Edge-Deployed Streaming Real-Time Trajectory Prediction

Alexander Prutsch[1*], Erik Bonek[2, 3], David Schinagl[1], Matthias Wess[2,3], Horst Possegger[1,3]

[1*]Institute of Visual Computing, Graz University of Technology, Graz, Austria.
[2]Institute of Computer Technology, TU Wien, Vienna, Austria.
[3]Christian Doppler Laboratory for Embedded Machine Learning.

*Corresponding author(s). E-mail(s): alexander.prutsch@tugraz.at;
Contributing authors: e12118550@student.tuwien.ac.at; david.schinagl@tugraz.at;
matthias.wess@tuwien.ac.at; possegger@tugraz.at;

Abstract

Future trajectories of neighboring traffic agents heavily influence the path planning and decision-making of autonomous vehicles. While traditional trajectory forecasting focuses on snapshot-based predictions that treat scenarios independently, real-world deployment demands continuous, real-time data stream processing with low latency and temporal consistency. To address these requirements, we introduce a lightweight, streaming-based trajectory forecasting framework that propagates temporal context across successive timesteps. By utilizing the predicted endpoints of previous forecasts as structural anchors, our approach efficiently guides a scene encoder to extract highly targeted context encodings without requiring multi-stage refinement or segment-wise decoding. To ensure deterministic execution bounds on real-world robotics hardware, we enforce static token boundaries via proximity-based sampling to satisfy edge-compilation constraints. Evaluated on the Argoverse 2 multi-agent and single-agent benchmarks, our method achieves state-of-the-art streaming accuracy while requiring fewer computational resources than snapshot baselines. Furthermore, hardware deployment tests on an embedded NVIDIA Jetson Orin Nano platform demonstrate that our model maintains an inference frequency exceeding 10 Hz while preserving a highly favorable latency-accuracy trade-off, proving its viability for deterministic, low-power, real-time execution in dense, continuous traffic environments.

Keywords: Motion Forecast, Autonomous Driving, Edge-Deployment, Trajectory Prediction

1 Introduction

Anticipating the behavior of other agents enables self-driving cars to plan their maneuvers proactively and respond appropriately to the predicted movements of surrounding traffic participants. This strong influence on decision-making and path planning makes trajectory prediction essential for efficient and reliable navigation in interaction with other road users. 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.

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. Initially, the focal agent position provides a strong prior for relevant context elements. However, as illustrated in Figure 1, scene elements along the future path play a crucial role to dynamically adjust the predictions: For example, when predicting a turn, a potentially crossing pedestrian on the sidewalk must be considered.

Recent trajectory prediction approaches commonly use refinement schemes [1–3] to incorporate this knowledge: Initial coarse future trajectories are iteratively refined to enhance prediction accuracy. For instance, SmartRefine [3] proposes an adaptive multi-stage refinement scheme with scene context re-encoding to improve predictions of the strong baseline model QCNet [2]. Methods that use such refinement iterations achieve highly accurate results on large-scale benchmarks like Argoverse 2 (AV2) [4]. 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.

The majority of existing trajectory prediction approaches focuses on processing and evaluating snapshots of individual scenarios in standard benchmark settings [4, 5]. Such a snapshot-based treatment, however, is in sharp contrast to real-world deployment, where trajectory prediction systems for autonomous vehicles operate in a continuous environment. In real-world scenarios, we must ensure consistency across successive predictions, which is essential for integrating the predictions into downstream decision-making systems. Additionally, continuous streaming-based processing also offers the advantage of exploiting temporal context to improve the accuracy of future motion estimates. Recently, pioneering streaming-based approaches [6, 7] laid the foundation for addressing the challenges of continuous trajectory prediction. For example, RealMotion [7] demonstrates that a simple backbone architecture combined with streaming mechanisms achieves promising results on the AV2 dataset [4]. 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

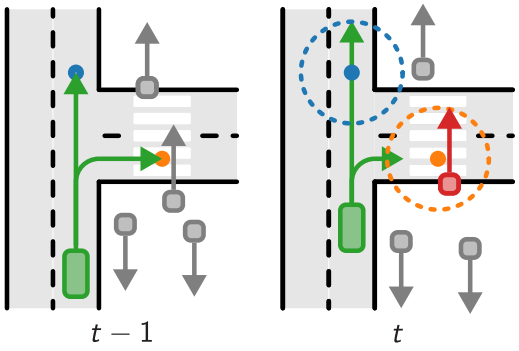


Fig. 1: Traffic scenes are constantly changing, which is often neglected in trajectory prediction methods operating on snapshots rather than in a continuous setting. For example, for the trajectory prediction of the turning process of the focal agent (green), a possible interaction with a pedestrian (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.

endpoint-aware modeling (SEAM) [8][1] 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 [4], across both single- and multi-agent benchmarks, demonstrate that our streaming mechanism provides significant advantages over previously proposed streaming techniques [7]. We show that our approach outperforms more complex and resource-demanding models, while also addressing practical considerations such as reducing temporal fluctuations across consecutive predictions for the same agent. We further extend our analysis towards practical real-world operation. To bridge the gap between theoretical streaming frameworks and physical deployment constraints, we adapt our architecture to enforce deterministic execution bounds through a static, proximity-based token ceiling. Beyond standard desktop evaluations, we validate the operational feasibility of our framework by deploying it directly

[1] A preliminary version of this work appeared in the Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) 2026.

onto embedded robotics hardware, exploring the multi-dimensional latency-accuracy trade-offs introduced under rigid static compilation constraints. Ultimately, our integrated framework combines high accuracy with minimal, deterministic 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.
- We establish a hardware-compatible serialization strategy that enforces static token limits via a proximity-based sampling heuristic, satisfying strict embedded edge-compilation graph requirements with minimal preprocessing overhead.
- We provide a comprehensive hardware evaluation by compiling our architecture via ONNX and deploying it on an NVIDIA Jetson Orin Nano platform, mapping out the multi-dimensional latency, accuracy, and compute-budget trade-offs.
- 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.

2 Related Work

Trajectory prediction methods typically take agent states and map information as input. Map data is predominantly modeled as vectorized data [9, 10] and encoded by PointNet-like architectures [11], while agent time series information is commonly processed by recurrent architectures [10, 12], attention-based modules [7, 13–15] or state-space models [16]. To capture relationships between agents and map elements, recent methods utilize graph neural networks [9, 17–21] or attention-based encoders [1, 13, 22–26]. Trajectory decoding is performed using either multi-layer perceptrons (MLPs) [7, 13] or more advanced cross-attention-based designs [1, 16, 27].

2.1 Refinement Modules for Trajectory Prediction

Iterative refinement is a common strategy in state-of-the-art trajectory prediction approaches [1–3, 26, 28]. Long-horizon benchmarks like Argoverse 2 [4] and WOMD [5] 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 [1, 3]. Initial predictions provide strong guidance for identifying key scene elements which can be used to iteratively refine the trajectory hypotheses.

MTR [1] and its follow-up works [26, 27, 29] iteratively refine trajectory predictions, using dynamic intention queries for aggregating scene context during decoding. R-Pred [30] 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 [2] 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 [3] 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 [16] 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 [16] and SmartRefine [3] 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*. [6] 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 [7] shows that incorporating context and predictions from previous frames enables accurate results even with a simple backbone [13]. 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 [16] also integrates both mechanisms into their approach. HPNet [21] includes the exploration of a basic historical prediction attention module on the comparatively simple AV1 dataset [31]. QCNet [2] 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 [6, 7] and leverage the propagation of agent-centric scene context across timesteps [7] 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 [1, 3], 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 [1, 3], 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 [6, 7], 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. As illustrated in 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 SEAM is a 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.

