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

Supplementary Material

The overall structure of the supplementary material is listed as follows:

A. Implementation Details

A.1. Backbones

We apply our SmartRefine to six classic and state-of-the-art motion prediction backbones: HiVT [39], Prophnet [33], mmTransformer [14], DenseTNT [10], QCNet [40], and QCNet (no refine) [40]. For implementation, we reproduce Prophnet since it is not open-source, and utilize the open-source codes for all other backbones. In the rest of this section, we introduce the implementation details of our SmartRefine.

A.2. Compressor

We use an MLP network to reduce the hidden dimension of the trajectory embeddings from backbones. The following hyperparameters are used:

- Number of layers: 2
- Input size: the original hidden size set by the trajectory generation backbone
- Output size: 64

A.3. Context Retrieve and Encoding

During refinement, we first select anchors along the trajectories, which are used to retrieve contexts. We represent each anchor's retrieval range as $R_{i,v} = \mathcal{F}(i) \cdot v$, as mentioned in Sec. 3.2.1. In practice: 1) $\mathcal{F}(i) = \beta(\frac{1}{2})^{i-1}$, and we set radius decay constant β as 0.8; 2) the average speed v around one anchor is calculated based on the speed when the agent passes through the anchor's corresponding trajectory segment.

When encoding the context, the early fusion strategy [15] is utilized. Specifically, for each context, we first use MLP layers to encode the context components (positions, semantic information, and distance to the anchor) separately, then add them together, and then apply MLP layers to encode the added embedding. We use the following model hyperparameters:

- Number of Anchors: 2 for Argoverse and 4 for Argoverse 2
- Number of encoder layers: 2
- Input size of each context: 2 for (x,y) positions and 1 for semantic information
- Hidden size: 64
- Fusion operator: "add"
- Number of fusion layers: 2

Rank	Method	minFDE ↓	minADE ↓	MR ↓
1	SEPT-iDLab (SEPT)*	1.15	0.61	0.14
2	GACRND-XLAB (XPredFormer)*	1.20	0.62	0.15
3	QCNet-AV2 (QCNet)	1.19	0.62	0.14
4	MTC (MTC)*	1.17	0.61	0.14
5	Mingkun Wang	1.19	0.62	0.14
6	AnonNet (AnonNet)*	1.23	0.63	0.15
7	ls (TraceBack)*	1.20	0.64	0.14
8	SmartRefine (ours)	1.23	0.63	0.15
-	QCNet (no ensemble)	1.24	0.64	0.15
-	GANet (published version)	1.35	0.73	0.17

Table 7. Argoverse 2 leaderboard (single agent track) at the time of the paper submission. Unpublished works are marked with the symbol "*". Our SmartRefine with QCNet as trajectory generation backbone ranked #8 on the leaderboard. Methods before ours are all unpublished except two methods: 1) The #3 method is the ensemble version of QCNet, while our method does not utilize ensemble. For a fair comparison, our method outperforms the ensemble-free version of QCNet (as marked in the second last row). 2) The #5 method is linked to GANet [32], which was published before. However, the performance in the original published GANet paper (as marked in the last row) is much poorer than that of the #5 method. We suppose the current #5 performance is obtained by the ensemble version of GANet or an unpublished extended version of GANet. Thus our method outperforms all published ensemble-free works on the Argoverse 2 leaderboard (single agent track) at the time of the paper submission.

A.4. Cross Attention for Refinement

We utilize a multi-head attention module to refine each trajectory segment. It takes the trajectory embeddings as queries and the context encodings as keys/values. We use the following model hyperparameters:

- Number of attention layers: 1
- Hidden size: 64
- Number of attention heads: 8
- Dropout: 0.1
- Activation: ReLU

A.5. Decoder

Our module has three decoders: trajectory decoder, probability decoder, and quality decoder. We represent them all as

Backbones Datasets	Metrics	Different Ideologies of Refinement Techniques					
		no ref	DCMS^1	QCNet^1	R-Pred^1	MTR^M	Ours^A
HiVT Argo	minFDE Latency	0.969 54±4.0	0.958 55±4.4	0.933 64±5.1	0.929 62±5.9	0.915 92±9.4	0.911 67±8.4
Prophnet Argo	minFDE Latency	1.004 59±1.7	0.996 60±2.4	0.984 68±3.2	0.981 65±3.1	0.968 88±5.9	0.967 71±6.2
mmTransformer Argo	minFDE Latency	1.081 15±4.8	1.066 16±5.5	1.048 22±5.6	1.045 21±5.5	1.022 51±8.4	1.023 27±9.7
DenseTNT Argo 2	minFDE Latency	1.624 1,075±199	1.601 1,076±199	1.563 1,133±217	1.576 1,085±209	1.553 1,125±213	1.553 1,099±212
QCNet (no ref) Argo 2	minFDE Latency	1.304 338±53	1.293 339±53	1.253 392±54	1.274 348±55	1.256 387±62	1.258 363±67

Table 8. Comparison of refinement methods. 1/M/A denotes one-iteration, multi-iteration, and adaptive-iteration refinement methods respectively. Our method which utilizes adaptive refinement achieves the best trade-off in minFDE and latency.

MLP networks. We use the following model hyperparameters:

- Number of layers: 2
- Hidden size: 64
- Output size: $2 \cdot T_f$ for trajectory decoder, and 1 for probability/quality decoder.

A.6. Optimization

We train our model to minimize the negative log-likelihood of the ground truth trajectory. The training hyperparameters are set as follows:

- Loss balance constant α in Sec. 3.3: 0.01
- Number of refinement iteration I: 5
- Number of training epochs: 32
- Batch size: 8 for one single GPU and we use 8 GPUs
- Learning rate schedule: Cosine
- Initial learning rate: 0.001
- Optimizer: AdamW
- Weight decay: 0.0001

B. Evaluation Details

B.1. Standard Metric

minADE & minFDE. minADE measures the Euclidean distance error averaged over all timesteps for the closest prediction, relative to ground truth. In contrast, minFDE considers only the distance error at the final timestep.

