Diffusion-Based Particle-DETR for BEV Perception

Supplementary Material

In this document we provide additional reasoning, results, and comprehensive details supporting our method.

Appendix A. Label Ambiguity

Label ambiguity arises when the matching between predictions and targets depends on the sampling of the references. Figure 4 showed an example where we match using the total distance. In Figure 9 we provide also a case where the matching is by index and label ambiguity is present. We assume there are N targets $\mathbf{y}_1, ..., \mathbf{y}_N$ and N predictions $\mathbf{b}_1, ..., \mathbf{b}_N$, and predictions $\mathbf{b}_i$ is matched with target $\mathbf{y}_i$.

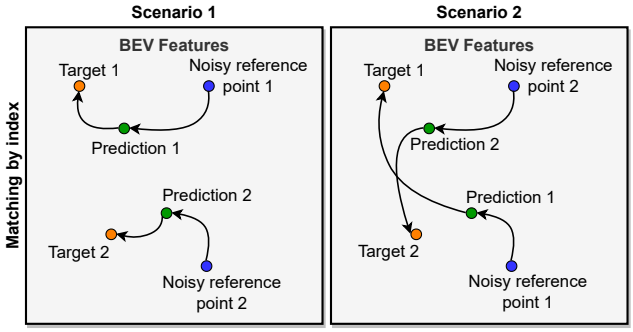

Figure 9. **Label ambiguity when matching by index**. Permuting the initially sampled reference points causes the matching to change which makes training unstable.

When matching by index, suppose reference $\mathbf{r}_i$ has BEV coordinates (x, y). The model either looks up the BEV features at (x, y) or interpolates the queries at (x, y) and produces a prediction which is matched to $\mathbf{y}_i$. However, if $\mathbf{r}_j$ is sampled at location (x, y) the prediction will be the same, but the target will be $\mathbf{y}_j$. This confuses the model because the same features at (x, y) can have different targets.

To assess the impact that ambiguous targets may have on the results, we study a simple toy task. We fix a single random image $\mathbf{I} \in \mathbb{R}^{C,H,W}$ and construct a network with the following forward pass:

- First, the image is passed through two convolution layers that keep the output size the same as the initial size.
- Then, we look up, i.e. interpolate, the features at a number of reference points, provided as an additional input.
- The resulting features are processed by two linear layers after which new 2D locations are returned.

Thus, our network takes in an image and some reference points, and returns new 2D locations as output. The loss function is the simple ℓ_1 loss between the predictions and a number of fixed targets. Our experiments suggest that when we sample the input reference points randomly, and there are only a few of them, the resulting label ambiguity is *limiting* and prevents the network from overfitting, even on a single image. This is the case when there are fewer or equal random references than the number of targets. In Figure 4 we have 10 targets. With only 10 random references, the model does not converge, even if we let it run indefinitely.

With more references than targets, the model does manage to converge, with the convergence speed depending on the number of references. This is because only some predictions are used in the loss function, which makes predictions more localized to where their reference points start from. This behaviour does not exist if the references are fixed across training iterations, in which case the model always converges to a loss of zero, irrespective of whether it finds all targets or only some of them.

In a deterministic setup more references only speed up training. But when they are random, it becomes *necessary* to have many of them in order to converge.

Appendix B. Implementation

Here we provide additional information about the implementation and the experiments. Table 3 contains the training hyperparameters, while Algorithms 1 and 2 provide PyTorch-like pseudocodes for the training and testing logic. Most function names are borrowed from [9].

For the implementation, our codebase is based on that of BEVFormer [34]. We train all models for 24 epochs on the NuScenes dataset [4] on 8 NVidia A100 GPUs, while the evaluation is always performed on a single GPU. At both training and test time the batch size is set to 1.