In the following, we first 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 [7]. We present our extension to the multi-agent setting in Section 3.6, while Section 3.7 summarizes the architectural optimizations and deployment strategies required to deploy SEAM on edge hardware.

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 [7, 13, 15, 16] 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

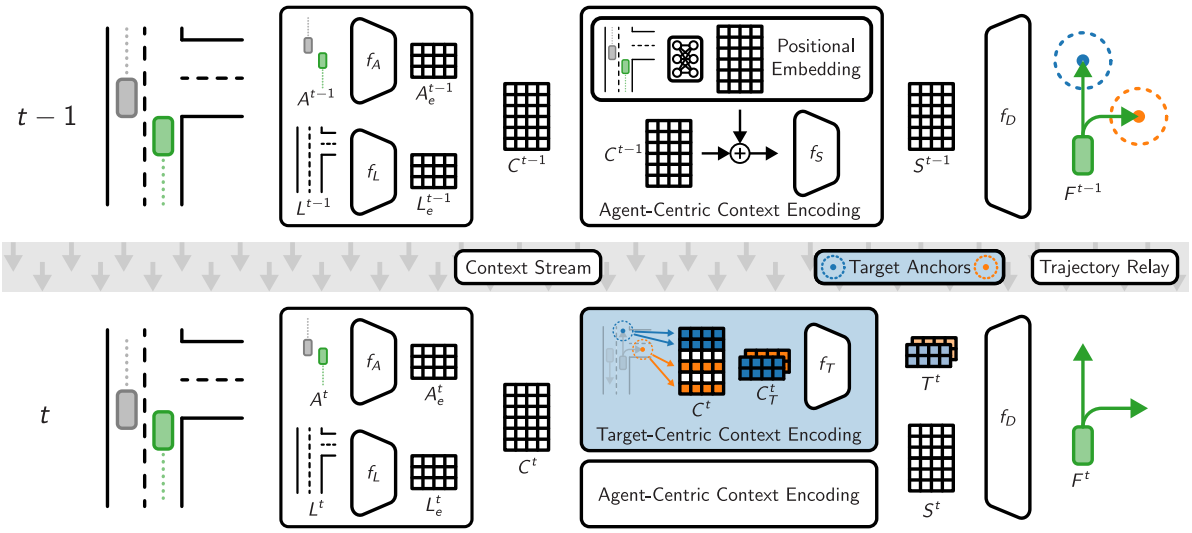


Fig. 2: Overview of our streaming-based trajectory prediction architecture SEAM. We assume that for frame $t-1$ no previous prediction exists for our focal agent. Thus, only a standard trajectory prediction model pass is executed. In the next frame t, we incorporate the endpoints from the previous predictions 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.

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 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 [13]. 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 [14, 15] 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 [32] 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 [1, 7, 13, 15, 16, 27] of using a mini-PointNet-like [11] 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 [13] to the scene tokens. A common way for encoding relations among map elements and agents is self-attention across scene tokens [3, 13–16]. 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, *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

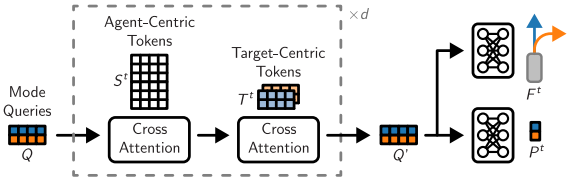


Fig. 3: Architecture of our novel dual-context attention decoder, leveraging both agent-centric S^t and target-centric T^t features.

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.

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 [33] 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, inspired by [7], to model the streaming setting through information flow from the previous frame $t-1$. We follow the setup of [7] for both mechanisms to enable a fair comparison: To account for focal agent motion between $t-1$ and t, we apply motion-aware layer normalization (MLN) [34] 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 $P_w \in \mathbb{R}^K$.

3.7 Model Adaptations for Edge Deployment

To assess the latency of our approach on resource-constrained hardware, we exported the model to the ONNX format[2] (Opset version 17) for deployment on an embedded robotics platform. Prior to conversion into a TensorRT engine, the ONNX model was simplified using the ONNX Simplifier[3]. Specifically, we utilized an NVIDIA Jetson AGX Orin Developer Kit configured in Jetson Orin Nano emulation mode. While our standard PyTorch [35] implementation[4] natively supports the dynamic allocation of input tensor shapes during runtime, ONNX compilation generally expects static input dimensions in order to optimize the hardware execution graphs.

This requirement conflicts with the standard trajectory prediction modeling approach, where typically all map elements and neighboring agents within a predefined region of interest (ROI) are sampled. This naturally yields a varying number of agents N_a and lane segments N_l per scene, and therefore varying model inputs: agent tensor $A \in \mathbb{R}^{N_a \times T_h \times D_a}$ and lane data tensor $L \in \mathbb{R}^{N_l \times P_l \times D_l}$. To satisfy the strict input constraints of the embedded hardware platform, we establish a fixed maximum budget for agent and lane tokens using a simple, deterministic distance heuristic. This straightforward method prioritizes the map elements and actors closest to the focal agent. Although more sophisticated selection mechanisms could be explored, such as reachable path analysis or target-driven intent filtering, our proximity-based strategy balances robust effectiveness with minimal preprocessing overhead.

Enforcing fixed ceilings on N_a and N_l guarantees a predictable compute budget, but it inherently introduces a trade-off in spatial coverage. Specifically, it can lead to truncated contexts or reduced spatial awareness in highly dense traffic scenarios compared to the unconstrained baseline. Crucially, treating N_a and N_l as fixed hyperparameters presents a direct opportunity to rigorously analyze the latency-accuracy profile of our model. Restricting the number of input tokens significantly minimizes computational overhead and runtime latency on the Orin platform. Conversely, excessive token reduction can degrade predictive performance—particularly for long-horizon forecasts or in cluttered environments—where aggressive downsampling risks omitting vital contextual cues.

The same design principle must also be applied during feature streaming to guarantee a static input size for our target-centric context encoder f_T. Specifically, we predefine the maximum number of scene tokens N_c', encompassing both agents and lanes, within the target-centric feature set. In practice, N_c' directly correlates with the scale of the chosen region of interest (ROI) used for collecting target-centric features. For our experiments, we use a conservative upper-bound baseline as the default setting, defining the token ceiling as $N_c' = N_a + N_l$. This heuristic ensures that, even in the worst-case scenario, all tokens from the initial agent and lane sets can be ingested by the context encoder without truncation. However, N_c' can also be tuned independently as a separate hyperparameter for further latency optimization.

4 Experiments

We present experimental results on the single-agent and multi-agent benchmarks of the Argoverse 2 (AV2) [4] dataset, along with a latency analysis to assess the

[2] https://onnx.ai
[3] https://github.com/onnxsim/onnxsim
[4] https://github.com/a-pru/seam

practical applicability of our approach. Additionally, we provide an ablation study for our endpoint-aware modeling on both AV2 and Argoverse (AV1) [31], with further results included in the appendix.

4.1 Experimental Settings

Dataset: We evaluate our model on the Argoverse 2 (AV2) [4] 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 [5] provides only 1 s of history and nuScenes [36] offers 2 s but at only 2 Hz. Further reducing these 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 [6, 7, 16]. 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) [31] which features 2 s history and considers a future of 3 s.