Miss Rate. Miss rate is a measure of what fraction of scenarios fail to generate any predictions within the lateral and longitudinal error thresholds, relative to the ground truth future. In Argoverse and Argoverse 2 dataset, the threshold is set to 2.0 by default.

B.2. Additional Metric

#Param. Analysis. #Param. is the total number of model parameters. It reveals the model's memory cost. We count the model's parameters via PyTorch Lightning.

Flops Analysis. Flops specifically refer to the number of floating-point operations when running the model, such as the matrix multiplications and activations. Flops are often used to measure the computational cost or complexity of a model. We measure Flops of model forward with batch_size=1 during inference so gradient calculations are not considered.

Latency Analysis. Latency is the time required to execute the model, which is generally used in the context of measuring computation efficiency. To simulate realistic settings where multiple agents exist, we measure the latency of the model with batch_size=32 over Argoverse / Argoveres 2 val set.

C. Performance on Argoverse 2 Leaderboard

In Table 7, we show the details of how our SmartRefine performs on the Argoverse 2 leaderboard (single agent track). At the time of the paper submission, our SmartRefine with QCNet as trajectory generation backbone ranked #8 on the leaderboard. Methods before ours are all unpublished except two methods:

- The #3 method QCNet is the ensemble version of QCNet, while our method does not utilize ensemble. For a fair comparison, our method outperforms the ensemble-free version of QCNet (as marked in the second last row).
- The #5 method is linked to GANet [32], which was published before. However, the performance in the original published GANet paper (as marked in the last row) is much poorer than that of the #5 method. We suppose the cur-

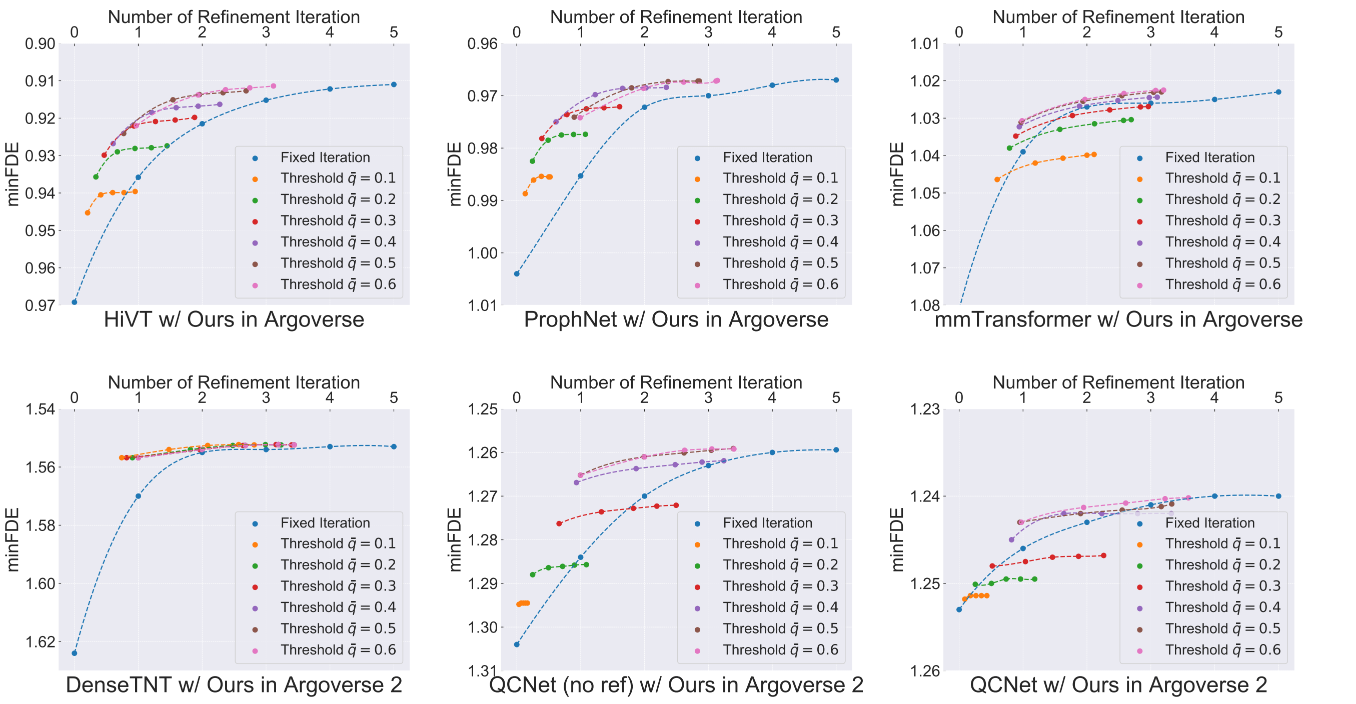

Figure 4. Comparison between the fixed and adaptive number of refinement iterations, when we apply our SmartRefine on all six considered backbones respectively (on val set of Argoverse and Argoverse 2). The blue curve represents fixed refinement iterations (for both training and inference). Other curves denote adaptive refinement iterations with different quality score threshold $\bar{q}$ during inference time (5 refinement iterations are utilized during training). For each threshold, we ablate different limits for maximum refinement iteration I', resulting in 5 points for each curve (refer to Algorithm 1 for detailed descriptions of $\bar{q}$ and I'). Two observations: 1) on HiVT, ProphNet, DenseTNT, and QCNet (no ref), our adaptive refinement strategy outperforms the fixed refinement strategy given any threshold $\bar{q}$. 2) on mmTransformer, and QCNet, our adaptive refinement strategy outperforms the fixed refinement strategy when we set a higher threshold $\bar{q}$ (0.4, 0.5, 0.6) when we decide whether another refinement iteration is needed.

rent #5 performance is obtained by the ensemble version of GANet or an unpublished extended version of GANet.