Diffusion box updates. The original DiffusionDet [9] only works with axis-aligned boxes. For the baseline, we re-implement and modify it to use rotated boxes. Inspired by [66], each stage of the decoder outputs the elements $(\delta c_x, \delta c_y, c_z, \delta w, \delta h, l, \delta\theta, v_x, v_y)$, which are applied to the input boxes (c_x, c_y, w, h, θ) to produce the updated boxes at the current stage $(c'_x, c'_y, w', h', \theta')$ as follows:

$$
\begin{aligned}
\bar{w} &= w\cos\theta + h\sin\theta \\
\bar{h} &= w\sin\theta + h\cos\theta \\
c'_x &= \bar{w}\delta c_x + c_x \\
c'_y &= \bar{h}\delta c_y + c_y \\
w' &= \exp(\delta w)w \\
h' &= \exp(\delta h)h \\
\theta' &= \theta + \delta\theta.
\end{aligned}
$$

Algorithm 1: Particle-DETR Training

```
# Inputs
   # F: BEV features, (C, H, W)
   # fixed_queries:  queries over which to
               interpolate, (N, C)
   # GTs:  ground-truth boxes, (N, C)
   # scale:  the signal-to-noise ratio

# Extract object centers in BEV
GT_centers = GTs[..., :2]

# Pad references up to a desired number
ref_points = pad_refs(GT_centers)

# Scale and apply diffusion
ref_points = (2 * ref_points - 1) * scale
diff_time = randint(0, T)
eps = normal(mean=0, std=1)
ref_points = sqrt(alpha_cumprod(diff_time))
            * ref_points + sqrt(1 -
            alpha_cumprod(diff_time)) * eps

# Interpolate the queries
queries = grid_sample(fixed_queries,
            ref_points)
keys = F
values = F

# Call the decoder and compute loss
pred_boxes = decoder(queries, keys, values,
            ref_points, diff_time)
loss = set_prediction_loss(GTs, pred_boxes)
```

Algorithm 2: Particle-DETR Inference

```
# Inputs
   # F: BEV features, (C, H, W)
   # fixed_queries:  queries over which to
               interpolate, (N, C)

# Prepare random references
eps = normal(mean=0, std=1)
ref_points = normalize(eps)
# DDIM times [(T-1, T-2), ..., (0, -1)]
times = reversed(linespace(-1, T, steps))
time_pairs = list(zip(times[:-1], times[1:])

# Interpolate the queries
queries = grid_sample(fixed_queries,
            ref_points)
keys = F
values = F

all_preds = []
for (t_now, t_next) in time_pairs:
   pred_boxes, queries = decoder(queries,
               keys, values, ref_points,
               t_now)
   all_preds.append(pred_boxes)
   ref_points = ddim_step(ref_points,
               pred_boxes, t_now, t_next)
   ref_points = ref_renewal(ref_points)

# Filter predictions
preds = nms(all_preds)
preds = radial_suppression(preds)
```

GIoU loss. Unlike the original DiffusionDet [9], in our implementation we do not use the GIoU loss [51] because its implementation for rotated boxes with supported back-propagation is non-trivial. We leave the investigation of how similar metrics can be used as losses for future work.

Setting	Value
Num. learnable object queries	900
Num. references at test time	Variable
BEV size, (H, W, C)	$(200, 200, 256)$
Optimizer	AdamW
Signal-to-noise ratio	2
DDIM steps [56]	3
NMS discard threshold	0.1
Min. confidence threshold	0.02
Radial suppression threshold	0.5
Query formation	Interpolation
Matching type	*Simple* many-to-one
BEV encoder	BEVFormer-Base [34]
Decoder type	DeformableDETR [73]
Training epochs	24
LR schedule	Cosine, $2 \times 10^{-4} \rightarrow 10^{-6}$
Clip gradient 2-norm	35
Batch size	1

Table 3. **Particle-DETR training hyperparameters**. Additional hyperparameters follow BEVFormer-Base [34].

