SmartRefine: A Scenario-Adaptive Refinement Framework for Efficient Motion Prediction

Yang Zhou[1*] Hao Shao[1,2*] Letian Wang[3]
Steven L. Waslander[3] Hongsheng Li[2,4,5] Yu Liu[1,5] ✉

[1]SenseTime Research [2]CUHK MMLab [3]University of Toronto
[4]CPII under InnoHK [5]Shanghai Artificial Intelligence Laboratory

Abstract

Predicting the future motion of surrounding agents is essential for autonomous vehicles (AVs) to operate safely in dynamic, human-robot-mixed environments. Context information, such as road maps and surrounding agents' states, provides crucial geometric and semantic information for motion behavior prediction. To this end, recent works explore two-stage prediction frameworks where coarse trajectories are first proposed, and then used to select critical context information for trajectory refinement. However, they either incur a large amount of computation or bring limited improvement, if not both. In this paper, we introduce a novel scenario-adaptive refinement strategy, named SmartRefine, to refine prediction with minimal additional computation. Specifically, SmartRefine can comprehensively adapt refinement configurations based on each scenario's properties, and smartly chooses the number of refinement iterations by introducing a quality score to measure the prediction quality and remaining refinement potential of each scenario. SmartRefine is designed as a generic and flexible approach that can be seamlessly integrated into most state-of-the-art motion prediction models. Experiments on Argoverse (1 & 2) show that our method consistently improves the prediction accuracy of multiple state-of-the-art prediction models. Specifically, by adding SmartRefine to QCNet, we outperform all published ensemble-free works on the Argoverse 2 leaderboard (single agent track) at submission[1]. Comprehensive studies are also conducted to ablate design choices and explore the mechanism behind multi-iteration refinement. Codes are available at our webpage.

1. Introduction

Predicting the future motion of surrounding agents (*e.g.*, vehicle, cyclist, pedestrian) is crucial for autonomous driving frameworks [11, 22–24, 31] to safely and efficiently make decisions in a dynamic and human-robot-mixed environment. Context information, such as high-definition maps (HD maps) and surrounding agents' states, provides crucial geometric and semantic information for motion behavior, as agents' behaviors are highly dependent on the map topology and impacted by interaction with surrounding agents. For instance, vehicles usually move in drivable areas and follow the direction of lanes, and agents' interactive cues such as yielding would inform other agent's decision-making. As a result, recent motion prediction models are shown to significantly benefit from delicate context representation designs [2, 9, 13, 30] and context encodings [16, 21, 26]. However, complicated context encodings usually come at a high computational cost and high memory footprint. Since vehicles are high-speed robots and minimal delay could result in catastrophic accidents, recent advanced state-of-the-art methods could have low applicability due to limitations in mobile computation capacity and the hard real-time requirements of autonomous driving.

By contrast, human drivers can easily predict surrounding agents' future behaviors, even if they confront a daunting amount of context information. As implied by neuroscience, humans' efficient reasoning capabilities benefit from their *selective attention* mechanism [17, 19], which identifies compact context information critical to the task for efficient reasoning. Similarly, motion prediction models are shown to be able to produce high-quality predictions with only a few critical context elements provided, such as only giving the ground-truth future reference lanes and conducting prediction in the relative coordinates [28, 29, 35]. Therefore, if we can identify the critical context elements, and aggregate more information from these critical inputs to further refine the predictions, both the computational efficiency and prediction performance can be significantly improved.

While multiple refinement strategies have been proposed, the proper design of refinement is non-trivial. [36, 40] apply a prediction backbone to generate trajectory proposals, which are then used as anchor trajectories for refinement. While accuracy improvement is observed, the refinement still em-

* Equal contribution, ✉ Corresponding author.
[1] November 2023.

ploys all context information without selection, which leads to high computation costs. [5, 25] propose pooling and grouping methods to select context features via fixed manual rules and refine trajectory for fixed iterations. However, the mechanism of refinement is still under-studied, and only fixed refinement strategies are applied, where many iterations of refinement usually exchange a large amount of computation for limited performance improvement.

To this end, we therefore propose a scenario-adaptive refinement strategy, SmartRefine, which improves prediction accuracy with minimal additional computation for a wide variety of prediction models. Our key insight is that, while motion prediction models confront various driving scenarios, the prediction quality and refinement potential in different scenarios are not uniform: though some scenarios can benefit from intense refinement, some scenarios can be insensitive to refinement or even pushed away from ground truth due to over-refinement. Further, different scenarios may require different refinement configurations (*i.e.* how to select and encode context), while previous methods typically apply fixed refinement configurations to all scenarios (as shown in Table 1). To address these issues, the proposed SmartRefine can adapt the anchor/context selection and context encoding according to each scenario's properties, and can select the number of refinement iterations by introducing a quality score to measure the prediction quality and remaining refinement potential of each scenario. Thus, our method can allocate computation resources to those scenarios that require refinement, and terminate the refinement of the other scenarios to avoid worse prediction and wasted computation, achieving a better trade-off between accuracy and efficiency.

It is worth noting that SmartRefine is not only lightweight, but also designed as a general and flexible framework that is decoupled from the primary prediction model backbone, and only requires a generic interface to the model backbone (predicted trajectories and trajectory features). Therefore, SmartRefine can be easily integrated into most state-of-the-art motion prediction models. This distinguishes our method from previous refinement methods [5, 25, 36, 40] which are either computationally heavy, or highly coupled with a particular backbone, if not both.