Metrics: We evaluate our approach using the standard AV2 [4] 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: We follow [7, 16] to enable streaming processing using existing benchmarks. Figure 4 compares the standard snapshot-based trajectory prediction paradigm to the streaming-based processing. Both paradigms use the same scenario splits to separate training and validation data, ensuring a fair comparison. In particular, the historical model input T_h is limited to a fraction of the available history. We then operate using a sliding window and conduct multiple model executions. For AV2 [4] we set the model input history to $h = 3$ s, which corresponds to

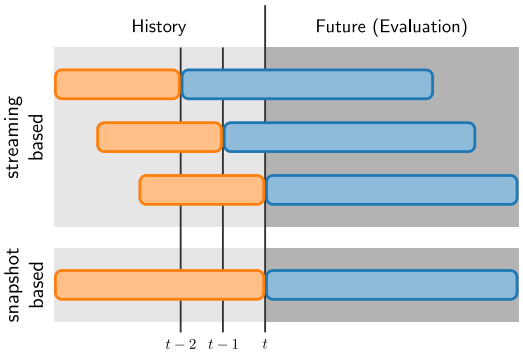


Fig. 4: Comparison between snapshot-based and streaming-based trajectory prediction paradigms. In this example, predictions from the third pass of the streaming model (at time t) can be directly evaluated against those from the snapshot-based approach.

$T_h = 30$ input samples, and 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. The final prediction is made at the same time point as in snapshot-based methods, allowing a fair comparison.

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. 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 futures. For our ablation on Argoverse 1 [31] we use a historical input of 1 s ($T_h = 10$) and execute three passes at $t \in \{1, 1.5, 2\}$ s.

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.

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 [16],

Method	Streaming	minADE_1	minFDE_1	MR_6	minADE_6	minFDE_6	brier-minFDE_6
SIMPL [37]	✗	2.03	5.50	0.19	0.72	1.43	2.05
HPTR [25]	✗	1.84	4.61	0.19	0.73	1.43	2.03
Forecast-MAE [13]	✗	1.74	4.36	0.17	0.71	1.39	2.03
MTR [1]	✗	1.74	4.39	0.15	0.73	1.44	1.98
EMP-D [15]	✗	1.75	4.35	0.17	0.71	1.37	1.98
GANet [28]	✗	1.77	4.48	0.17	0.72	1.34	1.96
ProIn [38]	✗	1.77	4.50	0.18	0.73	1.35	1.93
RealMotion [7] *	✓	1.65	4.12	0.15	0.67	1.29	1.93
QCNet [2]	✗	1.69	4.30	0.16	0.65	1.29	1.91
RealMotion [7] +Val*	✓	<u>1.59</u>	<u>3.93</u>	0.15	0.66	1.24	1.89
Tamba [39]	✗	1.66	4.24	0.17	0.64	1.24	1.89
ProphNet [40]	✗	1.80	4.74	0.18	0.68	1.33	1.88
MTR++ [27]	✗	1.64	4.12	<u>0.14</u>	0.71	1.37	1.88
DyMap [41]	✗	-	-	-	0.71	1.29	1.87
SmartRefine [3]	✗	1.65	4.17	0.15	<u>0.63</u>	<u>1.23</u>	<u>1.86</u>
DeMo [16]	✓	**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*. [6] is not included (their custom dataset prevents direct comparison). The best scores are set in bold, second best are underlined. *For RealMotion [7] 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.

Method	Streaming	avgMinADE_1	avgMinFDE_1	actorMR_6	avgMinADE_6	avgMinFDE_6	avgBrierMinFDE_6
FJMP [42]	✗	1.52	4.00	0.23	0.81	1.89	2.59
Forecast-MAE [13]	✗	1.30	3.33	0.19	0.69	1.55	2.24
RealMotion [7]	✓	1.14	2.97	0.18	0.62	1.32	2.01
DeMo [16]	✓	1.12	2.78	<u>0.16</u>	<u>0.58</u>	1.24	1.93
SEAM (Ours)	✓	<u>1.05</u>	<u>2.57</u>	<u>0.16</u>	0.62	<u>1.21</u>	<u>1.85</u>
QCNeXt [43]	✗	**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. The best scores are set in bold, second best are underlined.

achieves the best score on the main metric (brier-minFDE_6), outperforming recent work like Tamba [39] and even more resource-intensive methods, such as QCNet [2] and SmartRefine [3]. This strong performance without any refinement iterations highlights the effectiveness of our endpoint-aware modeling. We also outperform RealMotion [7], a prior streaming-based prediction method, while also achieving lower inference latency (section 4.4). DeMo [16] achieves slightly lower displacement errors since their decoupled decoding yields more diverse trajectories, but at a significantly higher latency (section 4.4). The small gap between minFDE_6 and 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.

4.3 Argoverse 2 Multi-Agent Results

Previous methods, *e.g.,* [3, 39], 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

GPU	Method	Streaming	Latency ($B = 1$) Offline	Latency ($B = 1$) Online	Latency ($B = 32$) Offline	Latency ($B = 32$) Online	Latency ($B = 64$) Offline	Latency ($B = 64$) Online	Model Parameters	AV2 Test Set brier-minFDE$_6$
V100	SmartRefine [3]	✗	>118 ms*		>680 ms*		-		8.0M	1.86
	QCNet [2]	✗	118 ms		680 ms		-		7.7M	1.91
	RealMotion [7]	✓	**86 ms**	**24 ms**	247 ms	88 ms	436 ms	150 ms	**2.9M**	1.93
	DeMo [16]	✓	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**
A10	RealMotion [7]	✓	51 ms	**13 ms**	255 ms	85 ms	512 ms	172 ms	**2.9M**	1.93
	DeMo [16]	✓	73 ms	18 ms	223 ms	71 ms	433 ms	143 ms	5.9M	**1.84**
	SEAM (Ours)	✓	**50 ms**	**13 ms**	**95 ms**	**39 ms**	**185 ms**	**65 ms**	4.6M	**1.84**

Table 3: Latency analysis for predicting B Argoverse 2 scenarios in the single-agent setting using different NVIDIA GPUs. *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. The best scores are set in bold, second best are underlined.

the same time. The results in Table 2 show that, in this setting, our SEAM outperforms previous streaming-based methods [7, 16] in the main metrics. The current best-performing method is the snapshot method QCNeXt [43], an extension of QCNet [2]. 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 Workstation GPU Latency Analysis

We utilize the official code implementations to measure the latencies of QCNet[5] [2], RealMotion[6] [7], and DeMo[7] [16]. 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.