SimOTA. When evaluating the simOTA matching strategy [20] we remove the cost matrix masking which is present in DiffusionDet [9]. There, additional cost is added to those predictions whose center does not fall close enough to or within a target box, which effectively prevents these predictions from ever being matched. In our setup, this masking introduced a large amount of instability. Hence, we utilize simOTA using the raw cost matrix, where all predictions are considered as potential matchings to all targets.

Appendix C. Additional Experiments

Here we provide additional experimental results. All results and plots are from the NuScenes [4] validation dataset, unless otherwise noted.

Deterministic and random references. Our Particle-DETR provides rich opportunities to tweak the test-time performance after training. Once the model is trained, one can freely change the hyparparameters governing the inference behaviour. First, we assess how the joint training with two sets of references, fixed and random, affects performance. Once the model from this joint setup is trained, we can evaluate with both sets of references or with either one of them. Table 6 shows the performance when evaluating

DDIM steps	Diff. queries	mAP ↑	NDS ↑
1	300	0.4192	0.5301
	500	0.4192	0.5302
	700	0.4202	0.5306
	900	0.4211	0.5310
	1200	0.4208	0.5304
	1500	0.4208	0.5302
3	300	0.4109	0.5245
	500	0.4155	0.5275
	700	0.4171	0.5285
	900	0.4188	0.5289
	1200	0.4184	0.5285
	1500	0.4192	0.5290
5	300	0.4164	0.5275
	500	0.4190	0.5288
	700	0.4201	0.5295
	900	0.4198	0.5290
	1200	0.4196	0.5292
	1500	0.4197	0.5293
7	300	0.4172	0.5281
	500	0.4194	0.5289
	700	0.4203	0.5297
	900	0.4199	0.5290
	1200	0.4201	0.5292
	1500	0.4189	0.5287
9	300	0.4179	0.5280
	500	0.4192	0.5290
	700	0.4198	0.5294
	900	0.4197	0.5291
	1200	0.4186	0.5287
	1500	0.4184	0.5284

Table 4. **Joint effects of DDIM steps and diffusion object queries when using both diffusion and fixed queries.** With both query sets, adding more random queries or DDIM steps does not have any noticeable effect. In practice, one could use just a single DDIM step with a variable number of queries.

DDIM steps	Diff. queries	mAP ↑	NDS ↑
1	300	0.3540	0.4936
	500	0.3907	0.5140
	700	0.4046	0.5208
	900	0.4105	0.5242
	1200	0.4143	0.5261
	1500	0.4162	0.5273
3	300	0.3755	0.5038
	500	0.3944	0.5141
	700	0.4032	0.5191
	900	0.4076	0.5208
	1200	0.4114	0.5232
	1500	0.4127	0.5242
5	300	0.3930	0.5122
	500	0.4062	0.5201
	700	0.4111	0.5234
	900	0.4133	0.5250
	1200	0.4144	0.5253
	1500	0.4154	0.5263
7	300	0.3999	0.5164
	500	0.4096	0.5224
	700	0.4123	0.5241
	900	0.4144	0.5257
	1200	0.4148	0.5259
	1500	0.4154	0.5265
9	300	0.4019	0.5175
	500	0.4110	0.5237
	700	0.4134	0.5249
	900	0.4149	0.5259
	1200	0.4148	0.5261
	1500	0.4147	0.5258

Table 5. **The joint effects of DDIM steps and diffusion object queries when utilizing diffusion queries only.** Here, we explore whether adding more DDIM steps or more diffusion queries improves performance. When using only diffusion queries, more DDIM steps and more queries improve performance.

with both sets. The performance is statistically-significant and outperforms BEVFormer on all metrics.

We analyze how each reference set contributes to the results. To do this, we look at the self-attention values throughout the decoder layers. When evaluating with both query sets, fixed queries from the first decoder layer spend about 94% of their attention on other fixed queries, whereas the diffusion queries, associated with the random references, spend 82.5% of their attention on the fixed queries. This imbalance is rectified in the subsequent decoder layers. Particularly, in the last decoder layer both query sets spend about 50% of their attention onto the other query set.