To summarize, the contributions of this work are threefold:

- We introduce SmartRefine, a scenario-adaptive refinement method that considers comprehensive design choices and configurations for refinement and adapts them to each scenario, to effectively enhance prediction accuracy with limited additional computation.
- We propose a generic and flexible refinement framework, which can be easily integrated into most state-of-the-art motion prediction models to enhance performance. Codes are released to facilitate further research.
- We conduct extensive experiments on Argoverse and Argoverse 2 datasets (both the validation set and test set), where we evaluate SmartRefine by applying it to multiple state-of-the-art motion prediction models. We show that SmartRefine improves the accuracy of all considered motion prediction models with little additional computation. Specifically, by adding SmartRefine to QCNet [40], we outperform all published ensemble-free works on the Argoverse 2 leaderboard (single agent track) at the time of the paper submission.

	Refine	Context Selection/Encoding			Refinement Iteration		
		All	Fixed Strategy	Adaptive Strategy	Single	Multiple	Scenario Adaptive
TNT [38]	✗						
GoalNet [37]	✗						
GANet [32]	✗						
ProphNet [33]	✗						
DCMS [36]	✔	✗	✗	✗	✔	✗	✗
QCNet [40]	✔	✔	✗	✗	✔	✗	✗
R-Pred [5]	✔	✗	✔	✗	✔	✗	✗
MTR [25]	✔	✗	✔	✗	✗	✔	✗
SmartRefine (ours)	✔	✗	✗	✔	✗	✗	✔

Table 1. A comparison between the proposed SmartRefine framework and previous methods in terms of 1) whether refinement is conducted; 2) how the context selection and encoding is conducted; 3) how to determine the number of refinement iterations.

2. Related Work

2.1. Goal-Conditioned Trajectory Prediction

Trajectory prediction typically takes the road map, agent history states, and semantic type as inputs and outputs future agent states. To encode scene context (road map, surround agents) around the target agents, early works [2, 6] rasterize them into a bird-eye-view image and process it with convolutional neural networks. To encourage relationship reasoning and reduce computation, recent works adopt vector-based encoding schemes, where each scene context is represented as a vector, and encoded with permutation-invariant operators such as pooling [26], graph convolution [7], and attention mechanism [8, 12, 33]. Inspired by human hierarchical decision-making where the motion intention determines the specific trajectory, goal-conditional prediction, which first predicts or predefines goal candidates, and then predicts trajectory conditioned on them, is shown to be effective and has been widely adopted in state-of-the-art methods. Multipath [2] predefines a set of anchor trajectories by clustering and predicts offset for the trajectories. TNT [38] predicts goal points that offset from a lane centerline. GoalNet [37] uses lane segments as trajectory anchors. we predict the lane that a vehicle will pass in the future. FRM [18] predicts the occupancy of each waypoint on the lanes, and then predicts the fine-grained trajectory. GANet [32] proposes a goal area-based framework for predicting goal areas and fusing crucial distant map features. MTR [25] pre-defines a set of goal points as queries by clustering the trajectory data of each agent type, and adopts attention layers to aggregate context information based on the queries. Prophnet [33] uses trajectory proposals as learnable anchors to enable the attention layers to encode goal-oriented scene contexts. However, while these methods exploit goal-conditioned prediction, they only leverage goal-conditioned contexts once. In this paper, we utilize trajectory anchors as goals in an iterative manner, where the predicted trajectory at one iteration can benefit the next iteration by specifying more precise anchors as goals to extract more relevant map information.

2.2. Refinement: Two-Stage Trajectory Prediction

Inspired by refinement networks [1, 20] in computer vision, refinement strategies have been introduced to the trajectory prediction community. Trajectory refinement typically takes proposed trajectories in the first stage as inputs, and outputs the offset and probability for each proposed trajectory. DCMS [36] takes the output of the first stage as anchor trajectories and conducts refinement to predict the offset. QCNet [40] uses a small GRU to embed the proposed trajectories in the first stage and predict the offset of the trajectories by fusing the same scene context. R-Pred [5] proposes tube-query scene attention layers to refine based on local contexts, and interaction attention layers to refine based on agents' interactions. MTR [25] uses the predicted trajectory as anchors to retrieve contexts along the trajectory for refinement. The refinement is conducted for multiple iterations. However, existing methods typically employ a relatively large refinement network or utilize fixed refinement strategies (e.g. retrieval range, context encodings, and refinement iterations), which leads to sub-optimal performance in various scenarios. By contrast, SmartRefine can comprehensively adapt refinement configurations based on each scenario's properties, and smartly choose the number of refinement iterations. Besides, SmartRefine is designed as a lightweight and flexible approach that can be seamlessly integrated into state-of-the-art motion prediction. We will show through experiments how our method can refine prediction with minimal additional computation.

3. Method

SmartRefine is a scenario-adaptive refinement method to enhance the performance of motion prediction models with limited additional computation by adaptively iterating between retrieving critical context information and predicting more accurate trajectories. The overall structure is illustrated in Fig. 1 and Algorithm 1. In Sec. 3.1, we introduce the formulation of the problem. In Sec. 3.2, we elaborate on the proposed methodologies. In Sec. 3.3, we introduce the training details of our framework.

3.1. Problem formulation

Given the observed states of the target agent $\mathbf{s}_h = [s_{-T_h}, s_{-T_h+1}, \ldots, s_0]$ in the historic T_h steps, we aim at predicting its future states $\mathbf{s}_f = [s_1, s_2, \ldots, s_{T_f}]$ of T_f steps and associated probabilities $\mathbf{p}$. Naturally, the target agent will interact with the context $\mathbf{c} = (\mathbf{o}_h, \mathbf{m})$, including historic states of surrounding agents $\mathbf{o}_h$, and the HD map $\mathbf{m}$. For the HD map, we adopt the vectorized representation [9, 13], where each lane is defined as a sequence of points along its centerline. Each point is represented by the coordinates and semantic information. Thus typical motion prediction task is formulated as $(\mathbf{s}_f, \mathbf{p}) = f(\mathbf{s}_h, \mathbf{c})$, where f denotes the prediction model.