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. Also on the newer GPU architecture (NVIDIA A10), our approach achieves favorable latency results, obtaining the lowest online and offline latency across all batch sizes. SEAM achieves minimal latency by using only a single decoder pass, unlike DeMo [16], QCNet [2] and SmartRefine [3], 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 [7] 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 [7] and DeMo [16]) does not provide details on their multi-agent implementations. Moreover, no official implementation is available for QCNeXt [43]. 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

[5] https://github.com/ZikangZhou/QCNet
[6] https://github.com/fudan-zvg/RealMotion
[7] https://github.com/fudan-zvg/DeMo

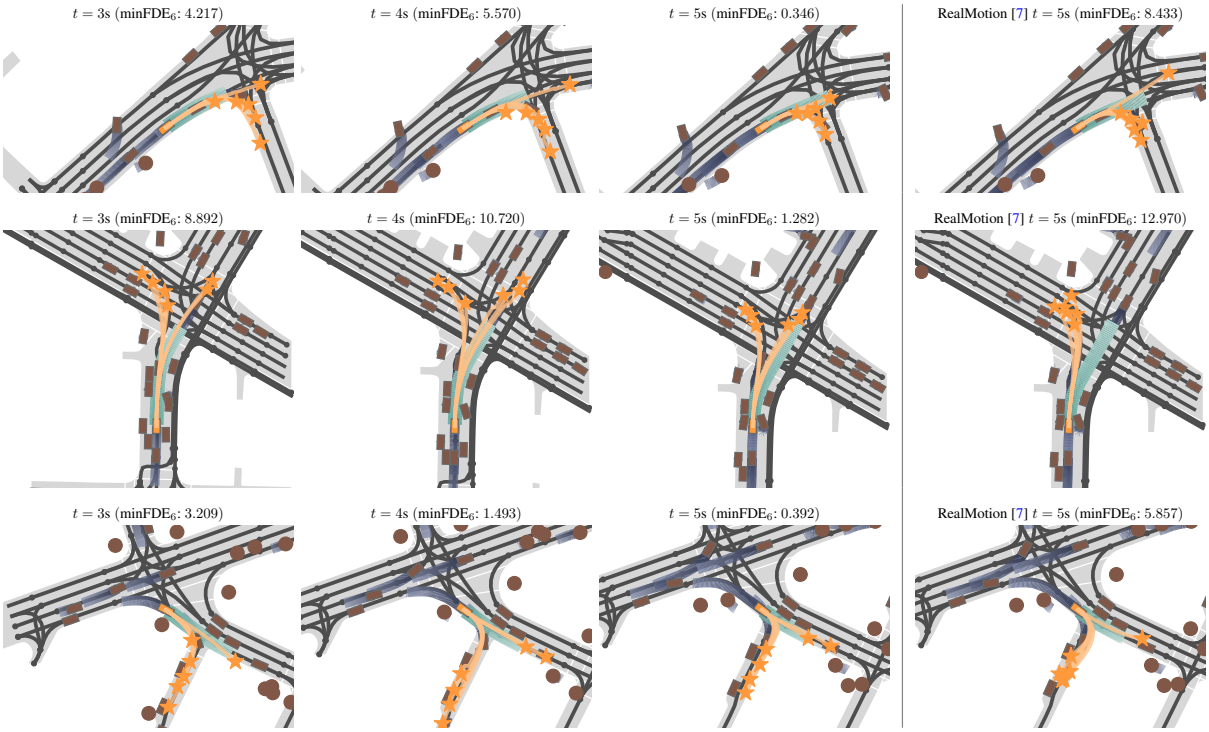


Fig. 5: Qualitative Argoverse 2 results. We show the **predictions** of our streaming-based method at $t \in \{3, 4, 5\}$s. The visualizations also show **ground truth future**, **agent histories**, and **neighboring agents**. The right column shows the final predictions at $t = 5$s for RealMotion [7] in the streaming-based setting.

DeMo and QCNet, highlighting the high efficiency of our approach.

4.5 Qualitative Results

To illustrate the evolving nature of streaming processing, we show the predictions at $t \in \{3, 4, 5\}$s in Figure 5. To demonstrate the benefits of our endpoint-aware modeling against previous work on streaming trajectory prediction, we compare against the predictions of RealMotion [7] at $t = 5$s. Common failure cases are demonstrated in Figure 6. These are mostly introduced by agent movements which cannot be anticipated at the prediction time, but also occur due to inadequate map data, *i.e.,* missing annotations for driveways that would indicate potential turn maneuvers.

Dataset	h	Stream	CS+TR [7]	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 [7] to our endpoint-aware modeling (EAM). The first line shows the traditional snapshot-based use. Best scores in bold.

4.6 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 [7]. The top row in each group presents the results of our backbone architecture

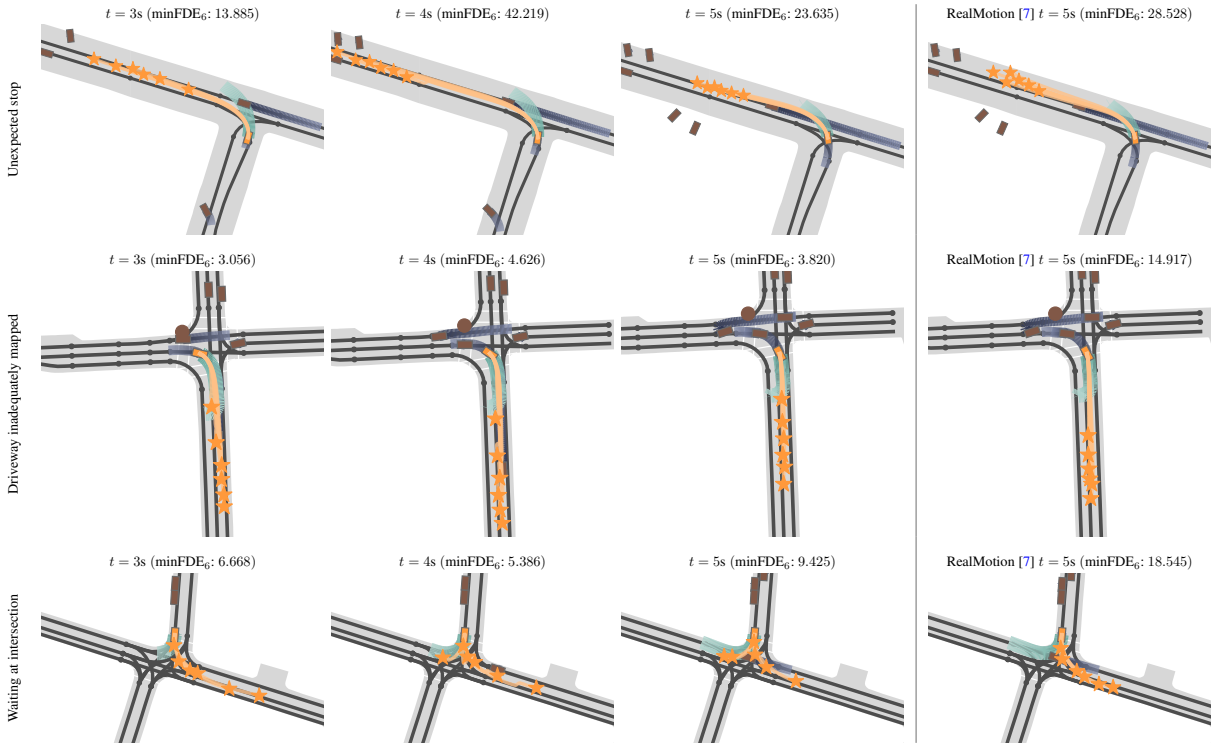


Fig. 6: Typical failure cases on scenarios from the Argoverse 2 validation set. We show the **predictions** of our streaming-based method at $t \in \{3, 4, 5\}$s, as well as the **ground truth future**, **agent histories**, and **neighboring agents**. The right column shows the final predictions at $t = 5$s for using RealMotion [7] in the streaming setting. In the first scenario, a vehicle stops for an unknown reason. In the second scenario, the vehicle begins turning into a driveway that is not modeled in the lane data. The third scenario depicts a vehicle waiting at an intersection where its future path is unclear.

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. Best scores in bold.

operating in the traditional snapshot-based prediction setting without any streaming mechanism. This 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 [7]. 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 agent-centric features, further

underscoring the importance of endpoint information in streaming processing.

