**0.54**	**1.13**	**1.79**

Table 2. Multi-agent trajectory prediction results on the Argoverse 2 test set. For all metrics lower values indicate better performance, table sorted in descending order by avgBrierMinFDE$_6$. For all methods we report results without model ensembling.

Method	Streaming	Latency ($B = 1$)		Latency ($B = 32$)		Latency ($B = 64$)		Model	AV2 Test Set
		Offline	Online	Offline	Online	Offline	Online	Parameters	brier-minFDE$_6$
SmartRefine [42]	✗	>118 ms*		>680 ms*		-		8.0M	1.86
QCNet [43]	✗	118 ms		680 ms		-		7.7M	1.91
RealMotion [30]	✓	**86 ms**	**24 ms**	247 ms	88 ms	436 ms	150 ms	**2.9M**	1.93
DeMo [39]	✓	139 ms	39 ms	275 ms	83 ms	466 ms	142 ms	5.9M	**1.84**
SEAM (Ours)	✓	91 ms	28 ms	**140 ms**	**50 ms**	**246 ms**	**74 ms**	4.6M	**1.84**

Table 3. Latency analysis for predicting B Argoverse 2 scenarios in the single-agent setting using a NVIDIA V100 GPU. *Online* (for streaming-based methods) is the time required to process a single sliding window; *Offline* is the time required to process the full stream (to generate the benchmark predictions). For streaming methods the *online* latency is relevant for practical applications. *SmartRefine does not provide an implementation for AV2—since its main results add refinement iterations to QCNet, latency is higher than for QCNet.

4.3. Argoverse 2 Multi-Agent Results

Previous methods, *e.g.* [14, 42], often report only single-agent results. However, the multi-agent benchmark is more relevant for practical systems: models deployed in real-world settings must be capable to predict trajectories for more than one agent in a scene at the same time. The results in Table 2 show that, in this setting, our SEAM outperforms previous streaming-based methods [30, 39] in the main metrics. The current best-performing method is the snapshot method QCNeXt [44], an extension of QCNet [43]. However, the displacement errors differ only by a few centimeters, whereas its extensive use of recurrence and refinement iterations in decoding poses significant challenges for real-time deployment. Since model latency directly impacts the delay between sensor observations and prediction output, models with computational overhead may lead to worse results in real-world scenarios due to delayed response. For reference, an agent traveling at 50 km/h covers approximately 0.7 m in 50 ms. Thus, marginally lower displacement errors at increased processing delay offer no benefit in practice.

4.4. Latency Analysis

Argoverse 2 Single-Agent Setting: Table 3 compares latencies for predicting various batch sizes of AV2 single-agent scenarios. Our approach has favorable runtime compared to related work, especially since it scales well with larger batch sizes which are relevant in practice as the single-agent benchmark evaluates only one agent per scenario. SEAM achieves minimal latency by using only a single decoder pass, unlike DeMo [39], QCNet [43] and SmartRefine [42], which employ multiple decoding stages. Additionally, it predicts the complete future trajectory without splitting it into segments, as done in QCNet and SmartRefine. In comparison to RealMotion [30] our more efficient decoder design leads to a latency advantage. The fast processing speed makes our approach highly suitable for real-world deployment, as it combines streaming modeling with computational efficiency — two key considerations for practical use.

Argoverse 2 Multi-Agent Setting: Related work on streaming-based processing (RealMotion [30] and DeMo [39]) does not provide details on their multi-agent implementations. Moreover, no official implementation is available for QCNeXt [44]. As a result, a direct latency comparison is not possible. However, based on the nature of the task, we can reasonably assume that per-scenario latency in a multi-agent setting exceeds the single-agent latencies reported in Table 3. For our model, we measure an average online latency of *38 ms* per scenario in the multi-agent setting on the AV2 validation set using a single NVIDIA V100 GPU (batch size $B = 1$). Notably, this is even faster than the single-agent prediction latencies reported for DeMo and QCNet, highlighting the high efficiency of our approach.

4.5. Ablation Studies

Information Streaming Mechanisms: In Table 4 we compare our endpoint-aware modeling (EAM), which includes the target-centric feature encoding and the dual-context decoder, to the context stream (CS) and trajectory relay (TR) streaming mechanisms introduced by previous work [30]. The top row in each group presents the results