In this work, the introduced refinement strategy will slightly modifies problem formulation. As in Fig. 1, we first have a backbone model f_b generating initial trajectories $\mathbf{s}_f^0$ and trajectory features $\mathbf{h}_f^0$, as typical prediction methods $(\mathbf{s}_f^0, \mathbf{h}_f^0, \mathbf{p}) = f_b(\mathbf{s}_h, \mathbf{c})$. The initial predicted trajectory $\mathbf{s}_f^0$ is then used to select anchors $\mathbf{a}^0$ and retrieve critical context elements $\mathbf{c}^0$, which are then passed along with the trajectory features $\mathbf{h}_f^0$ into the refinement model f_r for trajectory refinement. Note that the refinement can be multi-iteration, thus refinement of the i-th iteration is formulated as $(\delta\mathbf{s}_f^i, \mathbf{h}_f^i, \mathbf{p}) = f_r(\mathbf{h}_f^{i-1}, \mathbf{s}_f^{i-1}, \mathbf{c}^{i-1})$. The generated offset $\delta\mathbf{s}_f^i$ is added to the input trajectory $\mathbf{s}_f^{i-1}$ for refinement.

3.2. Scenario-Adaptive Refinement

3.2.1 Adaptive Anchor/Context Selection

Trajectory refinement necessitates informative anchor selection and context retrieval. After the backbone generates the initial trajectories and trajectory features, SmartRefine first selects an adaptive number of anchors along the trajectories, and then retrieves contexts near the anchors with adaptive radius (illustrated in the bottom left in Fig. 1).

Anchor Selection. To select anchors along the trajectory, an intuitive option is to choose the last waypoint of the trajectory as the anchor for long-term context retrieval, as in goal-based predictions [32, 38]. However, relying exclusively on the endpoint as an anchor often misses the intermediate evolution and progression of the trajectory, resulting in deviations and oscillations. On the other extreme, selecting all T_f waypoints as anchors will lead to large but unnecessary computations. To balance between context richness and computation efficiency, we adopt an adaptive approach, where the trajectory is divided into $N < T_f$ segments and each segment's endpoint is selected as an anchor. Note that motion prediction benchmarks typically consider varied prediction horizon lengths. To depress the cumulative error in long-horizon prediction, the number of segments N is designed to adapt to the prediction horizon.

Context Retrieval. When retrieving context information $\mathbf{c}$ around the anchors, previous methods usually employ a fixed radius or rectangle around the anchor to extract local contexts (e.g. map and nearby agents) [5, 25]. Such fixed retrieval strategies, though straightforward, can be sub-optimal

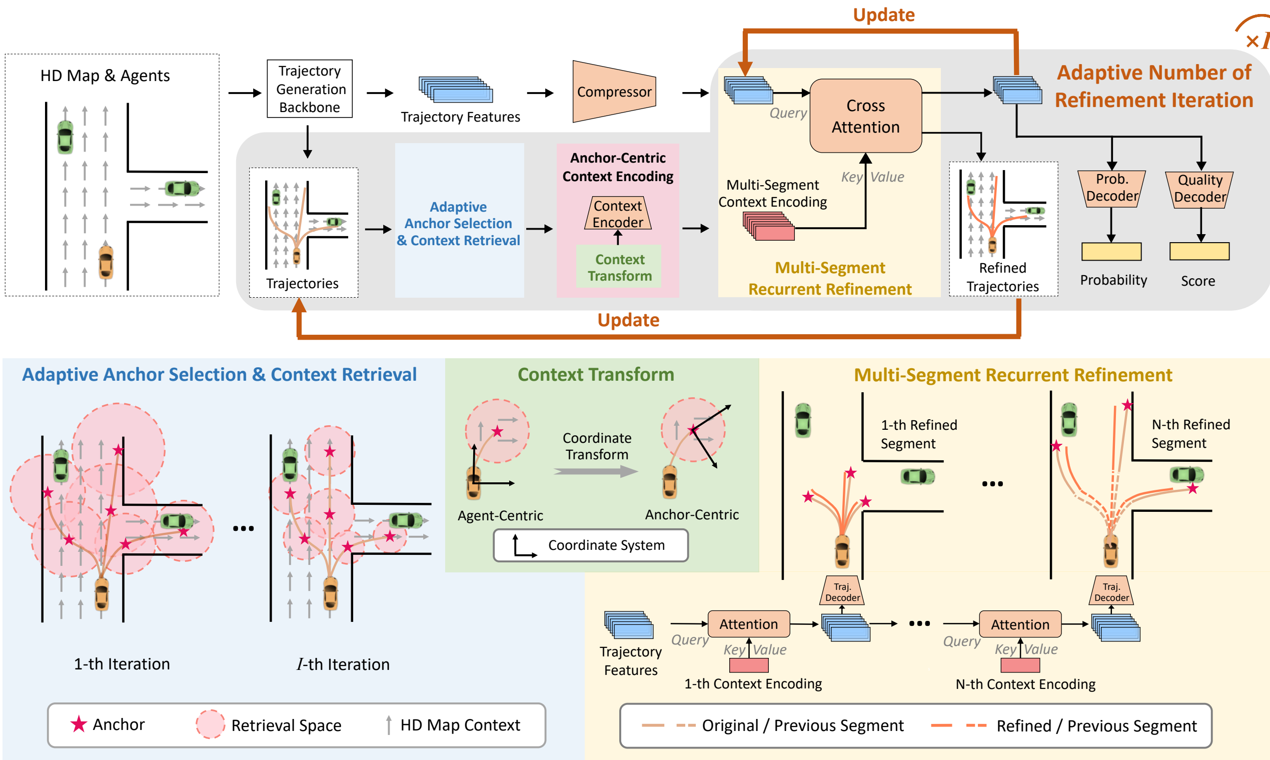

Figure 1. Overview of our framework. The top section concisely illustrates the full pipeline, while the lower section introduces details of three core modules. We first pass HD map and agent information to a prediction model backbone, generating initial trajectories and trajectory features. The initial predicted trajectory is then used to adaptively select anchors and retrieve critical context elements (bottom left). The contexts retrieved by each anchor are transformed to the coordinate frame centered at the corresponding anchors (bottom middle). The encoded contexts are then utilized to refine each trajectory segment by generating the offset of each trajectory segment (bottom right). Our model will predict a quality score measuring the prediction quality, and adaptively decide the number of refinement iterations (upper right). Our method is lightweight and can be seamlessly integrated with most existing motion prediction models.