5 Real-World Deployment Analysis on Embedded Hardware

To evaluate the practical viability and real-time operational limits of our approach, we analyze input scene complexity and its direct impact on computational scaling. Subsequently, we deploy our model onto an NVIDIA Jetson Orin Nano (8 GB) embedded platform to quantify the trade-offs between inference latency, precision, and accuracy.

5.1 Scene Complexity Analysis

Standard spatial region-based sampling with a 150 m radius yields highly variable scene scales depending on traffic density. To quantify this variability, Figure 7 provides a statistical analysis of the number of input agents N_a and lane segments N_l across the Argoverse 2 [4] validation set. The distribution indicates that for the vast majority of scenarios, a constrained token budget is sufficient to encapsulate the entire meaningful context. The median (50th percentile) counts are $N_a = 20$ and $N_l = 62$, while the 95th percentiles reach $N_a = 41$ and $N_l = 125$, respectively. Crucially, the long-tail nature of the distribution implies that only a small fraction of scenes actually demand large token counts. To better understand this phenomenon, Figure 8 visualizes the two worst-case outlier examples with (1) the highest density of neighboring agents and (2) the maximum number of lane segments within the dataset. These examples qualitatively demonstrate that relying on large input sizes introduces substantial computational redundancy. For instance, in highly crowded scenes, only a small subset of nearby actors actively influence the ego-vehicle's trajectory, while distant pedestrians remain irrelevant. Similarly, distance-based map sampling frequently includes peripheral road structures, such as adjacent highway dividers or parallel access roads, that are topologically unreachable by the focal agent. These observations motivate the empirical latency-accuracy study in the following sections, where we apply a proximity-based truncation strategy to demonstrate that capping token budgets filters out non-essential context without significantly sacrificing critical behavioral cues.

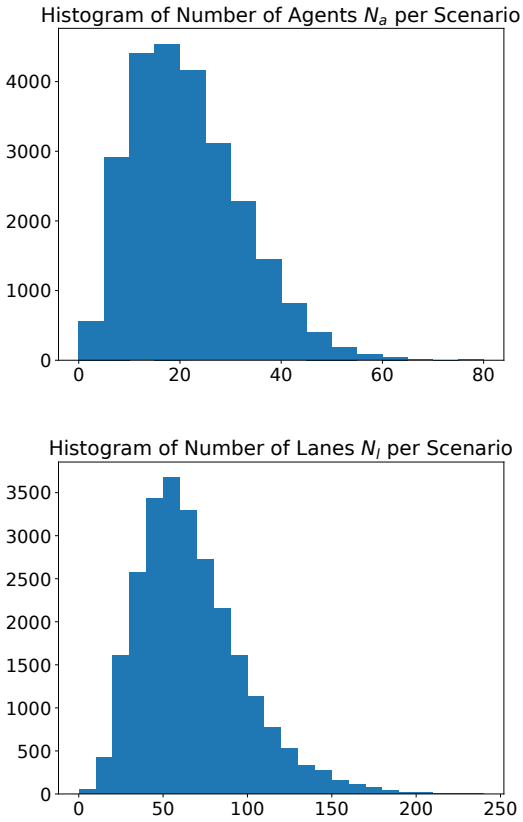


Fig. 7: Analysis of scene elements within the standard circular region of interest (150 m) across all 25k scenarios in the Argoverse 2 validation set.

5.2 Experimental Settings

In our hardware-centric evaluation, we analyze the relationships between inference latency, memory footprint, and power consumption relative to prediction accuracy by systematically varying the input token budgets, N_a and N_l. Our analysis focuses specifically on the single-agent forecasting setting, where scene complexity directly affects the cost of each focal agent prediction. Because our multi-agent framework decouples processing by computing initial marginal predictions prior to applying a global consistency module, scene complexity primarily bottlenecks these initial per-focal-agent operations. Crucially, our global consistency module scales solely with the number of predicted focal agents and remains constant with respect to the environmental context. The number of relevant focal agents represents an operational requirement defined

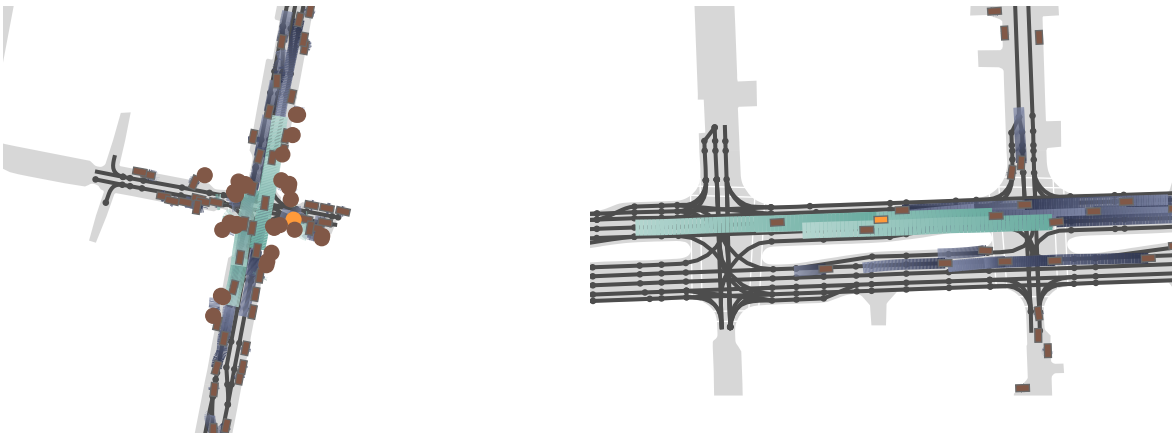


Fig. 8: Scenarios from the Argoverse 2 validation set featuring maximum scene complexity. Left: The scene with the highest agent count, driven by a high density of pedestrians. Right: The scene with the most lane segments, featuring adjacent multi-lane roads.

by the downstream motion planning module, whereas N_a and N_l serve as direct, internal tuning hyperparameters of the motion forecasting model itself. For all benchmarking experiments, we fix the batch size to 32 agents, simulating concurrent predictions for 32 unique focal actors, where each actor extracts context from its N_a closest neighbors and N_l closest lane segments. To provide a comprehensive operational profile, we benchmark the model across two distinct hardware power configurations of the Jetson Orin Nano, namely the 7 W and 15 W modes. Furthermore, to assess how mixed-precision execution influences the latency-accuracy trade-off, we evaluate the system using TensorFloat-32 (`TF32`) and half-precision (`FP16`) formats alongside standard single-precision (`FP32`) execution.

5.3 Edge-Deployment Experiments

Building on our complexity analysis and experimental setup, we present empirical results of our hardware evaluation. We analyze how varying token budgets, numerical precisions, and power caps influence the balance between execution speed and predictive accuracy to identify an optimal configuration for real-time systems.

Impact of Scene Complexity on Latency: As a foundational stress test, Figure 9 isolates the impact of individual token counts on overall model latency under the 15 W power configuration. To achieve this, we simulate a worst-case traffic scenario by fixing $N_a = 70$ agents while sweeping the lane budget N_l, and conversely fixing $N_l = 200$ lanes while sweeping the agent budget N_a. These upper bounds correspond to the maxima observed across the AV2 [4] dataset. These experiments confirm that execution latency scales predictably with both token dimensions. When comparing numerical precisions, `TF32` introduces modest execution speedups over native `FP32` without requiring structural modifications. Crucially, `FP16` precision delivers a $3\times$ reduction in absolute latency and exhibits the most favorable computational scaling behavior as the input token density expands.