When evaluating the final model with both static and random references we find that increasing the number of DDIM steps or the number of random queries does not have a significant effect, presumably because the queries corresponding to the static references are more important. This is highlighted in Table 4. The mAP and NDS results are slightly higher than if we are using the static queries only, as in Table 2, showing that the additional queries corresponding to

Metric	BEVFormer-Base	BEVFormer-Enh (ours)
NDS ↑	0.5168	**0.5287** (0.0003)
mAP ↑	0.4154	**0.4184** (0.0006)
mATE ↓	0.6715	**0.6386** (0.0009)
mASE ↓	0.2738	**0.2686** (0.0002)
mAOE ↓	0.3691	**0.3362** (0.0009)
mAVE ↓	0.4179	**0.3688** (0.0013)
mAAE ↓	0.1981	**0.1931** (0.0007)

Table 6. **Performance of BEVFormer-Enh**. We compare BEVFormer-Enh, evaluated with both static and random references, to the fully-deterministic BEVFormer. For our method, we repeat the evaluation 10 times and report the mean values. The standard deviations are shown in parentheses.

DDIM steps	Diff. queries	mAP ↑	NDS ↑
1	300	0.3540	0.4931
	500	0.3899	0.5125
	700	0.4042	0.5202
	900	0.4108	0.5239
	1200	0.4142	0.5260
	1500	0.4164	0.5272
3	300	0.3746	0.5031
	500	0.3952	0.5143
	700	0.4034	0.5187
	900	0.4080	0.5214
	1200	0.4113	0.5238
	1500	0.4139	0.5250
5	300	0.3940	0.5128
	500	0.4070	0.5210
	700	0.4124	0.5244
	900	0.4146	0.5257
	1200	0.4159	0.5267
	1500	0.4169	0.5269
7	300	0.3999	0.5162
	500	0.4104	0.5230
	700	0.4133	0.5244
	900	0.4154	0.5263
	1200	0.4162	0.5267
	1500	0.4165	0.5270
9	300	0.4031	0.5175
	500	0.4120	0.5240
	700	0.4142	0.5248
	900	0.4156	0.5263
	1200	0.4161	0.5270
	1500	0.4165	0.5272

Table 7. **The joint effects of DDIM steps and diffusion object queries when utilizing radial suppression.** When using only diffusion queries, more DDIM steps and more queries improve performance considerably.

random references do increase performance.

What happens if we use only the random references? In that case the mAP and NDS metrics are naturally lower, because the proposed corrections should all be relative to the current reference location, which is random. There is a clear trade-off between the usage of additional DDIM steps or additional references, and performance. Tables 5 and 7 show this with and without radial suppression. We highlight that given enough random references the Particle-DETR does manage to beat BEVFormer on mAP. On NDS, it only takes 900 references and a single DDIM step to beat it.

Filtering. We also provide additional justification for why filtering is needed. Since the many-to-one matching causes multiple predictions to be stacked on top of each other we found the usage of NMS necessary. The best threshold is 0.1, which we also combine with confidence-based filtering, as shown in Table 8.

Our radial suppression replaces a confident predicted box with the weighted average of the predictions within a ball neighborhood. We tune the radius of this neighborhood. For very small values no predictions are filtered. For large values predictions from multiple different objects are filtered. The optimal occurs around 0.5 meters, as shown in Table 9. Data for the joint tuning of the radius and the NMS can be found in Table 10.