especially when applied across diverse datasets and scenarios which could require different retrieval ranges. To address this issue, our idea is that, the retrieval range of one anchor should depend on 1) the refinement iteration i, because only fine-grained context information is needed in later refinement iterations, where the trajectories are more accurate compared to early iterations. 2) the target agent's speed v around the anchor, since large speed requires a far and long-term vision of the environment context. Thus we introduce an adaptive retrieval strategy, where each anchor's retrieval range R varies by the refinement iterations i and the agent's average velocity v around the anchor: $R_{i,v} = \mathcal{F}(i) \cdot v$. $\mathcal{F}(i)$ can be any monotonically decreasing function of the refinement iteration i, to reduce the retrieval range as the trajectory becomes more precise. We instantiate it with an exponential decay function $\mathcal{F}(i) = \beta(\frac{1}{2})^{i-1}$, where β denotes a constant. The retrieval range is also constrained within a permissible range $[R_{\min}, R_{\max}]$ to ensure computational stability and efficiency.

3.2.2 Anchor-Centric Context Encoding

With the retrieved context elements, most existing methods directly use their embeddings generated by the prediction backbone for further refinement [40]. However, these context embeddings from the backbone are typically encoded in the coordinate frame centered at the target agent's current position. While this agent-centric embedding emphasizes the context details around the target agent, the refinement process instead desires the context nuances along the future trajectories. To address this issue, we propose an anchor-centric context encoding approach, to better capture future trajectory details. (illustrated in the bottom middle in Fig. 1) This is achieved by transforming the context features into the coordinate frame centered and aligned with the anchor, and then encoding the features to generate anchor-centric context embeddings for refinement. Note that as the positions and orientations of anchors are dynamically changing across different refinement iterations i, these anchor-centric context features could also vary. This encoding process is applied to every context element retrieved around the anchor a, generating a set of context embeddings $\mathbf{z}_a$.

3.2.3 Recurrent and Multi-Iteration Refinement

The anchor-centric context embeddings introduced in the previous section are then fused with the trajectory embeddings to refine predictions. To this end, previous methods usually fuse all embeddings once and refine the trajectory of the whole future horizon, which suffers from long-term cumulative error. To mitigate this issue, we adopt a recurrent refinement strategy where we divide the trajectory into N segments, and refine the trajectory N times, one trajectory segment each time (illustrated in the bottom right in Fig. 1). Note that here N equals the number of anchors, which means each trajectory segment corresponds to one anchor. For each trajectory segment, we refine it only using the context retrieved by the corresponding anchor to enhance local context fusion. Besides, refining all N segments finishes one iteration of refinement, and we will conduct multiple refinement iterations.

Recurrent Refinement. Specifically, in each recurrent refinement step, we refine one trajectory segment, with a set of scene context embeddings $\mathbf{z}_a$ around the segment's corresponding anchor a, and the target agent's future trajectory embeddings $\mathbf{h}_f$. We adopt the cross attention mechanism [27] to fuse them, where the trajectory embeddings are used as queries to attend to the keys/values from the context embeddings $\mathbf{z}_a$. The fused trajectory embeddings will be used to predict $\delta\mathbf{s}$, the offset of waypoints in the trajectory segment, to subtly adjust the original trajectory segment. The updated trajectory embeddings will also be leveraged as new queries to refine the next segment. After all N steps within one iteration, the whole trajectory will be adjusted, and the trajectory embeddings will also be updated with rich context information. Note that we predict multiple possible predictions, thus the trajectory embeddings will also be used to predict the probability of each prediction.

Multi-Iteration Refinement. After one iteration terminates, the updated trajectory and trajectory embeddings will be used to start another refinement iteration. We first repeat the adaptive anchor selection and context retrieval mentioned in Sec. 3.2.1, and then conduct another N recurrent refinement steps. Thus in the multi-iteration refinement, the trajectory and trajectory embeddings used in one iteration can come from either the backbone (for the first iteration) or the previous refinement iteration (for the later iterations). Note that in the first refinement iteration, we employ a compressor network to reduce the hidden dimension of the trajectory embedding from the backbone for efficient refinement, which we found brings little performance loss.

3.2.4 Adaptive Number of Refinement Iterations

As mentioned in the previous section, the refinement is multi-iteration. As we get into later iterations, the predicted trajectory becomes more accurate. Nonetheless, there's a diminishing return as we utilize more refinement iterations. To trade-off between performance gain and additional computation, we propose an adaptive refinement strategy where the number of refinement iterations is dynamically adapted according to the current prediction quality and remaining potential refinement improvement. Specifically, we propose a quality score to quantify current prediction quality.