Latency vs. Accuracy Trade-Off: Figure 10 maps the latency-accuracy Pareto frontier across varying agent (left) and lane (right) token budgets. The empirical curves reveal that constraining lane tokens has a much more pronounced impact on prediction accuracy than restricting agent tokens. This indicates that incorporating a small, highly localized set of neighboring actors is sufficient for high-fidelity motion forecasting, while scaling beyond this threshold introduces diminishing returns.

Analyzing the precision formats under the 15 W hardware configuration reveals that mixed-precision optimization significantly shifts the trade-off curve. Specifically, `FP16` execution delivers an approximate $3\times$ speedup over the baseline `FP32` implementation, while any corresponding degradation in displacement error remains entirely negligible. Testing different input sizes from the AV2 dataset, the memory footprint varies from 2.4 GB ($N_a = 70$, $N_l = 10$, and `FP16`) to 3.6 GB

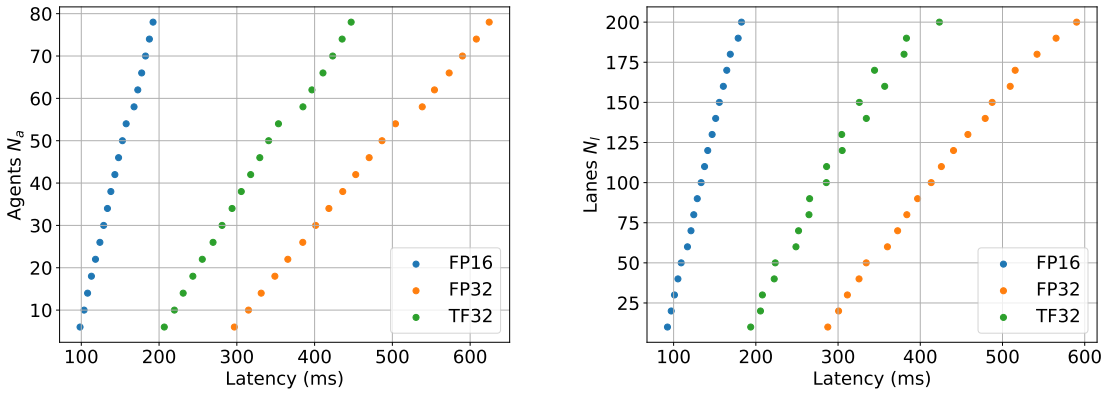


Fig. 9: Comparison of latency as a function of the number of agents N_a (left, with the number of lanes fixed to $N_l = 200$) and the number of lanes N_l (right, with the number of agents fixed to $N_a = 70$) across three data types.

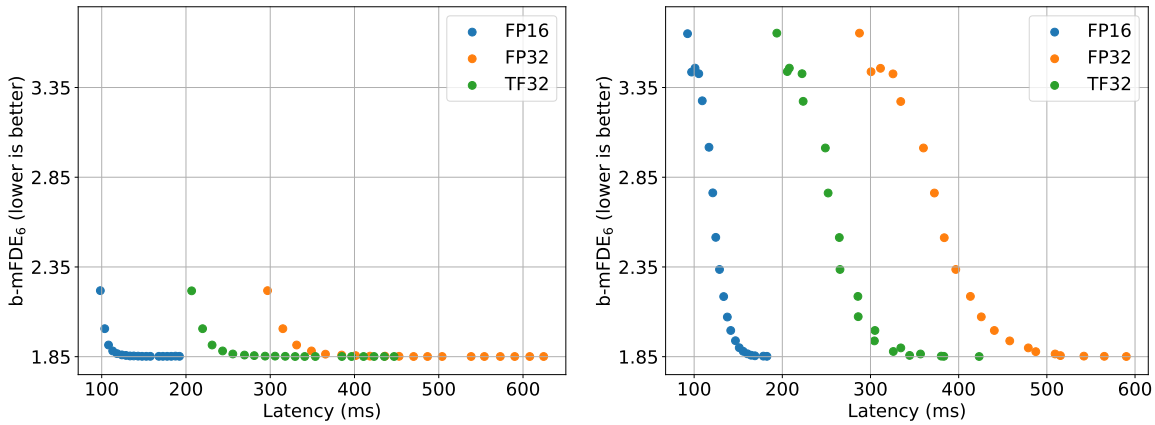


Fig. 10: Latency versus accuracy (brier-minFDE$_6$) under varying numbers of agents N_a (left, with the number of lanes fixed to $N_l = 200$) and lanes N_l (right, with the number of agents fixed to $N_a = 70$) across three data types. We sweep N_a from 6 to 78 and N_l from 10 to 200.

($N_a = 78$, $N_l = 200$, and `FP32`). These system-level findings demonstrate empirically that careful token serialization paired with half-precision compilation enables our streaming framework to meet strict deterministic real-time constraints on low-power edge hardware without sacrificing forecasting reliability.

Performance under Strict Power Constraints: Limiting the power consumption of the Jetson Orin Nano from 15 W down to the highly restrictive 7 W mode reduces the CPU clock frequency from 1510.4 MHz to 960 MHz and the GPU clock frequency from 624.75 MHz to 408 MHz. Additionally, the number of active texture processing clusters (TPCs) is reduced from 4 to 2. Across all evaluated numerical precision formats, this power cap induces an average latency increase of approximately $2.48\times$ across the sweep of agent N_a and lane N_l token budgets. Granular analysis reveals that this performance penalty scales non-uniformly across precisions: single-precision `FP32` execution exhibits a massive slowdown factor of $2.65\times$, whereas half-precision `FP16` is significantly more resilient, suffering a lower inflation factor of $2.36\times$. These dynamics provide compelling evidence that utilizing reduced numerical precision is essential when deploying streaming architectures onto heavily resource-constrained target platforms; `FP16` compilation not only yields direct inference speedups but also exhibits greater computational efficiency under

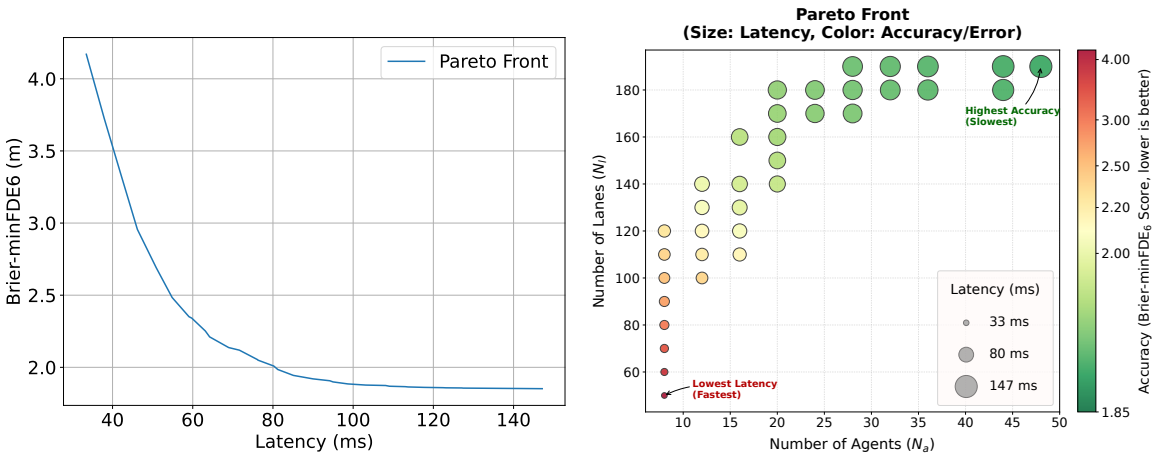


Fig. 11: Left: Latency vs. accuracy Pareto front when jointly varying the number of agents and lanes in scene inputs. Right: Both latency (marker size) and accuracy (color) as functions of varying input element counts.

strict hardware-enforced thermal and power limits, all while maintaining negligible accuracy degradation.