Thus our method outperforms all published ensemble-free works on the Argoverse 2 leaderboard (single agent track) at the time of the paper submission.

D. Comparison with other refinement methods

In the refinement methods listed in Table 1, DCMS and R-Pred are not open-sourced, and MTR is intended for Waymo dataset. For a fair and convenient comparison, we follow their ideologies to reproduce them under our framework. As in Table 8, DCMS has the worst minFDE but the best latency since it doesn't use any contexts. QCNet performs well but has the largest latency in Argoverse 2 with big maps, as it unselectively utilizes all the context. R-Pred selectively uses context along the whole trajectory and achieves good performance compared to other one-iteration methods (DCMS and QCNet). MTR as a multi-iteration refinement method outperforms one-iteration methods but has higher latency. Our method which utilizes adaptive refinement achieves the best trade-off in minFDE and latency.

E. Ablation Studies with All Backbones

In this main paper, we only reported ablation studies on certain backbones due to the page limit. Here we show the ablation studies on all backbones.

E.1. Refinement Iterations

In Fig. 2 of the main paper, we compared the fixed refinement strategy and our proposed adaptive strategy on two backbones: HiVT (Argoverse), and QCNet no refine (Argoverse 2). Here we show the results of our SmartRefine with all six considered backbones, shown in Fig. 4. Specifically, the blue curve denotes refinement with a fixed number of iterations (for both training and inference). Other curves denote adaptive refinement iterations with different quality score threshold $\bar{q}$ during inference (5 refinement iterations are utilized during training). For each threshold, we ablate different limits for maximum refinement iteration I', resulting in 5 points for each curve. Readers are referred to Algorithm 1 for detailed descriptions of $\bar{q}$ and I'. As shown in Fig. 4, the performance of all backbones can be improved by refinement. When we compare our adaptive strategy with the fixed strategy: 1) on HiVT, ProphNet, DenseTNT, and QCNet (no

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

#Anchor numbers	ProphNet w/ Ours Argoverse	
	minFDE	#Param.
1	0.978	143K
2	0.967	216K
3	0.967	290K
5	0.966	442K
6	0.967	518K

#Anchor numbers	mmTransformer w/ Ours Argoverse	
	minFDE	#Param.
1	1.035	143K
2	1.023	216K
3	1.021	290K
5	1.020	442K
6	1.018	518K

#Anchor numbers	DenseTNT w/ Ours Argoverse	
	minFDE	#Param.
1	1.582	176K
3	1.555	327K
4	1.553	406K
5	1.552	486K
6	1.553	566K

#Anchor numbers	QCNet (no ref) w/ Ours Argoverse	
	minFDE	#Param.
1	1.282	142K
3	1.260	284K
4	1.258	359K
5	1.258	435K
6	1.259	511K

#Anchor numbers	QCNet w/ Ours Argoverse	
	minFDE	#Param.
1	1.245	142K
3	1.242	284K
4	1.240	359K
5	1.242	435K
6	1.241	511K

Table 9. Ablation study on the number of anchors, when we apply our SmartRefine on all six considered backbones respectively (on val set of Argoverse and Argoverse 2). We can see a common trend that increasing the anchor number 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. Also, the two datasets desire different numbers of anchors because they consider different lengths of the prediction horizon. Thus we set the anchor number as **2** for Argoverse (marked grey), and **4** for Argoverse 2 experiments (marked grey). The experiments reported in the main paper are based on these two settings.

ref), our adaptive refinement strategy outperforms the fixed refinement strategy basically given any threshold $\bar{q}$. 2) on mmTransformer, and QCNet, our adaptive refinement strategy outperforms the fixed refinement strategy when we set a higher threshold $\bar{q}$ (0.4, 0.5, 0.6) when we decide whether another refinement iteration is needed.

E.2. Anchor Numbers

The results are shown in Table 9. We can see a common trend that increasing the anchor number 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. Also, the two datasets desire different numbers of anchors because they consider different lengths of the prediction horizon. Thus we set the anchor number as **2** for Argoverse (marked grey), and **4** for Argoverse 2 (marked grey). The experiments reported in the main paper are based on these two settings.

E.3. Context Representation

The results are shown in Table 10. We can see our adaptive anchor-centric encoding effectively outperforms the fixed agent-centric context encoding, on all backbones.

E.4. Retrieval Radius

The results are shown in Table 11. Common observations can be drawn: 1) the fixed retrieval radius from 50 to 2 can be sub-optimal, as a large retrieval radius might lead to redundant or irrelevant context information, while a small radius might not provide sufficient context for refinement. 2) our SmartRefine achieves lower prediction error and Flops, by adapting the radius to each agent's velocity, and decaying the radius with the number of refinement iterations (see details in Sec 3.2.1). Here we compare two strategies for radius decay: linear decay and exponential decay. We adopt exponential decay as it outperforms linear decay.

F. Quality Score of All Backbones

The quality score distribution over multiple refinement iterations, when we apply our SmartRefine on all six considered backbones respectively. Results are shown in Fig. 11. Specifically, for each predicted trajectory, we measure its accuracy using the quality score. We will track how the quality score changes along the multi-iteration refinements.
Common observations on the six backbones can be drawn: *1) not every trajectory benefits from refinement; 2) the overall performance becomes better after refinement.* These results demonstrate the necessity of adaptive refinement.