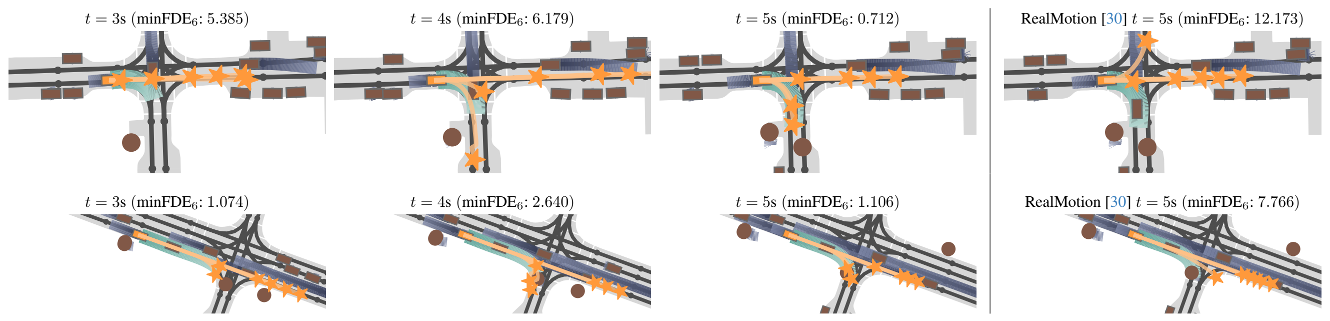

$t = 3$s (minFDE$_6$: 5.385)
$t = 4$s (minFDE$_6$: 6.179)
$t = 5$s (minFDE$_6$: 0.712)
RealMotion [30] $t = 5$s (minFDE$_6$: 12.173)
$t = 3$s (minFDE$_6$: 1.074)
$t = 4$s (minFDE$_6$: 2.640)
$t = 5$s (minFDE$_6$: 1.106)
RealMotion [30] $t = 5$s (minFDE$_6$: 7.766)

Figure 4. Qualitative results on two Argoverse 2 scenarios. 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 RealMotion [30] in the streaming-based setting. Top row: the focal agent is approaching an intersection where other traffic is currently passing by, making it difficult to identify possible movement. Bottom row: a pedestrian crosses the street at an intersection to the right. Our approach correctly predicts that the vehicle can either continue straight or turn right, either waiting before the crosswalk or proceeding directly if the pedestrian has already left the crosswalk.

Dataset	h	Stream	CS+TR [30]	EAM	mADE$_6$	mFDE$_6$	b-mFDE$_6$
AV1 Val	2s	✗	✗	✗	0.64	1.04	1.67
	1s	✓	✓	✗	0.62	0.98	1.62
	1s	✓	✗	✓	0.60	0.95	1.55
	1s	✓	✓	✓	**0.59**	**0.93**	**1.52**
AV2 Val	5s	✗	✗	✗	0.77	1.39	2.00
	3s	✓	✓	✗	0.75	1.29	1.93
	3s	✓	✗	✓	0.70	1.27	1.88
	3s	✓	✓	✓	**0.66**	**1.25**	**1.85**

Table 4. Comparison of different information forwarding mechanisms for streaming trajectory prediction. We compare the integration of context stream (CS) and trajectory relay (TR) introduced by [30] to our endpoint-aware modeling (EAM). The first line shows the traditional snapshot-based use. We highlight **best** values.

of our backbone architecture operating in the traditional snapshot-based prediction setting without any streaming mechanism. Comparing streaming-based processing using CS and TR to our approach highlights the effectiveness of our EAM in propagating information from the previous frame. Our prediction endpoint-aware modeling achieves a larger improvement than the two streaming mechanisms proposed by [30]. Leveraging endpoint information in the streaming setting proves to be highly beneficial for enhancing trajectory prediction accuracy. Overall, combining our EAM with CS and TR leads to the best results.

Target Context Radius: We evaluate the influence of different target context radii r in Table 5. Performance saturates at a radius of 30 meters, as incorporating further context at larger radii does not provide relevant information for our endpoint-aware modeling. A larger radius increases the number of target context tokens and thus latency, making a radius of 30 meters an optimal trade-off between performance and efficiency for practical applications. Notably, our approach remains effective even with a smaller radius of 10 meters, as injecting previous endpoint positions into the decoder already provides a strong prior. Incorporating the endpoint location during decoding also guides cross-attention toward

Dataset	r	mADE$_6$	mFDE$_6$	b-mFDE$_6$
AV2 Val	10m	0.68	1.27	1.89
	15m	0.67	1.26	1.87
	30m	**0.66**	**1.25**	**1.85**
	45m	**0.66**	**1.25**	**1.85**

Table 5. Comparison of different radii r to select context elements for our target-centric region. We highlight **best** values.

agent-centric features, further underscoring the importance of endpoint information in streaming processing.

4.6. Qualitative Results

Figure 4 shows predictions for two AV2 validation set scenarios, illustrating the evolving nature of streaming processing at $t \in \{3, 4, 5\}$s. To demonstrate the benefits of our endpoint-aware modeling against previous work on streaming trajectory prediction, we also show the output of RealMotion [30] at $t = 5$s. Additional qualitative results, including failure cases, are available in the supplementary material.

5. Conclusion

We propose a novel endpoint-aware modeling for streaming-based trajectory prediction. Our dual-context approach enables us to effectively incorporate the continuously evolving relevant scene context, eliminating the need for costly trajectory refinement iterations to achieve highly accurate predictions. We significantly outperform inference speed of highly iterative schemes [29, 42] and previous streaming based methods [30, 39] for practical scenario sizes, while achieving competitive results on the Argoverse 2 single-agent dataset and setting new state-of-the-art for streaming-based trajectory prediction on the multi-agent benchmark.

Acknowledgments. This work was partially funded by Addsafety (923936), a COMET Project funded by BMIMI, BMWET as well as the co-financing federal province of Styria. The COMET programme is managed by the FFG.