Latency-Accuracy Pareto Front: Figure 11 illustrates the latency-accuracy Pareto front obtained by sweeping the number of agents from 8 to 56 and the lane segments from 50 to 190. The left plot highlights that tolerating an initial latency increase from 35 ms to 80 ms yields a significant performance gain, whereas further expanding the input context to push latency over 140 ms results in strictly diminishing returns. This saturation demonstrates that while increasing input tokens initially improves predictions, it quickly becomes computationally inefficient, allowing practitioners to cleanly tune the latency-accuracy trade-off based on specific hardware capabilities. To further analyze this behavior, the right plot visualizes both latency and accuracy as functions of the agent and lane budgets. These trends indicate that map context is the primary driver of prediction accuracy, whereas a moderate number of neighboring agents is sufficient to yield strong results. Increasing the number of agents beyond this point adds latency but provides diminishing performance returns.

Optimal Configuration and Practical Analysis: The deployment analysis on the NVIDIA Jetson platform demonstrates that SEAM is highly suitable for real-time applications on robotics hardware, even when constrained to a low-power Jetson Orin Nano module. Based on our isolated sweeps across the agent and lane hyperparameters (Figure 10) alongside the empirical scene complexity distributions (recall Figure 7), we conduct joint experiments to identify an optimal operational configuration for both N_a and N_l. By selecting a cap of $N_a = 20$ neighboring agents and $N_l = 155$ lane segments to predict a batch of $B = 32$ focal agents, our model achieves a real-time inference latency of 97 ms on the Jetson Orin Nano using half-precision (`FP16`) execution, while maintaining an average prediction accuracy of 1.286 m in terms of minFDE$_6$. Compared to the unconstrained baseline setup ($N_a = 70$, $N_l = 200$, and `FP32` compilation), this optimized configuration has an accuracy degradation of less than 4 cm (from a baseline minFDE$_6 = 1.250$ m) while reducing inference latency by nearly $6\times$ (down from 590 ms). The 97 ms execution time breaks down into approximately 50 ms for the individual encoders (f_A and f_L) processing the raw data, 15 ms for the agent- and target-centric context encoders (f_S and f_T), and 32 ms for decoding (f_D). Furthermore, by additionally limiting the target-centric context size to the mean number of tokens per region ($N_c' = 26$) within our optimized configuration, we achieve a further speedup to 75 ms with negligible accuracy loss (*i.e.,* < 1 cm).

These results underline that latency can be drastically reduced with minimal impact on downstream accuracy. When considering the tight timing constraints of an end-to-end vehicle stack (encompassing perception, trajectory prediction, motion planning, and vehicle control), minimizing prediction latency yields critical safety advantages. Specifically, it minimizes the spatiotemporal displacement between new sensor inputs and ego-motion planning, ensuring the

autonomous system commands actions based on recent data rather than stale predictions.

6 Conclusion

We propose a novel endpoint-aware modeling approach 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 the inference speed of highly iterative schemes [3, 27] and previous streaming-based methods [7, 16] 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. To validate our approach for practical real-world deployment, we also adapt our architecture to satisfy the static graph constraints of hardware compilation engines by enforcing fixed agent and lane token limits via proximity-based sampling. Comprehensive evaluations on an embedded NVIDIA Jetson Orin Nano platform demonstrate that our model maintains a highly favorable latency-accuracy Pareto frontier. Specifically, optimizing the pipeline using half-precision (`FP16`) execution yields a $3\times$ speedup over `FP32` with negligible accuracy degradation, and exhibits robust performance even when hardware power consumption is strictly capped at 7 W. These results indicate that our streaming endpoint-aware modeling architecture is a practical path toward accurate motion forecasting on edge devices.

Acknowledgements. 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. This work was also partially supported by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association.

Appendix A Evaluation 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 hypothesis 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.

Appendix B Additional Results

B.1 Streaming Methods on AV2 Validation Set

Table B1 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.

B.2 Robustness to Error Propagation

Table B2 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

Method	minADE_1	minFDE_1	MR_6	minADE_6	minFDE_6	brier-minFDE_6
RealMotion [7]	1.65	4.10	0.16	0.67	1.30	1.94
DeMo [16]	**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 B1: 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 B2: Robustness of our endpoint-aware modeling on the AV2 [4] 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.

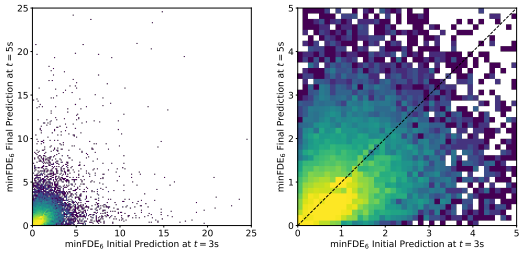


Fig. B1: Density plots of prediction errors. Both plots compare the first prediction done at $t = 3$ s (x-axis) versus the final prediction done at at $t = 5$ s (y-axis). The top plot shows that even when the initial error is large, the model can recover and correct its prediction. The bottom plot provides 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.

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 B1. 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.

B.3 Multi-Agent Ablation Study

We present an ablation study on extending single-agent approaches to the multi-agent setting in Table B3. 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 on multi-agent data allows the model to capture a broader range of behaviors (row 3), and incorporating an explicit global consistency module further improves performance, achieving the best results (row 4).

References

[1] Shi, S., Jiang, L., Dai, D., Schiele, B.: Motion Transformer with Global Intention Localization

Model	Training	GCM	actorMR$_6$	avgMinADE$_6$	avgMinFDE$_6$	avgBrierMinFDE$_6$
RealMotion [7]	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 B3: 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 [7] perform poorly in the multi-agent setting. Finetuning on the multi-agent dataset significantly improves performance, and incorporating our global consistency module (GCM) for scene-level scoring yields the best results.

and Local Movement Refinement. In: NeurIPS (2022)

[2] Zhou, Z., Wang, J., Li, Y.-H., Huang, Y.-K.: Query-Centric Trajectory Prediction. In: CVPR (2023)

[3] Zhou, Y., Shao, H., Wang, L., Waslander, S.L., Li, H., Liu, Y.: SmartRefine: A Scenario-Adaptive Refinement Framework for Efficient Motion Prediction. In: CVPR (2024)

[4] Wilson, B., Qi, W., Agarwal, T., Lambert, J., Singh, J., Khandelwal, S., Pan, B., Kumar, R., Hartnett, A., Pontes, J.K., Ramanan, D., Carr, P., Hays, J.: Argoverse 2: Next Generation Datasets for Self-driving Perception and Forecasting. In: NeurIPS Datasets and Benchmarks (2021)

[5] Ettinger, S., Cheng, S., Caine, B., Liu, C., Zhao, H., Pradhan, S., Chai, Y., Sapp, B., Qi, C.R., Zhou, Y., *et al.*: Large Scale Interactive Motion Forecasting for Autonomous Driving: The Waymo Open Motion Dataset. In: ICCV (2021)