G. Visualization of Refinement

As shown in Fig. 12, Fig. 13, Fig. 14, Fig. 15 and Fig. 16, we show the visualization results of the predicted trajectories by our method, before and after refinement. These results demonstrate how our method can refine the trajectory to be closer to ground truth and be more compliant to the road context.

Context Encoding	HiVT w/ Ours
	Argoverse
	minFDE
Agent-Centric	0.941
Anchor-Centric	0.911

Context Encoding	ProphNet w/ Ours
	Argoverse
	minFDE
Agent-Centric	0.988
Anchor-Centric	0.967

Context Encoding	mmTransformer w/ Ours
	Argoverse
	minFDE
Agent-Centric	1.064
Anchor-Centric	1.023

Context Encoding	DenseTNT w/ Ours
	Argoverse
	minFDE
Agent-Centric	1.589
Anchor-Centric	1.553

Context Encoding	QCNet (no ref) w/ Ours
	Argoverse
	minFDE
Agent-Centric	1.276
Anchor-Centric	1.258

Context Encoding	QCNet w/ Ours
	Argoverse
	minFDE
Agent-Centric	1.249
Anchor-Centric	1.240

Table 10. Ablation study on how the contexts are encoded, when we apply our SmartRefine on all six considered backbones respectively (on val set of Argoverse and Argoverse 2). We can see our adaptive anchor-centric encoding effectively outperforms the fixed agent-centric context encoding, on all backbones.

	HiVT w/ Ours (Argoverse)		
	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

	ProphNet w/ Ours (Argoverse)		
	Retrieval Radius	minFDE	Flops (M)
Fixed Radius	50	0.976	2,181
	20	0.974	621
	10	0.975	332
	2	0.980	41
Adaptive Radius	R_{max}=10, R_{min}=2, linear	0.970	140
	R_{max}=10, R_{min}=2, exp	0.967	132

	mmTransformer w/ Ours (Argoverse)		
	Retrieval Radius	minFDE	Flops (M)
Fixed Radius	50	1.042	1,466
	20	1.030	468
	10	1.031	193
	2	1.041	30
Adaptive Radius	R_{max}=10, R_{min}=2, linear	1.028	156
	R_{max}=10, R_{min}=2, exp	1.023	101

	DenseTNT w/ Ours (Argoverse 2)		
	Retrieval Radius	minFDE	Flops (M)
Fixed Radius	50	1.566	4,018
	20	1.556	1,476
	10	1.558	839
	2	1.561	181
Adaptive Radius	R_{max}=10, R_{min}=2, linear	1.555	541
	R_{max}=10, R_{min}=2, exp	1.553	396

	QCNet (no ref) w/ Ours (Argoverse 2)		
	Retrieval Radius	minFDE	Flops (M)
Fixed Radius	50	1.266	4,337
	20	1.258	1,691
	10	1.261	995
	2	1.270	199
Adaptive Radius	R_{max}=10, R_{min}=2, linear	1.258	577
	R_{max}=10, R_{min}=2, exp	1.258	408

	QCNet w/ Ours (Argoverse 2)		
	Retrieval Radius	minFDE	Flops (M)
Fixed Radius	50	1.244	4,560
	20	1.245	1,756
	10	1.243	869
	2	1.246	186
Adaptive Radius	R_{max}=10, R_{min}=2, linear	1.241	594
	R_{max}=10, R_{min}=2, exp	1.240	410

Table 11. Ablation study on the fixed and adaptive radius for context retrieval, when we apply our SmartRefine on all six considered backbones respectively (on val set of Argoverse and Argoverse 2). Common observations can be drawn: 1) the fixed retrieval radius from 50 to 2 can be sub-optimal, as a large retrieval radius might lead to redundant or irrelevant context information, while a small radius might not provide sufficient context for refinement. 2) our SmartRefine achieves lower accuracy and Flops, by adapting the radius to each agent's velocity, and decaying the radius with the number of refinement iterations (see details in Sec 3.2.1). Here we compare two strategies for radius decay: linear decay and exponential decay. We adopt exponential decay as it outperforms linear decay.