References

[1] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *CVPR*, 2020. 6

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. In *ECCV*, 2020. 5

[3] 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. 3, 5, 6, 11

[4] Jie Cheng, Xiaodong Mei, and Ming Liu. Forecast-MAE: Self-supervised Pre-training for Motion Forecasting with Masked Autoencoders. In *ICCV*, 2023. 2, 3, 4, 6, 7, 11, 12

[5] Sehwan Choi, Jungho Kim, Junyong Yun, and Jun Won Choi. R-Pred: Two-Stage Motion Prediction Via Tube-Query Attention-Based Trajectory Refinement. In *ICCV*, 2023. 2

[6] Alexander Cui, Sergio Casas, Kelvin Wong, Simon Suo, and Raquel Urtasun. GoRela: Go Relative for Viewpoint-Invariant Motion Forecasting. In *ICRA*, 2023. 2

[7] Tobias Demmler, Lennart Hartung, Andreas Tamke, Thao Dang, Alexander Hegai, Karsten Haug, and Lars Mikelsons. Dynamic Intent Queries for Motion Transformer-based Trajectory Prediction. In *IEEE IV*, 2025. 2

[8] Yinke Dong, Haifeng Yuan, Hongkun Liu, Wei Jing, Fangzhen Li, Hongmin Liu, and Bin Fan. ProIn: Learning to Predict Trajectory Based on Progressive Interactions for Autonomous Driving. *arXiv*, 2024. 6

[9] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles R Qi, Yin Zhou, et al. Large Scale Interactive Motion Forecasting for Autonomous Driving: The Waymo Open Motion Dataset. In *ICCV*, 2021. 2, 6

[10] Bin Fan, Haifeng Yuan, Yinke Dong, Zhengyu Zhu, and Hongmin Liu. Bidirectional Agent-Map Interaction Feature Learning Leveraged by Map-Related Tasks for Trajectory Prediction in Autonomous Driving. *IEEE Trans. Automation Science and Eng.*, 2025. 6

[11] 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, 11

[12] 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 *CVPR*, 2020. 2

[13] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *arXiv*, 2016. 12

[14] Yizhou Huang, Yihua Cheng, and Kezhi Wang. Trajectory Mamba: Efficient Attention-Mamba Forecasting Model Based on Selective SSM. In *CVPR*, 2025. 6, 7

[15] Peter J Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 1964. 12

[16] 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 TPAMI*, 2023. 2

[17] 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, 4, 11

[18] Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. Learning Lane Graph Representations for Motion Forecasting. In *ECCV*, 2020. 2

[19] Yicheng Liu, Jinghuai Zhang, Liangji Fang, Qinhong Jiang, and Bolei Zhou. Multimodal Motion Prediction with Stacked Transformers. *CVPR*, 2021. 2

[20] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019. 13

[21] Jean Mercat, Thomas Gilles, Nicole El Zoghby, Guillaume Sandou, Dominique Beauvois, and Guillermo Pita Gil. Multi-Head Attention for Multi-Modal Joint Vehicle Motion Forecasting. In *ICRA*, 2020. 2

[22] Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S Refaat, and Benjamin Sapp. Wayformer: Motion Forecasting via Simple & Efficient Attention Networks. In *ICRA*, 2023. 2

[23] Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, et al. SceneTransformer: A Unified Architecture for Predicting Multiple Agent Trajectories. *ICLR*, 2022. 2

[24] Ziqi Pang, Deva Ramanan, Mengtian Li, and Yu-Xiong Wang. Streaming Motion Forecasting for Autonomous Driving. In *IROS*, 2023. 2, 3, 6, 15

[25] 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, 3, 4, 6, 11

[26] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*, 2017. 2, 4

[27] Luke Rowe, Martin Ethier, Eli-Henry Dykhne, and Krzysztof Czarnecki. FJMP: Factorized Joint Multi-Agent Motion Prediction over Learned Directed Acyclic Interaction Graphs. In *CVPR*, 2023. 7

[28] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Motion Transformer with Global Intention Localization and Local Movement Refinement. In *NeurIPS*, 2022. 1, 2, 3, 4, 6

[29] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. MTR++: Multi-Agent Motion Prediction With Symmetric Scene Modeling and Guided Intention Querying. *IEEE TPAMI*, 2024. 2, 6, 8

[30] Nan Song, Bozhou Zhang, Xiatian Zhu, and Li Zhang. Motion Forecasting in Continuous Driving. In *NeurIPS*, 2024. 2, 3, 4, 5, 6, 7, 8, 11, 13, 14, 15, 16, 17, 18

[31] Xiaolong Tang, Meina Kan, Shiguang Shan, Zhilong Ji, Jinfeng Bai, and Xilin Chen. HPNet: Dynamic Trajectory Forecasting with Historical Prediction Attention. In *CVPR*, 2024. 2, 3

[32] Balakrishnan Varadarajan, Ahmed Hefny, Avikalp Srivastava, Khaled S Refaat, Nigamaa Nayakanti, Andre Cornman, Kan Chen, Bertrand Douillard, Chi Pang Lam, Dragomir Anguelov, et al. MultiPath++: Efficient Information Fusion and Trajectory Aggregation for Behavior Prediction. In *ICRA*, 2022. 2