NMS threshold	Conf. threshold	mAP ↑	NDS ↑
0.1	0.02	0.3845	0.5135
	0.05	0.3818	0.5138
	None	0.3847	0.5132
0.5	0.02	0.3876	0.5129
	0.05	0.3859	0.5142
	None	0.3876	0.5128
None	0.02	0.2264	0.4305
	0.05	0.2261	0.4314
	None	0.2264	0.4303

Table 8. **Filtering using NMS and confidence thresholds**. We evaluate a Particle-DETR trained with SimOTA [20] matching without radial suppression. We set the DDIM steps to 3 and vary the NMS threshold and the score threshold. The results show that NMS is needed. Furthermore, using SimOTA matching does not result in better performance than *simple* many-to-one matching.

Reference resampling. Following DiffusionDet [9] we resample the search tokens between DDIM steps. We experimented with different strategies but found the basic one in [9] to work best, as shown in Table 11. Thus, between each DDIM step we simply resample the references corresponding to less confident predictions. Alternative strategies which we tested include resampling close to the confident predictions, resampling without applying the DDIM steps, or no resampling altogether.

Model characteristics. Our BEVFormer-Enh model has

Setting	mAP ↑	NDS ↑
$r = 0.5$ m	0.4112	0.5234
$r = 0.75$ m	0.4143	0.5257
$r = 1$ m	0.4132	0.5257
$r = 1.25$ m	0.4102	0.5242
$r = 1.5$ m	0.4073	0.5226
$r = 1.75$ m	0.4030	0.5204
$r = 2$ m	0.3971	0.5172
$r = 3$ m	0.3697	0.5014
$r = 4$ m	0.3372	0.4806

Table 9. **The effect of radial suppression**. Here, we set the NMS threshold to 0.5, the DDIM steps to 3, the score threshold to 0.02 and only vary the radius of the radial suppression. The results suggest that a small radius is best because it keeps the averaging sufficiently localized. Larger radii cause predictions corresponding to different targets to be averaged, reducing detection accuracy.

the same number of parameters, FLOPS, and FPS as BEVFormer [34]. The Particle-DETR is similar but can use more compute depending on how many DDIM steps one runs. The compute in the query interpolation depends only on the number of reference points and not on the DDIM steps.

Setting		mAP ↑	NDS ↑
NMS threshold = 0.1	$r = 0.5$ m	**0.4188**	**0.5289**
	$r = 0.75$ m	0.4173	0.5283
	$r = 1$ m	0.4138	0.5265
	$r = 1.25$ m	0.4101	0.5246
	$r = 1.5$ m	0.4066	0.5228
NMS threshold = 0.5	$r = 0.5$ m	0.4112	0.5237
	$r = 0.75$ m	**0.4130**	**0.5251**
	$r = 1$ m	0.4123	0.5250
	$r = 1.25$ m	0.4096	0.5234
	$r = 1.5$ m	0.4067	0.5222

Table 10. **The joint effects of radial suppression and NMS**. Here, we set the DDIM steps to 3 and the score threshold to 0.02 and only vary the NMS threshold and the radius of the radial suppression. The best results are achieved with strong NMS suppression and a relatively small radius.

Resampling strategy	mAP ↑	NDS ↑
Standard normal resampling [9]	**0.4156**	**0.5264**
Resample near predictions	0.4144	0.5251
No DDIM, with resampling	0.4146	0.5252
No DDIM, no resampling	0.3959	0.5130

Table 11. **Different resampling strategies**. The simple strategy of pruning the low-confident references are replacing them with new random ones works best.

Metric	Particle-DETR	BEVFormer-Enh
NDS ↑	0.5283	0.5323
mAP ↑	0.4287	0.4326
mATE ↓	0.6011	0.5904
mASE ↓	0.2600	0.2595
mAOE ↓	0.4596	0.4504
mAVE ↓	0.4125	0.4109
mAAE ↓	0.1275	0.1287

Table 12. **Nuscenes `test` set metrics**. For the Particle-DETR we use 1500 queries and 1 DDIM step, with radial suppression radius set to 0.5 and NMS threshold set to 0.1.