[6] Pang, Z., Ramanan, D., Li, M., Wang, Y.-X.: Streaming Motion Forecasting for Autonomous Driving. In: IROS (2023)

[7] Song, N., Zhang, B., Zhu, X., Zhang, L.: Motion Forecasting in Continuous Driving. In: NeurIPS (2024)

[8] Prutsch, A., Schinagl, D., Possegger, H.: Streaming Real-Time Trajectory Prediction Using Endpoint-Aware Modeling. (2026)

[9] Gao, J., Sun, C., Zhao, H., Shen, Y., Anguelov, D., Li, C., Schmid, C.: VectorNet: Encoding HD Maps and Agent Dynamics From Vectorized Representation. In: CVPR (2020)

[10] Varadarajan, B., Hefny, A., Srivastava, A., Refaat, K.S., Nayakanti, N., Cornman, A., Chen, K., Douillard, B., Lam, C.P., Anguelov, D., *et al.*: MultiPath++: Efficient Information Fusion and Trajectory Aggregation for Behavior Prediction. In: ICRA (2022)

[11] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In: CVPR (2017)

[12] Mercat, J., Gilles, T., El Zoghby, N., Sandou, G., Beauvois, D., Gil, G.P.: Multi-Head Attention for Multi-Modal Joint Vehicle Motion Forecasting. In: ICRA (2020)

[13] Cheng, J., Mei, X., Liu, M.: Forecast-MAE: Self-supervised Pre-training for Motion Forecasting with Masked Autoencoders. In: ICCV (2023)

[14] Lan, Z., Jiang, Y., Mu, Y., Chen, C., Li, S.E., Zhao, H., Li, K.: SEPT: Towards Efficient Scene Representation Learning for Motion Prediction. In: ICLR (2023)

[15] Prutsch, A., Bischof, H., Possegger, H.: Efficient Motion Prediction: A Lightweight & Accurate Trajectory Prediction Model With Fast Training and Inference Speed. In: IROS (2024)

[16] Zhang, B., Song, N., Zhang, L.: DeMo: Decoupling Motion Forecasting into Directional Intentions and Dynamic States. In: NeurIPS (2024)

[17] Liang, M., Yang, B., Hu, R., Chen, Y., Liao, R., Feng, S., Urtasun, R.: Learning Lane Graph Representations for Motion Forecasting. In: ECCV (2020)

[18] Zeng, W., Liang, M., Liao, R., Urtasun, R.: LaneRCNN: Distributed Representations for Graph-Centric Motion Forecasting. In: IROS (2021)

[19] Jia, X., Wu, P., Chen, L., Liu, Y., Li, H., Yan, J.: HDGT: Heterogeneous Driving Graph Transformer for Multi-Agent Trajectory Prediction via Scene Encoding. IEEE TPAMI (2023)

[20] Cui, A., Casas, S., Wong, K., Suo, S., Urtasun, R.: GoRela: Go Relative for Viewpoint-Invariant Motion Forecasting. In: ICRA (2023)

[21] Tang, X., Kan, M., Shan, S., Ji, Z., Bai, J., Chen, X.: HPNet: Dynamic Trajectory Forecasting with Historical Prediction Attention. In: CVPR (2024)

[22] Liu, Y., Zhang, J., Fang, L., Jiang, Q., Zhou, B.: Multimodal Motion Prediction with Stacked Transformers. CVPR (2021)

[23] Ngiam, J., Caine, B., Vasudevan, V., Zhang, Z., Chiang, H.-T.L., Ling, J., Roelofs, R., Bewley, A., Liu, C., Venugopal, A., et al.: SceneTransformer: A Unified Architecture for Predicting Multiple Agent Trajectories. ICLR (2022)

[24] Nayakanti, N., Al-Rfou, R., Zhou, A., Goel, K., Refaat, K.S., Sapp, B.: Wayformer: Motion Forecasting via Simple & Efficient Attention Networks. In: ICRA (2023)

[25] Zhang, Z., Liniger, A., Sakaridis, C., Yu, F., Van Gool, L.: Real-Time Motion Prediction via Heterogeneous Polyline Transformer with Relative Pose Encoding. In: NeurIPS (2023)

[26] Gan, Y., Xiao, H., Zhao, Y., Zhang, E., Huang, Z., Ye, X., Ge, L.: MGTR: Multi-Granular Transformer for Motion Prediction with LiDAR. In: ICRA (2024)

[27] Shi, S., Jiang, L., Dai, D., Schiele, B.: MTR++: Multi-Agent Motion Prediction With Symmetric Scene Modeling and Guided Intention Querying. IEEE TPAMI (2024)

[28] Wang, M., Zhu, X., Yu, C., Li, W., Ma, Y., Jin, R., Ren, X., Ren, D., Wang, M., Yang, W.: GANet: Goal Area Network for Motion Forecasting. In: ICRA (2023)

[29] Demmler, T., Hartung, L., Tamke, A., Dang, T., Hegai, A., Haug, K., Mikelsons, L.: Dynamic Intent Queries for Motion Transformer-based Trajectory Prediction. In: IEEE IV (2025)

[30] Choi, S., Kim, J., Yun, J., Choi, J.W.: R-Pred: Two-Stage Motion Prediction Via Tube-Query Attention-Based Trajectory Refinement. In: ICCV (2023)

[31] Chang, M.-F., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., *et al.*: Argoverse: 3D Tracking and Forecasting with Rich Maps. In: CVPR (2019)

[32] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention Is All You Need. In: NeurIPS (2017)

[33] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-End Object Detection with Transformers. In: ECCV (2020)

[34] Wang, S., Liu, Y., Wang, T., Li, Y., Zhang, X.: Exploring Object-Centric Temporal Modeling for Efficient Multi-View 3D Object Detection. In: CVPR (2023)

[35] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019)

[36] Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: nuScenes: A Multimodal Dataset for Autonomous Driving. In: CVPR (2020)

[37] Zhang, L., Li, P., Liu, S., Shen, S.: SIMPL: A Simple and Efficient Multi-agent Motion Prediction Baseline for Autonomous Driving. RAL (2024)

[38] Dong, Y., Yuan, H., Liu, H., Jing, W., Li, F., Liu, H., Fan, B.: ProIn: Learning to Predict Trajectory Based on Progressive Interactions for Autonomous Driving. arXiv (2024)

[39] Huang, Y., Cheng, Y., Wang, K.: Trajectory Mamba: Efficient Attention-Mamba Forecasting Model Based on Selective SSM. In: CVPR (2025)

[40] Wang, X., Su, T., Da, F., Yang, X.: ProphNet: Efficient Agent-Centric Motion Forecasting with Anchor-Informed Proposals. In: CVPR (2023)

[41] Fan, B., Yuan, H., Dong, Y., Zhu, Z., Liu, H.: Bidirectional Agent-Map Interaction Feature Learning Leveraged by Map-Related Tasks for Trajectory Prediction in Autonomous Driving. IEEE Trans. Automation Science and Eng. (2025)

[42] Rowe, L., Ethier, M., Dykhne, E.-H., Czarnecki, K.: FJMP: Factorized Joint Multi-Agent Motion Prediction over Learned Directed Acyclic Interaction Graphs. In: CVPR (2023)

[43] Zhou, Z., Wen, Z., Wang, J., Li, Y.-H., Huang, Y.-K.: QCNeXt: A Next-Generation Framework For Joint Multi-Agent Trajectory Prediction. arXiv (2023)