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *NeurIPS*, 2017. 4

[34] Mingkun Wang, Xinge Zhu, Changqian Yu, Wei Li, Yuexin Ma, Ruochun Jin, Xiaoguang Ren, Dongchun Ren, Mingxu Wang, and Wenjing Yang. GANet: Goal Area Network for Motion Forecasting. In *ICRA*, 2023. 2, 6

[35] Shihao Wang, Yingfei Liu, Tiancai Wang, Ying Li, and Xiangyu Zhang. Exploring Object-Centric Temporal Modeling for Efficient Multi-View 3D Object Detection. In *CVPR*, 2023. 5

[36] Xishun Wang, Tong Su, Fang Da, and Xiaodong Yang. ProphNet: Efficient Agent-Centric Motion Forecasting with Anchor-Informed Proposals. In *CVPR*, 2023. 6

[37] 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, 2, 5, 6, 11, 13, 14

[38] Wenyuan Zeng, Ming Liang, Renjie Liao, and Raquel Urtasun. LaneRCNN: Distributed Representations for Graph-Centric Motion Forecasting. In *IROS*, 2021. 2

[39] Bozhou Zhang, Nan Song, and Li Zhang. DeMo: Decoupling Motion Forecasting into Directional Intentions and Dynamic States. In *NeurIPS*, 2024. 2, 3, 4, 6, 7, 8, 11, 13, 14, 15

[40] Lu Zhang, Peiliang Li, Sikang Liu, and Shaojie Shen. SIMPL: A Simple and Efficient Multi-agent Motion Prediction Baseline for Autonomous Driving. *RAL*, 2024. 6

[41] Zhejun Zhang, Alexander Liniger, Christos Sakaridis, Fisher Yu, and Luc Van Gool. Real-Time Motion Prediction via Heterogeneous Polyline Transformer with Relative Pose Encoding. In *NeurIPS*, 2023. 2, 6

[42] 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 *CVPR*, 2024. 1, 2, 3, 4, 6, 7, 8

[43] Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-Centric Trajectory Prediction. In *CVPR*, 2023. 1, 2, 3, 6, 7, 13

[44] Zikang Zhou, Zihao Wen, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. QCNeXt: A Next-Generation Framework For Joint Multi-Agent Trajectory Prediction. *arXiv*, 2023. 7

Streaming Real-Time Trajectory Prediction Using Endpoint-Aware Modeling

Supplementary Material

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

A. Streaming vs. Snapshot-based Processing

Figure 5 compares the standard snapshot-based trajectory prediction paradigm to the streaming-based processing employed by us and related work [30, 39]. 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 [37] we set the model input history to $h = 3\,\text{s}$, which corresponds to $T_h = 30$ input samples and we execute three predictions with a 1 s time gap in between at $t \in \{3, 4, 5\}$ s. Following the benchmark settings, we predict a future of 6 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 s seconds. For all evaluations and also for selecting our endpoint-centric features only the future up to T_f 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 [3] we use a historic input of 1 s ($T_h = 10$) and execute three passes at $t \in \{1, 1.5, 2\}$ s.

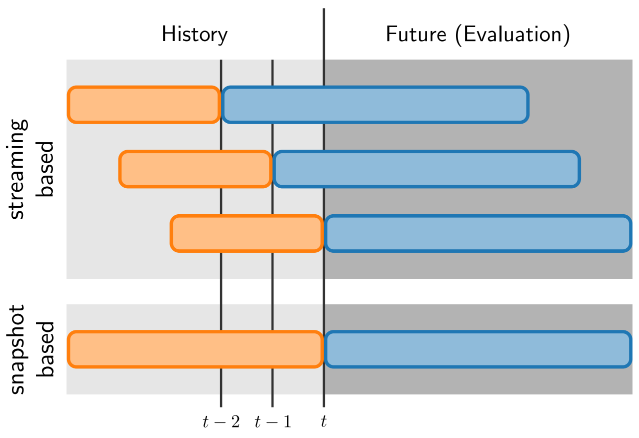


Figure 5. Comparison between snapshot-based and streaming-based trajectory prediction paradigms. Both approaches operate on the same benchmark input data, ensuring a fair comparison without additional data. In the streaming paradigm, past observations are processed using a sliding window, closely resembling 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.

B. 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 [4, 25, 30, 39]. To maintain a manageable number of scene tokens, we do not split the lanes into smaller segments as *e.g.* suggested by [11, 17]. 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 $D_l = 3$.

B.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

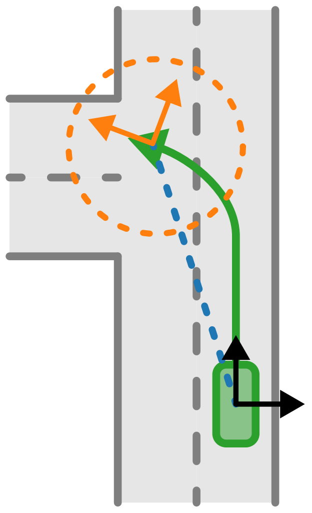


Figure 6. Modeling of target-centric coordinate system (orange) using the prediction (green) from the previous time frame. We also show the target context radius (orange) and the positional embedding (blue) used to model the relationship to the agent-centric origin (black).

and compute the orientation based on the curvature at the trajectory's end.