Algorithm 1 Adaptive Inference with SmartRefine

Input: Backbone model f_b, refinement model f_r, quality score decoder f_d, agents history trajectories $\mathbf{s}_h$, scene context $\mathbf{c}$, and score threshold $\bar{q}$, maximum refinement iteration at inference I'.
Output: Target agent's future trajectories $\mathbf{s}_f$ and corresponding probabilities $\mathbf{p}$.

```
s_f^0, h_f^0, p^0 = f_b(s_f^0, c)                  % Backbone prediction
q^0 = f_d(h_f^0)                                   % Initial quality score
if q^0 > q̄ then                                    % No need for refinement
    Return s_f^0, p^0
for i = 1, 2, ..., I' do                           % Multi-iteration refine
    Adaptively select anchors and contexts
    Adpatively encode contexts c^{i-1} (anchor-centric)
    Multi-segment recurrent refinement:
        δs_f^i, h_f^i, p^i, q^i = f_r(s_f^{i-1}, h_f^{i-1}, c^{i-1})
        s_f^i = s_f^{i-1} + δs_f^i
    if q^i < q^{i-1} then                          % Terminate refinement
        Return s_f^i, p^i
Return s_f^i, p^i                                  % Reach max refinement iteration
```

Quality Score Design. At the training stage, since we have access to the ground-truth trajectory and predicted trajectory of all refinement iterations, an intuitive measure for the quality of the predicted trajectory in iteration i is $d_{max} - d_i$, where d_{max} denotes the largest predicted error among all iterations. A small d_i means that the current prediction has already been improved from the largest prediction error d_{max}, and thus of high quality. However, this design as the quality score can be unstable as it lies in a big range and can vary a lot in different scenarios. We then normalize $d_{max} - d_i$ with $d_{max} - d_{min}$, where d_{min} denotes the smallest predicted error among all iterations. Thus the final quality score lies in between [0,1] and is designed as

$$q_i = \frac{d_{max} - d_i}{d_{max} - d_{min}} \tag{1}$$

To enable the model to predict the quality score, we first employ a GRU [4] to recurrently process the trajectory embeddings of all previous iterations, and then use an MLP to generate the quality score. Thus our model will generate three types of output: the predicted trajectories, the probabilities of the trajectories, and the quality score.

Adaptive Refinement Iteration. The contents above introduce how to design and generate the quality score at the training stage. At the inference stage, the initial trajectory features from the backbone will be appended with the quality score decoder to predict an initial quality score. Our model will also predict the quality score at each refinement iteration. Thus we propose a simple yet effective strategy to dynamically decide whether another refinement iteration is needed. As outlined in Algorithm 1, the adaptive strategy includes three major criteria: 1) an initial quality score threshold is set, and we only conduct refinement if the initial predicted quality score is below the threshold, which means the current prediction is not good enough and need refinement; 2) at any of the later refinement iterations, if the quality score stops growing compared to previous iterations, we terminate the refinement as this indicates that we are close to ideal performance, and further refinement will only bring diminishing returns or negative effects due to over-refinement; 3) a maximum refinement iteration is set, which is a hyper-parameter to balance performance and efficiency.

3.3. Training Loss

The training of our model considers three loss terms, and uses hyper-parameter α to balance them:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{reg}} + \alpha \cdot \mathcal{L}_{\text{score}} \tag{2}$$

where $\mathcal{L}_{\text{cls}}$ denotes the cross-entropy classification loss for predicting the probability of the multi-modal trajectories. For the regression loss $\mathcal{L}_{\text{reg}}$, as our model predicts a Laplace distribution for each time step's waypoint, we calculate the negative log-likelihood of the ground-truth trajectory in the predicted distribution. Note that among all predicted multi-modal trajectories, only the modal closest to the ground truth is considered for the regression loss. For the quality score loss $\mathcal{L}_{\text{score}}$, in the training stage, we fix the refinement iteration as I and each iteration will output a quality score. Thus for each iteration i, we calculate the ℓ_1 loss between the predicted quality score $\hat{q}_i$ and labeled quality score q_i (see Sec 3.2.4), and average the loss over all iterations:

$$\mathcal{L}_{\text{score}} = \frac{1}{I+1} \sum_{i=0}^{I} \|\hat{q}_i - q_i\|_1 \tag{3}$$

Similarly, the score loss also only considers the modal closest to the ground truth.

4. Experiments

We first report the prediction accuracy and computation cost on the validation set of the two datasets. As shown in Table 2, SmartRefine can consistently improve the prediction accuracy of all considered state-of-the-art methods with limited added parameters, Flops, and latency.

4.1. Experimental Settings

Dataset. We train and evaluate our method on two large-scale motion forecasting datasets: Argoverse [3] and Argoverse 2 [34]. Argoverse contains 333k scenarios collected from interactive and dense traffic. Each scenario provides the HD map and 2 seconds of history trajectory data, to predict the trajectory for future 3 seconds, sampled at 10Hz. Following the official guide, we split the training, validation, and test set to 205k, 39k, 78k scenarios respectively. Argoverse 2 upgrades the previous dataset to include 250K sequences with higher prediction complexity. It extends the historic and prediction horizon to 5 seconds and 6 seconds respectively, sampled at 10Hz. The data is split into 200k, 25k, and 25k for training, validation, and test respectively.

Metrics. Following the official dataset settings, we evaluate our model on the standard metrics for motion prediction, including minimum Average Displacement Error (minADE), minimum Final Displacement Error (minFDE), and Miss Rate (MR). These metrics allow models to forecast up to 6 trajectories for each agent and define the one that has the minimum endpoint error as the best trajectory to calculate the above metrics.

Baselines. As mentioned earlier, our SmartRefine can be seamlessly integrated into most existing trajectory prediction methods to improve accuracy with limited additional computation. In our experiment, we consider five popular and state-of-the-art methods as the prediction backbone to evaluate how our SmartRefine further improves the performance: HiVT [39], Prophnet [33], mmTransformer [14], DenseTNT [10] and QCNet [40]. Note that, since QCNet includes a refinement module itself, we introduce another baseline QCNet (no refinement) which removes the original refinement module from the QCNet. For implementation and computation details of our methods, please refer to the supplementary material.