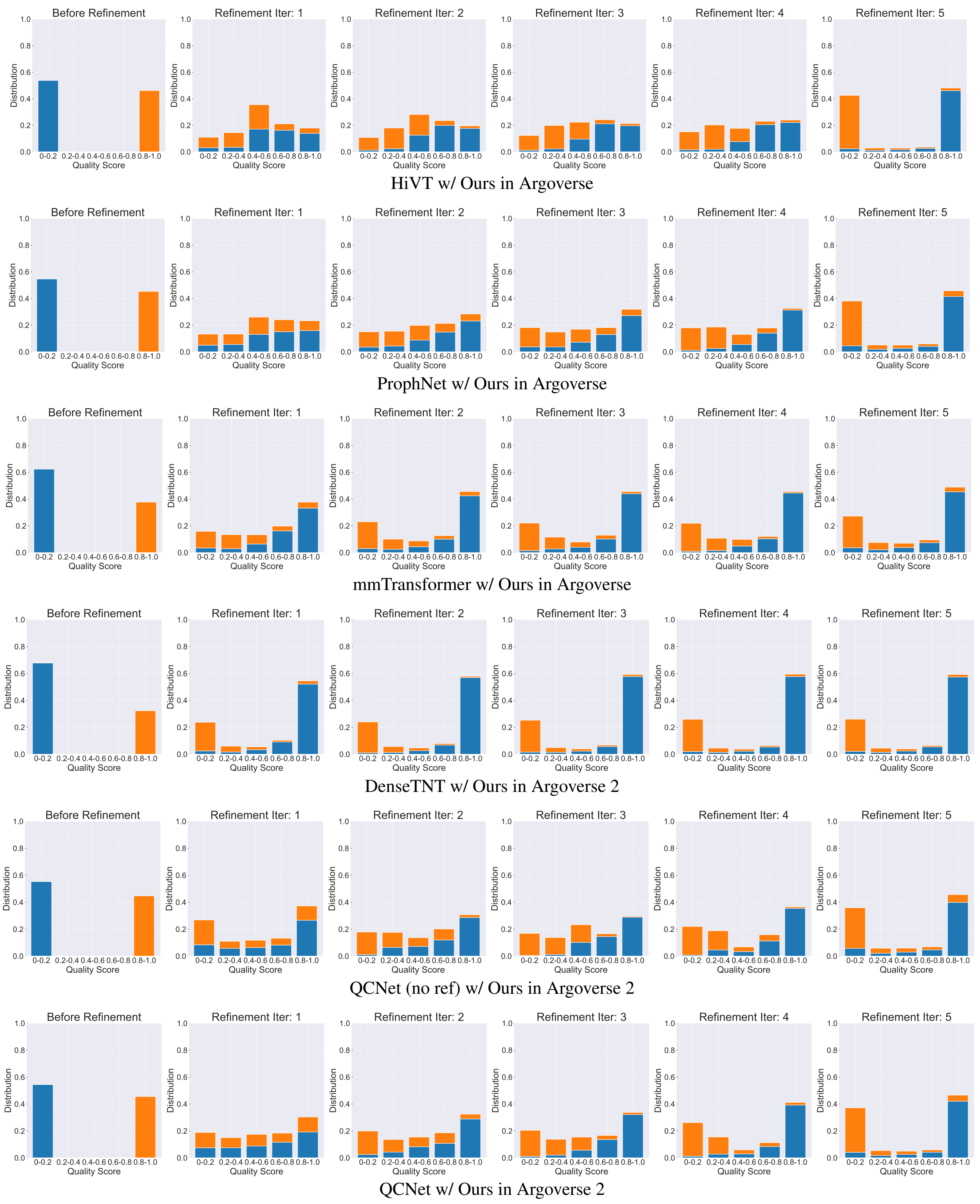

Figure 11. The quality score distribution over multiple refinement iterations, when we apply our SmartRefine on all six considered backbones respectively (Argoverse and Argoverse 2 training set). Specifically, for each predicted trajectory, we measure its accuracy using the quality score. We will track how the quality score changes along the multi-iteration refinements. Common observations on the six backbones can be drawn: *1) not every trajectory benefits from refinement; 2) the overall performance becomes better after refinement.* These results demonstrate the necessity of adaptive refinement.

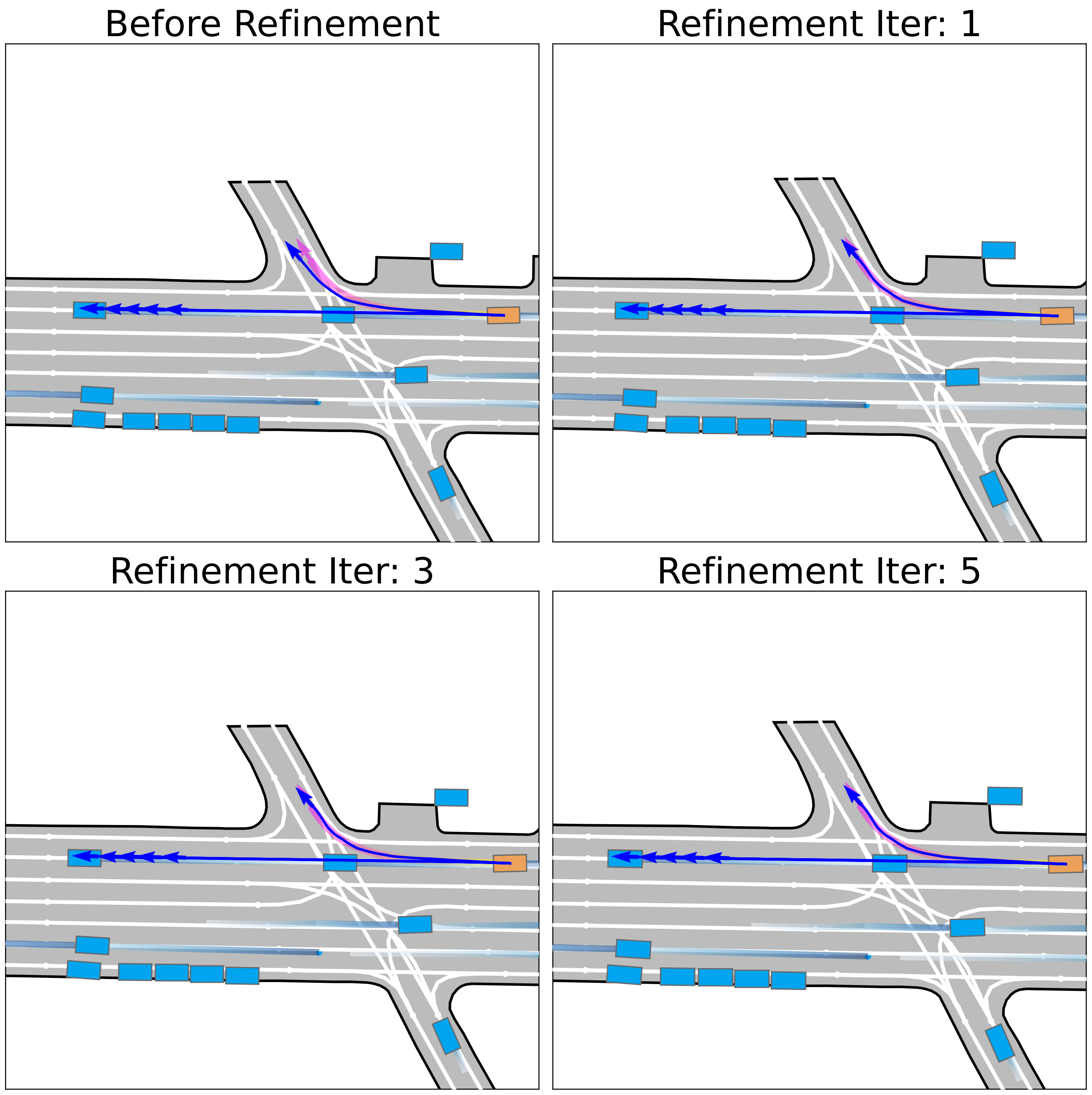

Figure 12. Visualization results. The dark blue arrows are multi-nodal predictions of the agent by model and the pink arrow is the ground truth future trajectory respectively. The trajectory (turn right) gets closer to the ground truth after refinement.