B.2. Target-Centric Coordinate Systems

Figure 6 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).

B.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 [13]
- Norm before attention operation

B.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 6 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 [13] activation function
- Layer 2: $D \times D$

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

B.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 (Appendix A), our decoder predicts T_a additional steps in addition to the T_f steps which are used as actual trajectories in the evaluation.

B.6. Optimization

We employ a common winner-takes-all strategy for training our model [4], 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 [15]) 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} [4], where a single trajectory is predicted for each non-focal agent present in the scene, and a smooth 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 [20] 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.

C. Multi-Agent Extension

C.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.

C.2. Optimization

To adapt our approach to the multi-agent setting in Argoverse 2 (AV2) [37], 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 $L = L_{\text{reg}} + L_{\text{cls}} + L_{\text{marginal}}$.

D. Evaluation Details

D.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.

D.2. Latency Measurements

We utilize the official code implementations to measure the latencies of QCNet[1] [43], RealMotion[2] [30], and DeMo[3] [39]. 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.

E. Additional Results

E.1. Streaming Methods on AV2 Validation Set

Table 7 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

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

Method	minADE$_1$	minFDE$_1$	MR$_6$	minADE$_6$	minFDE$_6$	brier-minFDE$_6$
RealMotion [30]	1.65	4.10	0.16	0.67	1.30	1.94
DeMo [39]	**1.48**	**3.73**	**0.13**	**0.61**	**1.19**	<u>1.86</u>
SEAM (Ours)	<u>1.60</u>	<u>3.96</u>	<u>0.15</u>	<u>0.66</u>	<u>1.25</u>	**1.85**

Table 7. 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**

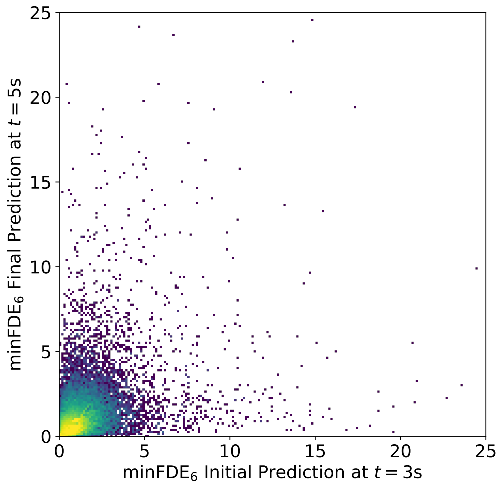
Table 8. Robustness of our endpoint-aware modeling on the AV2 [37] validation set by perturbing the prediction endpoints at $t \in \{3, 4\}$ s. These endpoints, which define the anchors for extracting our target-centric features, are modified using additive uniform noise ($\mathcal{U}$) or Gaussian noise ($\mathcal{N}$). The shown results correspond to the prediction errors at $t = 5$s.

likelihood of predicted trajectories, which is important for interpreting the results in downstream tasks like ego-motion planning.

E.2. Robustness to Error Propagation

Table 8 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 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 7. The plots compare the minFDE$_6$ for the first prediction, made at $t = 3$ s into the scenario, with the final prediction at $t = 5$ 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.



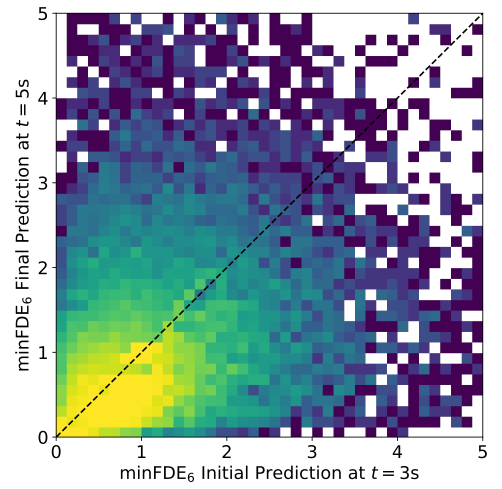


Figure 7. Density plots of prediction errors. Both plots compare the first prediction done at $t = 3$ s (x-axis) versus the final prediction done at at $t = 5$ s (y-axis). The top plot shows that even when the initial error is large, the model can recover and correct its prediction. The bottom plot provides 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.

Method	Latency ($B = 1$)		Latency ($B = 32$)		Latency ($B = 64$)	
	Offline	Online	Offline	Online	Offline	Online
RealMotion [30]	51 ms	**13 ms**	255 ms	85 ms	512 ms	172 ms
DeMo [39]	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 9. 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 [30]	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 10. 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 [30] 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.

Method	Fluctuation
RealMotion [30]	0.347
SEAM (Ours)	**0.341**

Table 11. We compare the trajectory fluctuation of our model to RealMotion [30]. The fluctuation metric defined by [24] gives an indication on the consistency of trajectories across multiple timesteps.

E.3. Latency on NVIDIA A10 GPU

We provide additional latency results using one NVIDIA A10 GPU, comparing streaming-based methods in Table 9. Also on this newer GPU architecture our approach achieves the best latency results, obtaining the lowest online and offline latency across all batch sizes.

E.4. Multi-Agent Ablation Study

We present an ablation study on extending single-agent approaches to the multi-agent setting in Table 10. 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).