4.2. Quantitative Result

Performance on Val Set. We first report the prediction accuracy and computation cost on the validation set of the two datasets. As shown in Table 2, SmartRefine can consistently improve the prediction accuracy of all considered state-of-the-art methods with limited added parameters, Flops, and latency. For instance, our method can reduce the minFDE of Prophnet, mmTransformer, HiVT, DenseTNT, and QCNet (no ref) by 6.0%, 3.7%, 5.4%, 4.3%, 3.5% respectively. Besides, our refinement model only added Flops by 130M on Argoverse and 408M on Argoverse 2, and thus is much smaller than the Flops of the backbone which range from 1.2G to 55.8G. Our method benefits QCNet less than other methods, because QCNet itself incorporates a relatively large refinement network that has 2,200K parameters. However, if we replace the refinement module in QCNet with ours, we can achieve competitive results with much less added param-

Dataset	Method	minFDE ↓	minADE ↓	MR ↓	#Param.(M) ↓	Flops(G) ↓	Latency(ms) ↓
Argoverse	HiVT [39]	0.969	0.661	0.092	2.5	2.6	54±4.0
	HiVT w/ Ours	0.911	0.646	0.083	2.7	2.7	67±8.4
	Prophnet* [33]	1.004	0.687	0.093	15.2	7.8	59±1.7
	Prophnet w/ Ours	0.967	0.675	0.092	15.4	7.9	71±6.2
	mmTransformer [14]	1.081	0.709	0.102	2.6	1.2	15±4.8
	mmTransformer w/ Ours	1.023	0.692	0.094	2.8	1.3	27±9.7
Argoverse 2	DenseTNT [10]	1.624	0.964	0.233	1.6	3.6	1,075±199
	DenseTNT w/ Ours	1.553	0.834	0.221	1.9	4.0	1,099±212
	QCNet (no ref)	1.304	0.729	0.164	5.5	47.0	338±53
	QCNet (no ref) w/ Ours	1.258	0.718	0.157	5.8	47.4	363±67
	QCNet [40]	1.253	0.720	0.157	7.7	55.8	392±54
	QCNet w/ Ours	1.240	0.716	0.156	8.0	56.2	418±68

Table 2. Performance and computation efficiency on Argoverse and Argoverse 2 val set. Methods that are not open-source and reproduced by us are marked with the symbol "*". QCNet (no ref) denotes the version where we remove the original refinement module in the QCNet. SmartRefine consistently improves the accuracy of all considered state-of-the-art methods with limited added computation and latency.

eters, flops, and latency.

Performance on Test Set. We also submit our method to Argoverse and Argoverse 2 test set and the results are shown in Table 3. Again, SmartRefine consistently improves the prediction accuracy of all considered methods. For instance, SmartRefine reduces the minFDE of HiVT, Prophnet, mmTransformer, DenseTNT, and QCNet (no ref) by 3.4%, 6.9%, 7.5%, 4.2%, 3.9% respectively. Specifically, our SmartRefine based on QCNet outperforms all published ensemble-free works on the Argoverse 2 leaderboard (single agent track) at the time of the paper submission. See Sec. C for a more detailed explanation.

4.3. Ablation studies

We conducted comprehensive ablation studies on the effect of each component in our proposed method, on the Argoverse validation set. We show the performance when we add SmartRefine on top of HiVT, and the results based on other backbones can be found in the supplementary material.

Adaptive Refinement Iterations. Fig. 2 ablates the effect of adaptive refinement iterations. When we use a fixed number of iterations (blue curve), the prediction accuracy improves with more refinement iterations, but diminishing or zero improvements are witnessed in later iterations. In comparison, when we adopt the adaptive refinement iteration proposed by SmartRefine, we can achieve higher accuracy with a smaller number of refinement iterations. Note that we studied different quality score thresholds $\bar{q}$ for adaptive refinement, while a higher threshold leads to better accuracy, we find $\bar{q} = 0.5$ achieves an ideal performance and higher thresholds perform similarly. Besides, for each threshold, we ablate different limits for maximum refinement iteration, resulting in 5 points for each curve.

Anchor Numbers. Table 4 ablates the effect of anchor numbers. We can see that increasing the anchor number from 1 to 2 reduces the minFDE. However, excessively increasing the number of anchors is ineffective, as it brings much larger model parameters with the same or slightly worse accuracy.

Dataset	Method	minFDE ↓	minADE ↓	MR ↓
Argoverse	HiVT [39]	1.17	0.77	0.13
	HiVT w/ Ours	1.13	0.77	0.12
	ProphNet* [33]	1.30	0.85	0.14
	Prophnet* w/ Ours	1.21	0.81	0.13
	mmTransformer [14]	1.34	0.84	0.15
	mmTransformer w/ Ours	1.24	0.81	0.14
Argoverse 2	DenseTNT [10]	1.66	0.99	0.23
	DenseTNT w/ Ours	1.59	0.85	0.22
	QCNet (no ref)	1.29	0.65	0.16
	QCNet (no ref) w/ Ours	1.24	0.64	0.15
	QCNet [40]	1.24	0.64	0.15
	QCNet w/ Ours	1.23	0.63	0.15

Table 3. Performance on Argoverse (1&2) test set. Methods that are not open-source and reproduced by us are marked with the symbol "*". The original DenseTNT and QCNet (no ref) did not report results in the test set, thus we train them to match the validation set and submit them to the test set.

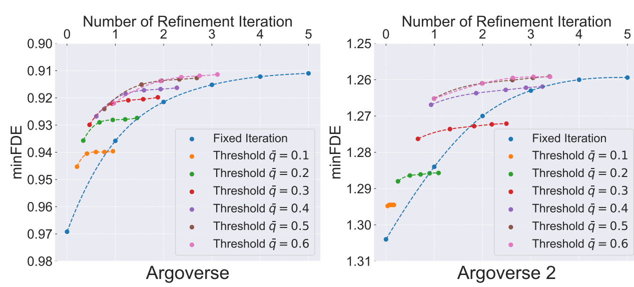

Figure 2. Comparison between the fixed and adaptive number of refinement iterations. For the adaptive methods, We tested different quality score thresholds $\bar{q}$ mentioned in Algorithm 1.