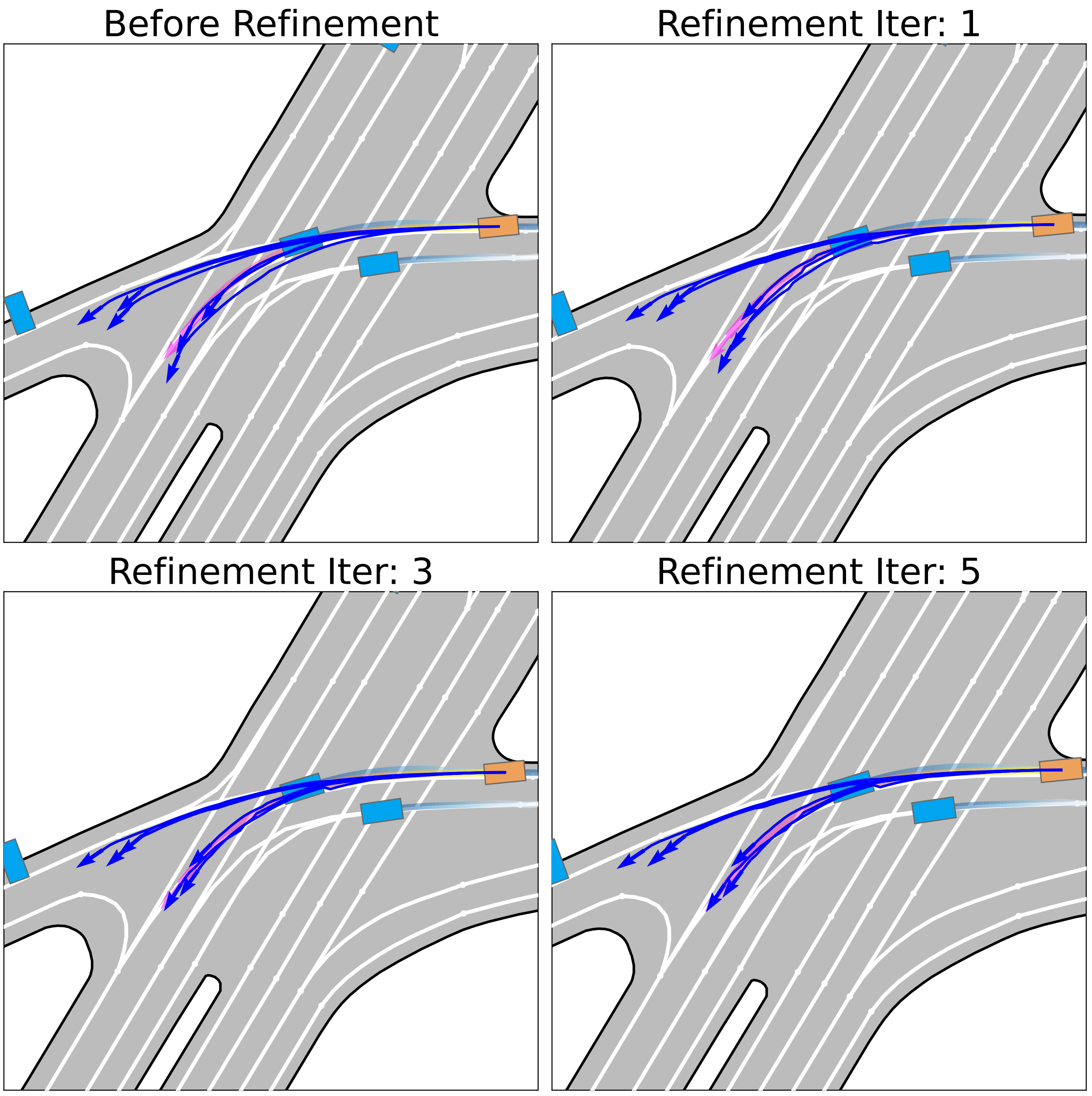

Figure 13. Visualization results. The dark blue arrows are multi-nodal predictions of the agent by model and the pink arrow is the ground truth future trajectory respectively. The trajectory closest to the ground truth gets closer after refinement.