E.5. Trajectory Fluctuation

Table 11 compares the trajectory fluctuation of our approach with that of RealMotion [30]. The fluctuation metric, as defined by [24] measures the consistency of trajectories across multiple prediction frames. Our approach achieves a lower fluctuation score than RealMotion, indicating more consistent predictions.

E.6. Result Visualizations

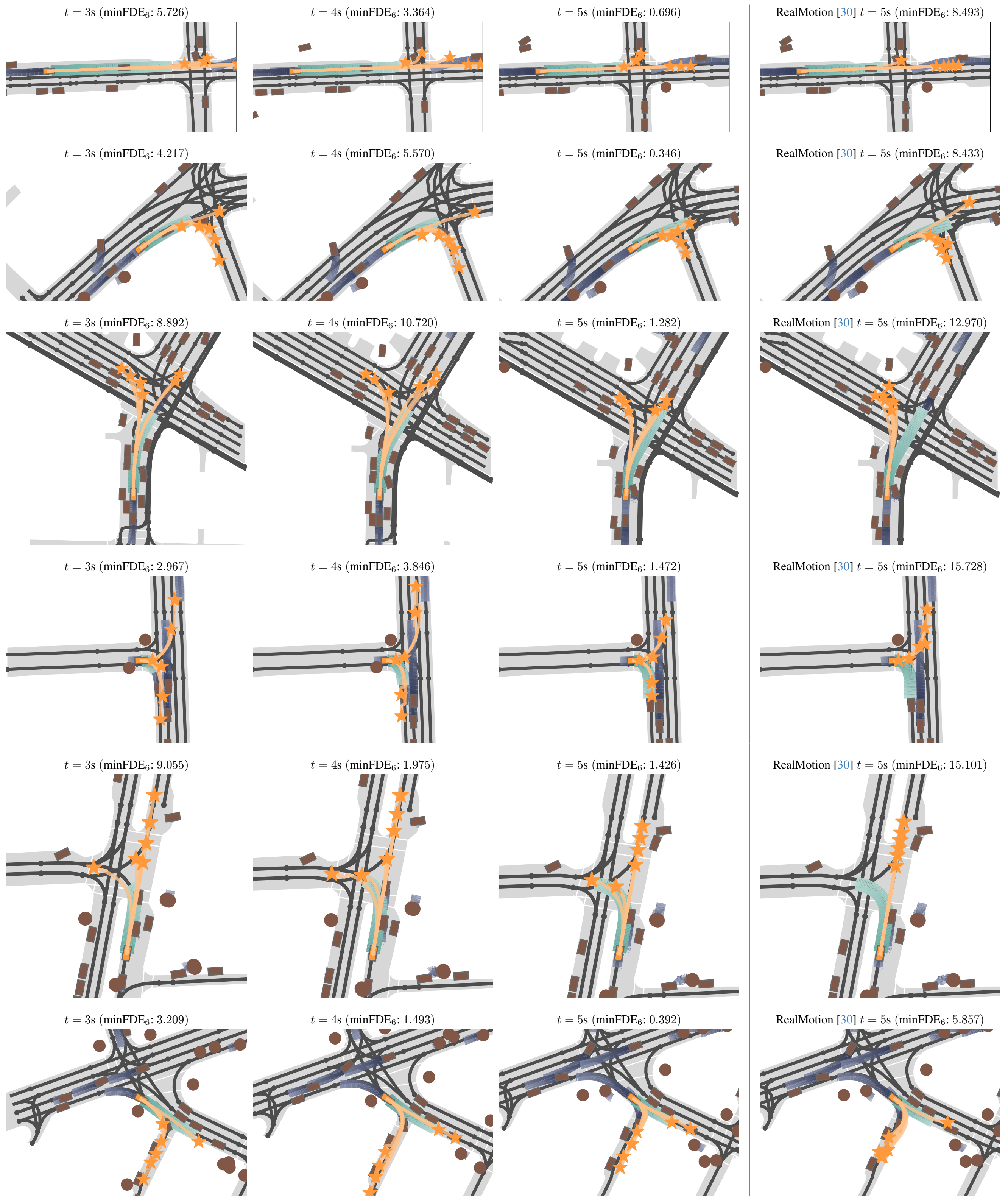
We present additional qualitative results on scenarios from the Argoverse 2 validation set in Figure 8 and Figure 9.

E.7. Failure Cases

We present failure cases, where our approach fails to correctly predict future trajectories in Figure 10. 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.