Context Representation. In Table 5, we compare the performance when we employ the fixed agent-centric context embeddings or the adaptive anchor-centric context embeddings (see details in Sec 3.2.2). We can see our adaptive anchor-centric encoding effectively outperforms the fixed agent-centric context encoding.

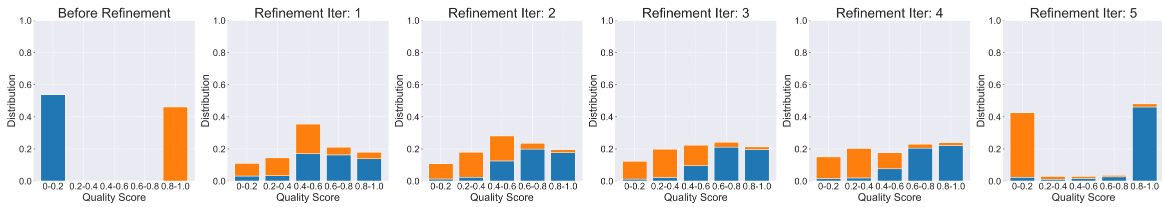

Figure 3. A study to understand the mechanism behind the refinement. Specifically, We mark the quality score distribution of the predictive trajectory before refinement, and track how the quality score changes along the multi-iteration refinement. We can see while the overall performance is improved, not every trajectory benefits from refinement, which implies the necessity of adaptive refinement. See Sec. 4.4 for detailed discussions.

Retrieval Radius. Table 6 studies the effect of the radius used to retrieve contexts. We can see a fixed retrieval radius from 50 to 2 can be sub-optimal, as a large retrieval radius might lead to redundant or unrelavant context information, while a small radius might not provide sufficient context for refinement. In comparison, our SmartRefine adapts the radius to each agent's velocity, and decays the radius with the number of refinement iterations (see details in Sec 3.2.1). Here we consider two strategies for radius decay: linear decay and exponential decay. Both methods lead to lower minFDE compared to the fixed radius, and the exponential decay has lower Flops compared to linear decay.

#Anchor numbers	minFDE	#Param.
1	0.928	134K
2	0.911	207K
3	0.911	280K
5	0.915	433K
6	0.916	509K

Table 4. Ablation study on the number of anchors.

Context Encoding	minFDE
Agent-Centric	0.941
Anchor-Centric	0.911

Table 5. Ablation study on how the contexts are encoded.

	Retrieval Radius	minFDE	Flops (M)
Fixed Radius	50	0.926	2,297
	20	0.923	722
	10	0.921	325
	2	0.930	58
Adaptive Radius	R_{max}=10, R_{min}=2, linear	0.911	245
	R_{max}=10, R_{min}=2, exp	0.911	130

Table 6. Ablation study on the context retrieval radius. Linear and exp denote different ways to decay the radius.

4.4. Discussion

While many previous works have explored various refinement strategies, few of them explain how exactly the refinement works and what limitations it has. For example, it remains unclear whether every trajectory is improved by the refinement. In this section, we take a step forward to study the mechanism behind the refinement. Specifically, for each predicted trajectory, we measure its accuracy using the quality score proposed earlier. We will track how the quality score changes along the multi-iteration refinements. As shown in Fig. 3, before the refinement, the quality score of the trajectories most fall in two categories: about 56% of the trajectories have a low quality score between 0 and 0.2 (marked as blue), and about 44% trajectories have a high quality score between 0.8 and 1 (marked as orange). We then conduct multi-iteration refinement and track how the quality score of the "blue" and "orange" trajectories change. Two phenomenons are observed:

- *Not every trajectory benefits from refinement*: as the refinement goes, we can see that the initially low-score "blue" trajectories gradually move to the right, which means they become more accurate. However, the initially high-score "orange" trajectories gradually move to the left, which means they become less accurate. In fact, we can see that initially low-score blue trajectories and initially high-score orange trajectories switched their quality score at the final iteration. We hypothesize that this is because while inaccurate trajectories can get much improvement from refinement, the accurate trajectories are good enough and can hardly be further improved, or even be pushed away from the ground truth due to over-refinement.
- *The overall performance is better:* though the initially accurate and inaccurate trajectories switch their quality scores at the final iteration, we see the final iteration has a higher percentage of high-score trajectories. This is why our refinement can lead to a higher overall performance.

These results show the necessity of adaptive refinement.

5. Conclusion

In this paper, we introduce SmartRefine, a scenario-adaptive refinement framework for efficient refinement of motion prediction models. Our method adopts adaptive strategies anchor selection, retrieval radius, and context encoding to conduct refinement that better suits each scenario. We then propose the quality score to indicate current prediction quality and potential refinement improvement, and adaptively decide how many refinement iterations are needed for each scenario. Our SmartRefine is not only lightweight, but can also be seamlessly plugged into most existing motion prediction models as it is decoupled from the prediction backbone and only relies on a generic interface between the backbone. Extensive experiments demonstrate the effectiveness of our approach in terms of both prediction accuracy and computation efficiency. We also study the mechanism behind multi-iteration refinement. In the future, we will further extend our framework to multi-agent joint prediction settings.