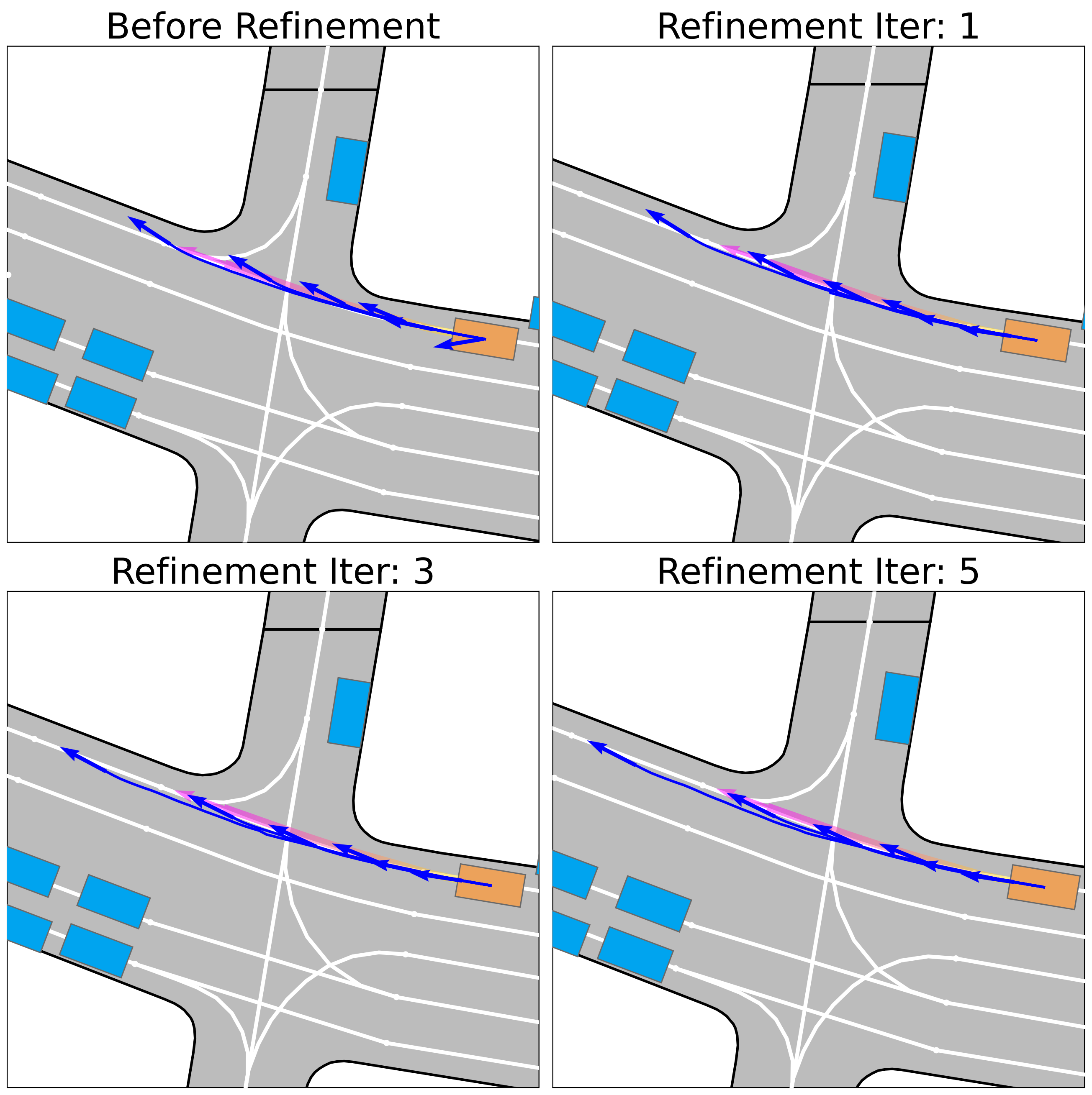

Figure 14. Visualization results. The dark blue arrows are multi-nodal predictions of the agent by model and the pink arrow is the ground truth future trajectory respectively. The shortest trajectory gets more aligned toward the ground truth direction, and the trajectory closest to the ground truth gets closer after refinement.