F. 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].



$t = 3$s (minFDE$_6$: 5.726)
$t = 4$s (minFDE$_6$: 3.364)
$t = 5$s (minFDE$_6$: 0.696)
RealMotion [30] $t = 5$s (minFDE$_6$: 8.493)
$t = 3$s (minFDE$_6$: 4.217)
$t = 4$s (minFDE$_6$: 5.570)
$t = 5$s (minFDE$_6$: 0.346)
RealMotion [30] $t = 5$s (minFDE$_6$: 8.433)
$t = 3$s (minFDE$_6$: 8.892)
$t = 4$s (minFDE$_6$: 10.720)
$t = 5$s (minFDE$_6$: 1.282)
RealMotion [30] $t = 5$s (minFDE$_6$: 12.970)
$t = 3$s (minFDE$_6$: 2.967)
$t = 4$s (minFDE$_6$: 3.846)
$t = 5$s (minFDE$_6$: 1.472)
RealMotion [30] $t = 5$s (minFDE$_6$: 15.728)
$t = 3$s (minFDE$_6$: 9.055)
$t = 4$s (minFDE$_6$: 1.975)
$t = 5$s (minFDE$_6$: 1.426)
RealMotion [30] $t = 5$s (minFDE$_6$: 15.101)
$t = 3$s (minFDE$_6$: 3.209)
$t = 4$s (minFDE$_6$: 1.493)
$t = 5$s (minFDE$_6$: 0.392)
RealMotion [30] $t = 5$s (minFDE$_6$: 5.857)

Figure 8. 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 [30] in the streaming setting.