Appendix D. Qualitative Analysis

Here we provide additional visualizations of the predictions. Figure 10 shows an example where our Particle-DETR, using only the diffusion queries, detects very small objects which are missed by BEVFormer. In fact, the AP at 0.5 meters for traffic cones is above 0.34, whereas that

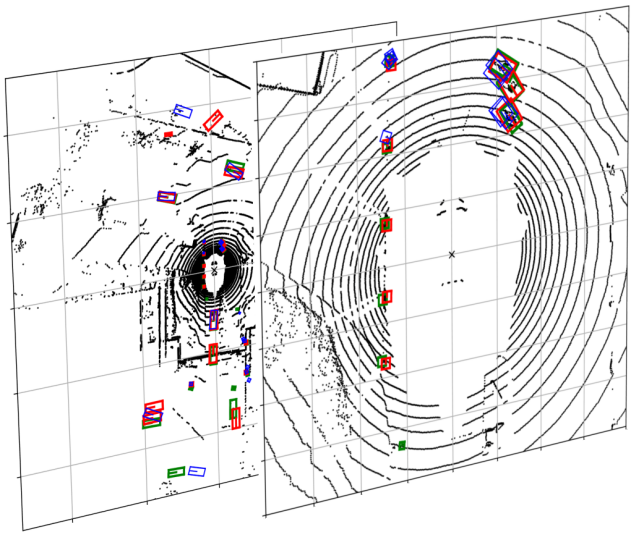

Figure 10. **Sample predictions in BEV**. Green boxes are ground-truths, red are predicted by our Particle-DETR, and blue is predicted by BEVFormer. The right figure is a zoomed-in version of the left one, centered around the ego-vehicle.

of BEVFormer is 0.28. Similar detection boosts can be observed also for cars (+1 AP point), bicycles (+2.4), motorcycles (+2.8), pedestrians (+1.8), and barriers (+9 AP points).

In general, the increased NDS results mostly from more accurate translation, orientation, and velocity. In Figure 14 we show predictions from our Particle-DETR projected onto the camera images. We highlight diverse driving conditions including bright sunshine, rain - where raindrops create localized blur in the images, and nighttime - where pixel intensity noise due to the low exposure time is present. In all these cases our method produces reasonably accurate predictions, while being a fully generative model.

Basins of attraction. We visualize the transformation of the starting references in Figure 13. In general, each GT attracts the starting references around it. This *gradient flow* is learned by the model and the many-to-one matching is a necessary condition for its existence. The basins of attrac-

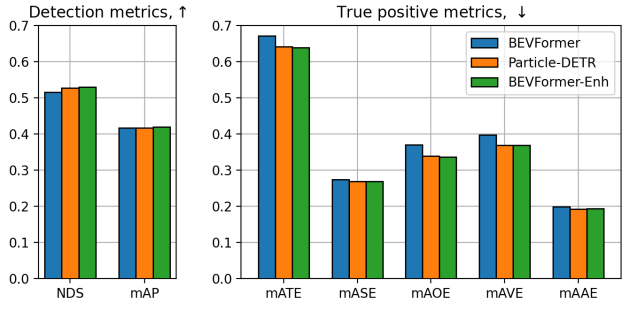

Figure 11. **Metrics comparison**. The improved NDS results from significantly lower orientation, translation, and velocity errors.