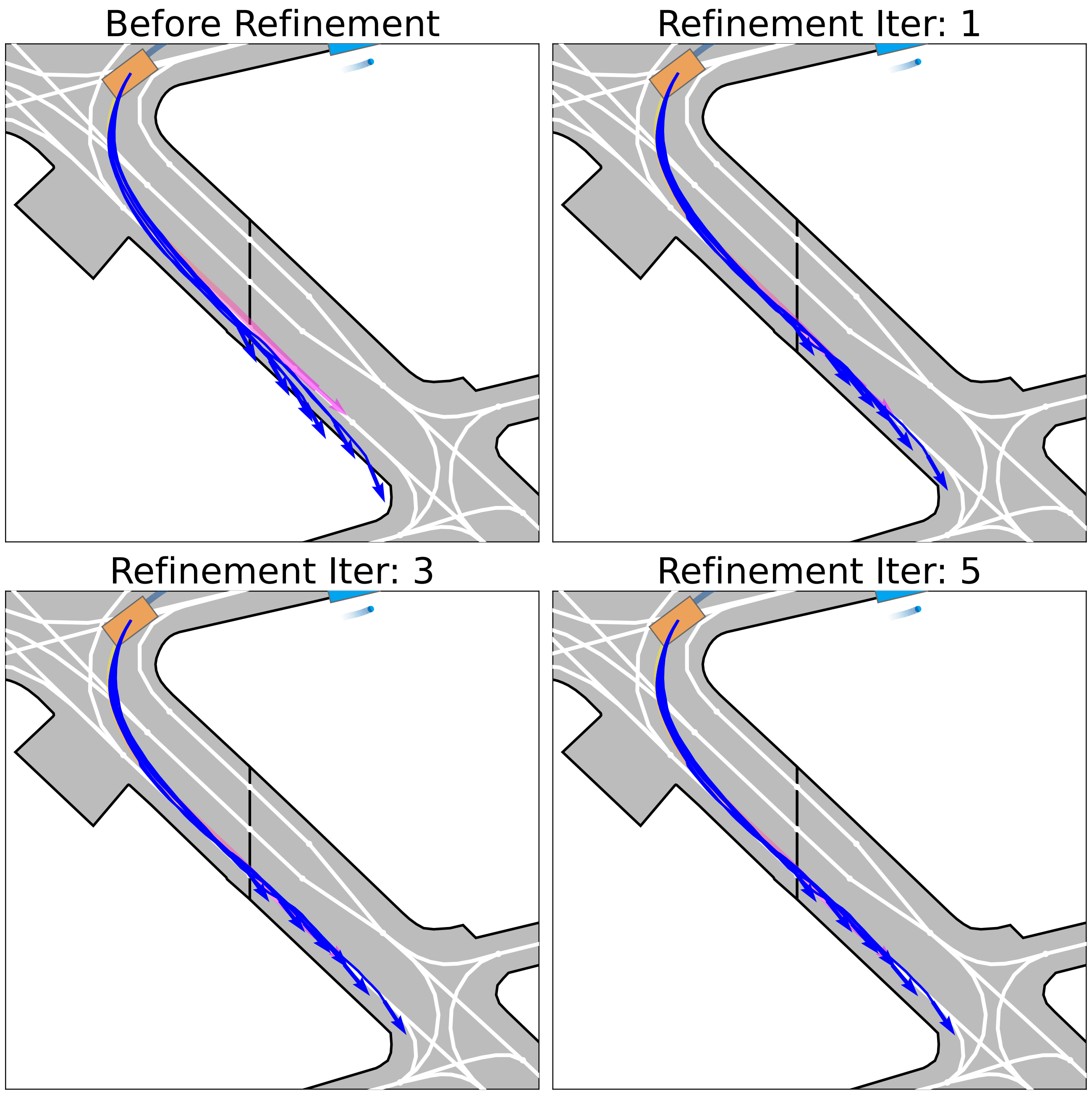

Figure 15. Visualization results. The dark blue arrows are multi-nodal predictions of the agent by model and the pink arrow is the ground truth future trajectory respectively. All trajectories get closer to the ground truth after refinement.

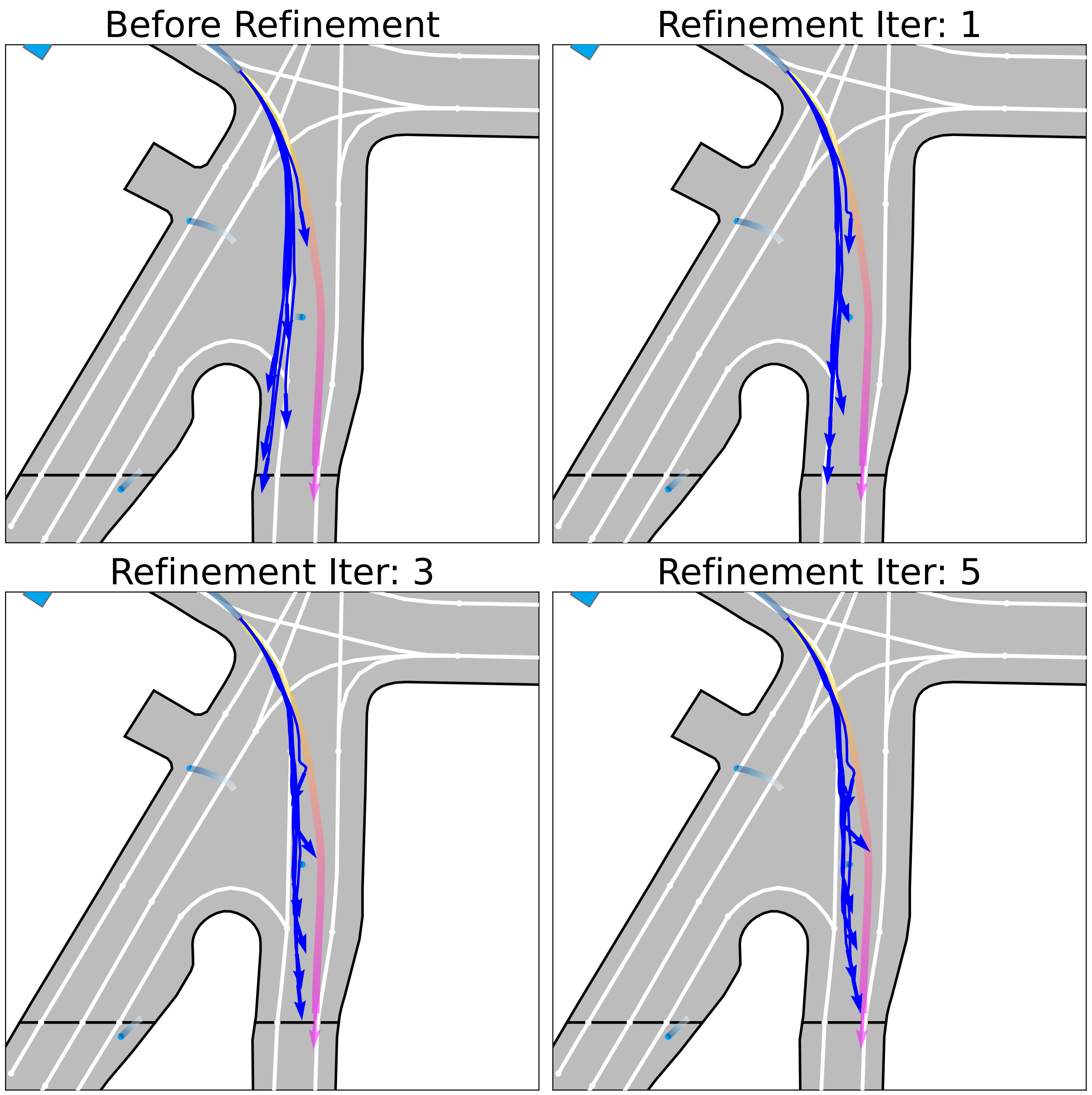

Figure 16. Visualization results when the future trajectory of a pedestrian is predicted. The dark blue arrows are multi-nodal predictions of the agent by model and the pink arrow is the ground truth future trajectory respectively. All trajectories get closer to the ground truth after refinement.