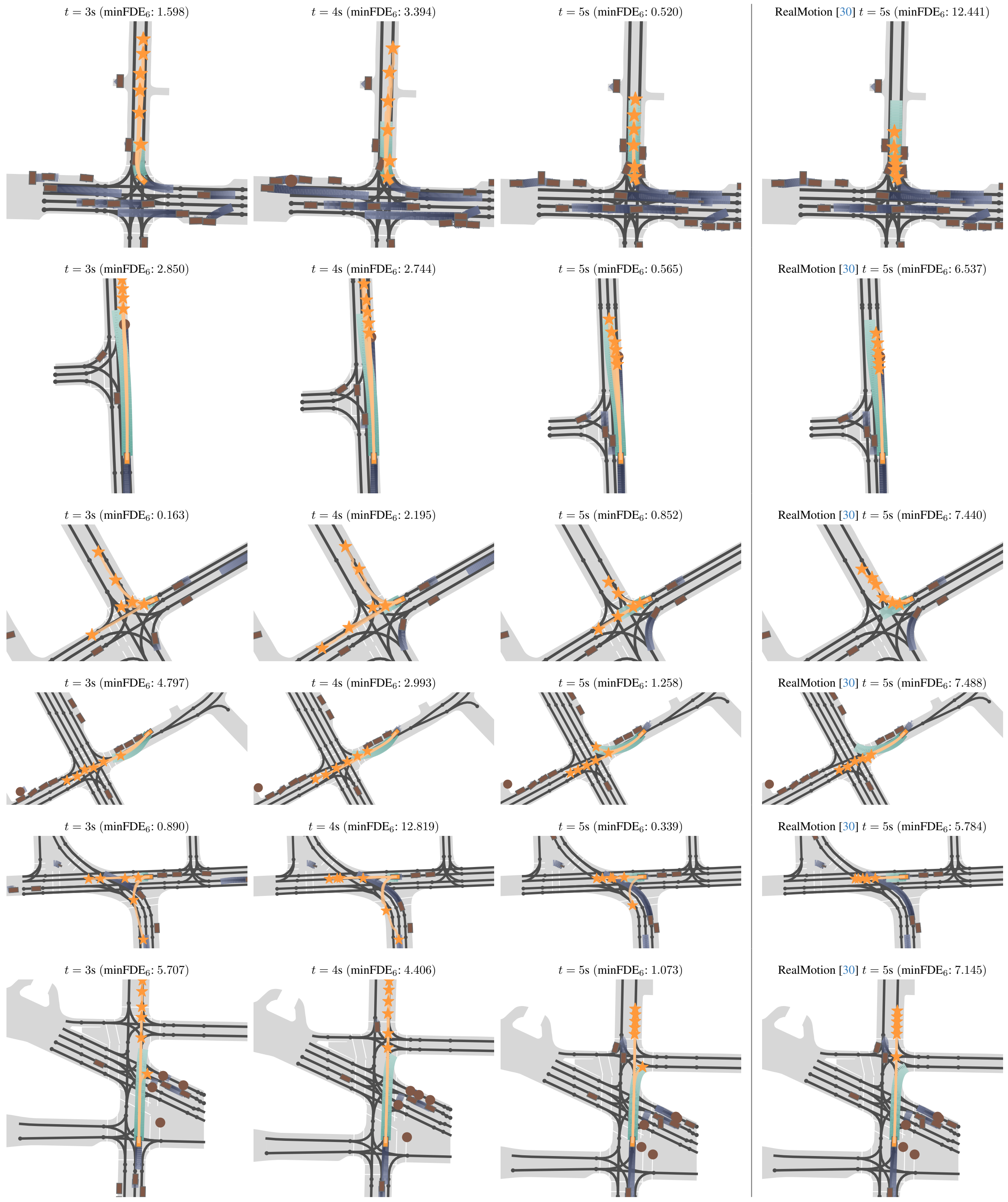

t = 3s (minFDE$_6$: 1.598)
t = 4s (minFDE$_6$: 3.394)
t = 5s (minFDE$_6$: 0.520)
RealMotion [30] t = 5s (minFDE$_6$: 12.441)
t = 3s (minFDE$_6$: 2.850)
t = 4s (minFDE$_6$: 2.744)
t = 5s (minFDE$_6$: 0.565)
RealMotion [30] t = 5s (minFDE$_6$: 6.537)
t = 3s (minFDE$_6$: 0.163)
t = 4s (minFDE$_6$: 2.195)
t = 5s (minFDE$_6$: 0.852)
RealMotion [30] t = 5s (minFDE$_6$: 7.440)
t = 3s (minFDE$_6$: 4.797)
t = 4s (minFDE$_6$: 2.993)
t = 5s (minFDE$_6$: 1.258)
RealMotion [30] t = 5s (minFDE$_6$: 7.488)
t = 3s (minFDE$_6$: 0.890)
t = 4s (minFDE$_6$: 12.819)
t = 5s (minFDE$_6$: 0.339)
RealMotion [30] t = 5s (minFDE$_6$: 5.784)
t = 3s (minFDE$_6$: 5.707)
t = 4s (minFDE$_6$: 4.406)
t = 5s (minFDE$_6$: 1.073)
RealMotion [30] t = 5s (minFDE$_6$: 7.145)

Figure 9. 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 [30] in the streaming setting.

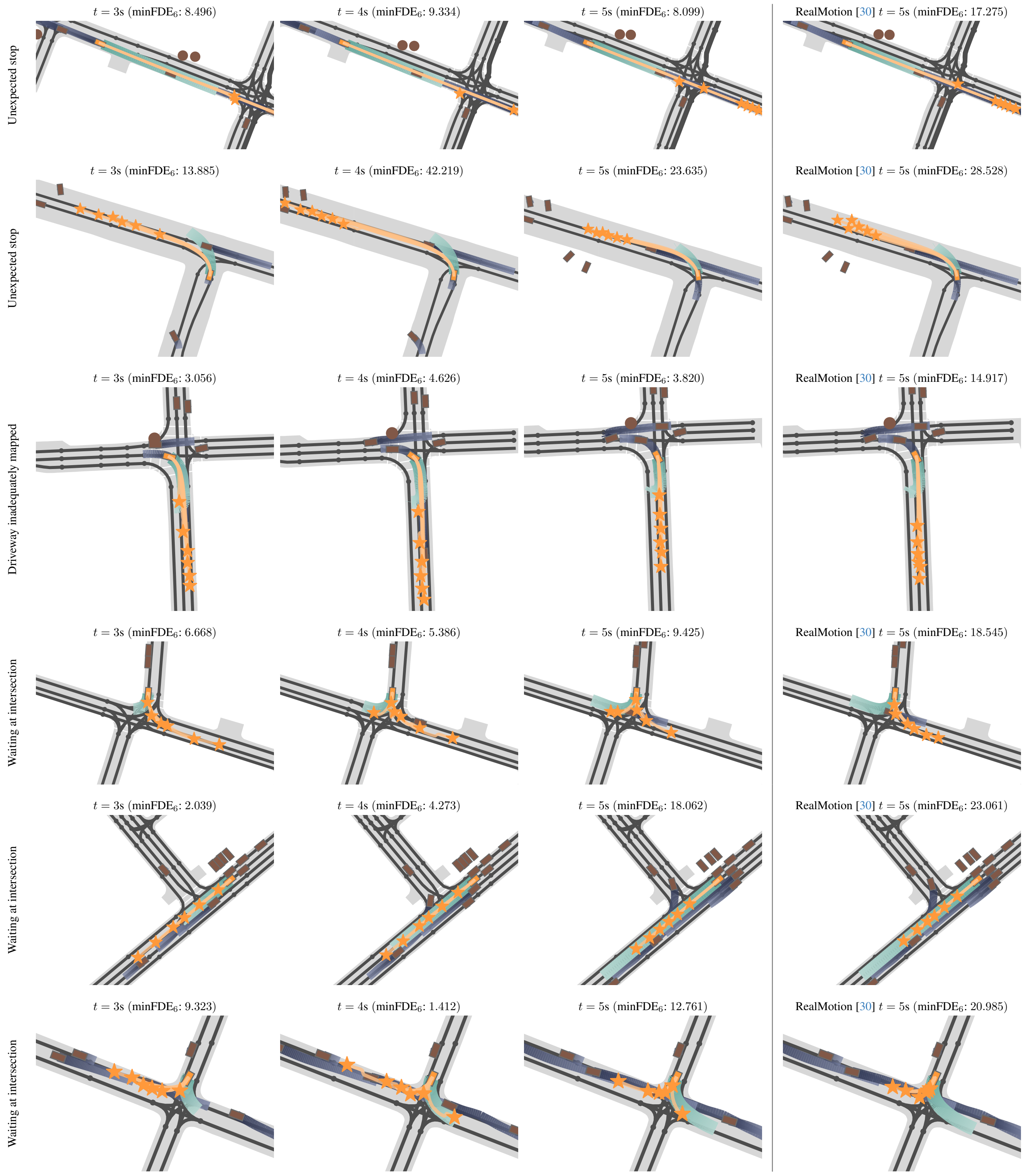


Figure 10. Failure cases of our approach on scenarios from the Argoverse 2 validation set. We show the **predictions** of our streaming-based method at $t \in \{3, 4, 5\}$s. The visualizations also show **ground truth future**, **agent histories**, and **neighboring agents**. The right column shows the final predictions at $t = 5$s for using RealMotion [30] 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.