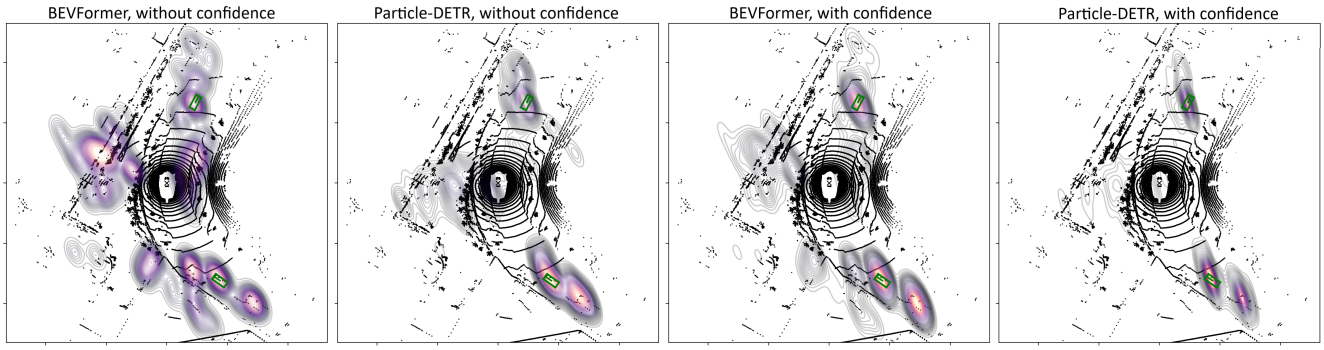

Figure 12. **Uncertainty comparison**. Our Particle-DETR produces meaningful heatmaps due to its many-to-one attractive nature.

tion are well localized and separated. We considered adding ℓ_1 regularization between the predictions and the starting references to explicitly make predictions more localized but this was not needed as such a localization property seems to develop naturally from the training setup.

The BEV is patched together from multiple camera views. Logically, it is desirable to prohibit reference points starting from one side of the ego-vehicle to be refined to the other side because a view from one side does not provide information about the opposite view. The attractive nature of the Particle-DETR more or less satisfies this constraint.

The references which start in problematic BEV regions, such as behind walls or outside of the road, are pushed to the sides of the BEV as the features at those locations do not correspond to any visible scene elements. The confidence of the corresponding predictions is near zero.

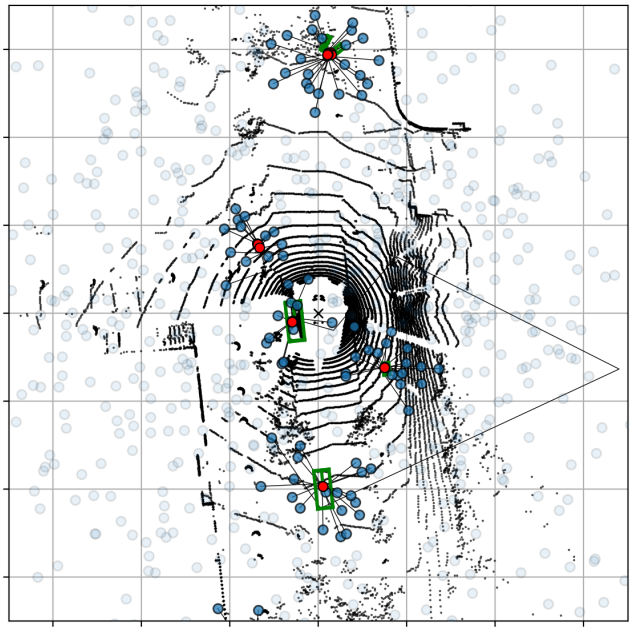

Figure 13. **Reference dynamics**. We plot the starting random references as faint blue circles. Predicted centers with sufficient confidence are shown in red. The starting references transformed to those predictions are in brighter blue. The black lines show how each reference has been modified through the six decoder stages. Better viewed zoomed.

Appendix E. Qualitative Evaluation on KITTI

In this section we qualitatively evaluate the generalization capability of our Particle-DETR on different datasets. Due to different sensor hardware, evaluating on multiple datasets requires additional adjustments. Here we showcase some results on KITTI demonstrating the generalization of our method. Compared to NuScenes, KITTI has a different sensor setup. We make the following design decisions: 1) We repeat the single frontal image across all cameras, *as if* the scene is identical from all sides of the vehicle. 2) When evaluating, we use the CAN bus and the camera embeddings *from NuScenes*. This is because the learned camera embeddings in BEVFormer are dataset-specific. 3) We do not use past frames when estimating the current BEV.