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 3

[2] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *arXiv preprint arXiv:1910.05449*, 2019. 1, 2

[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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8748–8757, 2019. 6

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 5

[5] Sehwan Choi, Jungho Kim, Junyong Yun, and Jun Won Choi. R-pred: Two-stage motion prediction via tube-query attention-based trajectory refinement. *arXiv preprint arXiv:2211.08609*, 2022. 2, 3

[6] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2090–2096. IEEE, 2019. 2

[7] Fang Da and Yu Zhang. Path-aware graph attention for hd maps in motion prediction. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6430–6436. IEEE, 2022. 2

[8] Lan Feng, Quanyi Li, Zhenghao Peng, Shuhan Tan, and Bolei Zhou. Trafficgen: Learning to generate diverse and realistic traffic scenarios. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3567–3575. IEEE, 2023. 2

[9] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11525–11533, 2020. 1, 3

[10] Junru Gu, Chen Sun, and Hang Zhao. Densetnt: End-to-end trajectory prediction from dense goal sets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15303–15312, 2021. 6, 7, 11

[11] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862, 2023. 1

[12] Quanyi Li, Zhenghao Mark Peng, Lan Feng, Zhizheng Liu, Chenda Duan, Wenjie Mo, and Bolei Zhou. Scenarionet: Open-source platform for large-scale traffic scenario simulation and modeling. *Advances in neural information processing systems*, 36, 2024. 2

[13] Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. Learning lane graph representations for motion forecasting. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 541–556. Springer, 2020. 1, 3

[14] Yicheng Liu, Jinghuai Zhang, Liangji Fang, Qinhong Jiang, and Bolei Zhou. Multimodal motion prediction with stacked transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7577–7586, 2021. 6, 7, 11

[15] Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S Refaat, and Benjamin Sapp. Wayformer: Motion forecasting via simple & efficient attention networks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2980–2987. IEEE, 2023. 11

[16] Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, et al. Scene transformer: A unified architecture for predicting multiple agent trajectories. *arXiv preprint arXiv:2106.08417*, 2021. 1

[17] Yael Niv. Learning task-state representations. *Nature neuroscience*, 22(10):1544–1553, 2019. 1

[18] Daehee Park, Hobin Ryu, Yunseo Yang, Jegyeong Cho, Jiwon Kim, and Kuk-Jin Yoon. Leveraging future relationship reasoning for vehicle trajectory prediction. *arXiv preprint arXiv:2305.14715*, 2023. 2

[19] Angela Radulescu, Yael Niv, and Ian Ballard. Holistic reinforcement learning: the role of structure and attention. *Trends in cognitive sciences*, 23(4):278–292, 2019. 1

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 3

[21] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, pages 683–700. Springer, 2020. 1

[22] Hao Shao, Yuxuan Hu, Letian Wang, Steven L Waslander, Yu Liu, and Hongsheng Li. Lmdrive: Closed-loop end-to-end driving with large language models. *arXiv preprint arXiv:2312.07488*, 2023. 1

[23] Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In *Conference on Robot Learning*, pages 726–737. PMLR, 2023.

[24] Hao Shao, Letian Wang, Ruobing Chen, Steven L Waslander, Hongsheng Li, and Yu Liu. Reasonnet: End-to-end driving with temporal and global reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13723–13733, 2023. 1

[25] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Motion transformer with global intention localization and local

movement refinement. *Advances in Neural Information Processing Systems*, 35:6531–6543, 2022. 2, 3

[26] 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 *2022 International Conference on Robotics and Automation (ICRA)*, pages 7814–7821. IEEE, 2022. 1, 2

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 5

[28] Letian Wang, Yeping Hu, Liting Sun, Wei Zhan, Masayoshi Tomizuka, and Changliu Liu. Hierarchical adaptable and transferable networks (hatn) for driving behavior prediction. *arXiv preprint arXiv:2111.00788*, 2021. 1

[29] Letian Wang, Liting Sun, Masayoshi Tomizuka, and Wei Zhan. Socially-compatible behavior design of autonomous vehicles with verification on real human data. *IEEE Robotics and Automation Letters*, 6(2):3421–3428, 2021. 1

[30] Letian Wang, Yeping Hu, Liting Sun, Wei Zhan, Masayoshi Tomizuka, and Changliu Liu. Transferable and adaptable driving behavior prediction. *arXiv preprint arXiv:2202.05140*, 2022. 1

[31] Letian Wang, Jie Liu, Hao Shao, Wenshuo Wang, Ruobing Chen, Yu Liu, and Steven L Waslander. Efficient reinforcement learning for autonomous driving with parameterized skills and priors. *arXiv preprint arXiv:2305.04412*, 2023. 1

[32] 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 *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1609–1615. IEEE, 2023. 2, 3, 11, 12

[33] Xishun Wang, Tong Su, Fang Da, and Xiaodong Yang. Prophnet: Efficient agent-centric motion forecasting with anchor-informed proposals. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21995–22003, 2023. 2, 3, 6, 7, 11

[34] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, et al. Argoverse 2: Next generation datasets for self-driving perception and forecasting. *arXiv preprint arXiv:2301.00493*, 2023. 6

[35] Luyao Ye, Zikang Zhou, and Jianping Wang. Improving the generalizability of trajectory prediction models with frenet-based domain normalization. *arXiv preprint arXiv:2305.17965*, 2023. 1

[36] Maosheng Ye, Jiamiao Xu, Xunnong Xu, Tengfei Wang, Tongyi Cao, and Qifeng Chen. Bootstrap motion forecasting with self-consistent constraints. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8504–8514, 2023. 1, 2, 3

[37] Lingyao Zhang, Po-Hsun Su, Jerrick Hoang, Galen Clark Haynes, and Micol Marchetti-Bowick. Map-adaptive goal-based trajectory prediction. In *Conference on Robot Learning*, pages 1371–1383. PMLR, 2021. 2

[38] Hang Zhao, Jiyang Gao, Tian Lan, Chen Sun, Ben Sapp, Balakrishnan Varadarajan, Yue Shen, Yi Shen, Yuning Chai, Cordelia Schmid, et al. Tnt: Target-driven trajectory prediction. In *Conference on Robot Learning*, pages 895–904. PMLR, 2021. 2, 3

[39] Zikang Zhou, Luyao Ye, Jianping Wang, Kui Wu, and Kejie Lu. Hivt: Hierarchical vector transformer for multi-agent motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8823–8833, 2022. 6, 7, 11

[40] Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-centric trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17863–17873, 2023. 1, 2, 3, 4, 6, 7, 11