All of these are expected to affect performance negatively, yet they are necessary to deal with the different sensor settings. Fig. 15 shows that, nonetheless, our Particle-DETR detects the objects meaningfully. The dependency on dataset-specific camera embeddings is a limitation of the underlying backbone, BEVFormer, not of our proposed detection part, which uses diffusion. Fig. 16 shows additional predictions on a different scene where the model fails to detect a number of vehicles.

Appendix F. Different Backbones

Due to the BEV transformation and the size of these models, adapting the diffusion framework to other backbones, like BEVFusion, BEVFormerV2, or HoP currently remains cumbersome. Therefore, remains as future work. Here, we offer some insights from brief investigations in this direction.

We have previously experimented with combining diffusion with SparseBEV but there, because there is no explicit BEV estimated, results are inferior. Due to the geometric

Figure 14. **Particle-DETR predictions projected onto the camera images**. We show three scenes, each with six cameras around the ego-vehicle. Green boxes are ground-truths, red are predicted by our Particle-DETR. We only show predictions with confidence greater than 0.2. The predictions are relatively accurate across diverse driving scenarios, including sunny, overcast, and night-time conditions.

Figure 15. **Predictions on KITTI**. We test how the Particle-DETR trained on NuScenes generalizes to scenes in the KITTI dataset. Best viewed zoomed.

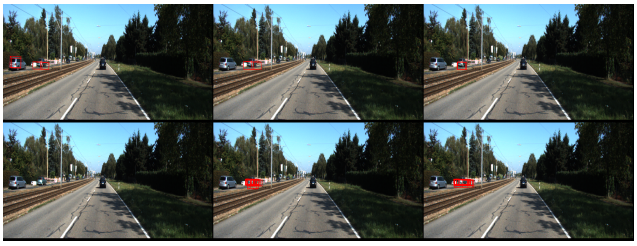

Figure 16. **Undetected objects on KITTI**. This is likely due to sensor differences between NuScenes and KITTI. The reasons are explained in Appendix. E.

projections that happen within the detector, diffused references close by may end up as predictions far away from each other. Additionally, the noisy references are in BEV coordinates but the features are obtained from individual camera coordinates. This makes predictions spread out radially from the ego-vehicle. Thus, our diffusion framework is best applied whenever a dense BEV is estimated, and where nearby world coordinates correspond to nearby BEV cells.

Appendix G. Additional Discussion

Standard DETR models are fully-deterministic. To get an uncertainty estimate over the predictions, one usually needs to explicitly modify the model architecture, for example by adding additional outputs, which stand in for the standard deviation of the box location. Here, our generative Particle-DETR is advantageous, in that a rudimentary form of uncertainty may be readily available.

First, we consider the baseline BEVFormer and we plot heatmaps over the predicted box centers, computed using kernel density estimation, as shown in Figure 12. The density (and color) at each predicted center in this plot is determined mainly by how close this point is to nearby predictions. The first heatmap shows the density over the predicted centers only, even without considering the confidence of each prediction. Since BEVFormer uses one-to-one matching, most of the predictions are quite spread apart and few of them are attracted to the same GT box. If we weight the predictions by their confidence, we get the third heatmap, which is more reasonable.

Now, we apply the same procedure to our Particle-DETR. In the second heatmap we plot the density of the predicted centers only, *before applying NMS*. Due to the many-to-one matching during training, the predictions stack, which prevents the density from being too spread apart and keeps it relatively focused on the true objects. If we further consider the predicted confidence as a weight to each point, we get the fourth heatmap where the density is even better localized.

Thus, the dynamics of how predictions are formed themselves contain information. Though for object detection we only use a few predictions, for uncertainty estimation many can be useful. We leave it for future work to investigate how to reason more formally about this opportunity.