PointBeV: A Sparse Approach to BeV Predictions

Supplementary Material

This document contains technical details about training and implementation of models (Section A), further memory consumption studies in (Section B), and additional details on our Sparse Feature Pulling module (Section C) and on our temporal model (Section D). We also detail more the sparse inference settings (Section E). Finally we display some predictions (Section F).

A. Technical Details

A.1. PointBeV Training

We train PointBeV using both image [13] and BeV augmentations [50]. For BeV augmentations, viewpoint changes (translations and rotations) are encoded in an augmentation matrix. This matrix is applied on the bounding box coordinates before building the ground-truth segmentation map, and when projecting the pillar points points in camera coordinates in the forward pass. We train static models for a maximum of 100 epochs, as they always converge earlier.

The number of training epochs is an upper bound, we get similar performance earlier ($\sim$50 epochs in low resolution with filtering, and $\sim$80 for high resolution without filtering). Moreover, PointBeV trained for 30 epochs already performs beyond previous SOTA (see table below), and each epoch is also much faster (15min/epoch for PointBeV vs. 52min/epoch for BEVFormer on a single 40GB A100 GPU with maximum model batch size).

IoU (↑) vehicle	*No vis. filtering*		*Vis. filtering*	
Backbone: EN-b4	224×480	400×800	224×480	400×800
PointBeV @30epochs	38.1	41.6	43.1	46.8

For the temporal model, PointBeV-T, we start from a static checkpoint and we add a single submanifold temporal attention layer. The temporal layer is trained using 8 past frames for fair comparisons corresponding to 2 seconds while the rest of the network is kept frozen.

A.2. Comparison with the Baselines

In Tab. 1, we take the official numbers from the papers whenever available. However, across different publications, the numbers are often reported in different settings in terms of image resolution or vehicle filtering, preventing direct comparison. Therefore, to provide a more complete view, we trained the models on the 4 different settings on which we compare them. We stress that the models are re-trained specifically for these settings, in opposition to simply evaluating the official checkpoints on different settings.

In cases where the code is available, such as for CVT [52] and Simple-BEV [13], we use it and only change the image resolution and the visibility filtering. The official CVT [52] repository already includes a metric that accounts for visibility so we simply modify the visibility applied in the loss function before retraining the models. For Simple-BEV [13], we modify the code, particularly the dataloader, to incorporate the visibility annotations. To ensure that the implementations are correct, we compare in Tab. 8 the results of our reproduction against those reported in the papers, in their proposed settings. We see that they are very similar.

When the code is not available, we replicated the method. This is the case for BEVFormer [31], which does not have an official segmentation code. For BEVFormer, we used 6 layers defined by one deformable self-attention followed by one deformable cross-attention with 4 heads and 8 offsets per points. We are comparing PointBeV with the static BEVFormer model, and for fair comparisons, we are using single-scale image features as indicated in the reproduction code of the official Simple-BEV repository. With BEVFormer, we achieved results superior to those reported in the original paper by utilizing a lower image resolution, which validates the reproduction code (see Tab. 8).

Method	Resolution	Visibility	IoU orig.	IoU reproduced
Simple-BEV [13]	448×800	with filtering	46.6	46.56
CVT [52]	224×480	with filtering	36.0	36.63
BEVFormer [31]	640×1600	with filtering	44.4	—
	448×800	with filtering	—	45.56

Table 8. **Comparison of the results obtained after training with the official code** under the setting of the paper and those reported in the papers. Our experiments reached similar results.

A.3. Parameter Count

We analyze the number of parameters for various models considered. The parameter counts were obtained from the official codes. For a fair comparison we report the number of parameters using the same ResNet-50 [14] backbone which also influences the neck network when there is one. The neck network of PointBeV considers two resolutions, returned by the backbone, and aligns them by applying bilinear interpolation to the smaller one. The channels of these resolutions are then concatenated, followed by a series of three convolutions to adjust the final channel dimension.

Method	Backbone	Neck	VT	Update	Temporal	Heads	Total
CVT [52]	8.5M	—	819k	244k	—	37.1k	9.6M
LaRa [2]	8.5M	1.9M	2.6M	4.9M	—	295k	18.2M
BEVFormer [31]	8.5M	9.5M	7.3M	—	—	442k	25.8M
PointBEV	8.5M	9.5M	291k	3.6M	564k	442k	22.9M

Table 9. **Comparison of the number of parameters of several models.** For a fair comparison we used the same backbone, i.e., ResNet-50 [14]. The column 'VT' corresponds to 'View Transform', that is any learnable operation involved in the camera-to-BEV projection (e.g., a cross-attention for CVT and LaRa [2], all deformable blocks for BEVFormer [31]). If available, BeV learnable grid parameters are accounted in the view transform column.

Method	**Backbone**	**Resolution**		
		224 × 480	448 × 800	640 × 1600
LaRa [2]	EN-b4	27	17	5
CVT [52]	EN-b4	38	12	3
Simple-BEV [13]	RN-50	11	11	10
BEVFormer [31]	RN-50	71	63	23
PointBeV	EN-b4	31	30	11
PointBeV	RN-50	31	28	17

Table 10. **Memory analysis of various models by comparing maximum batch sizes during *inference*** on a 40GB A100. 'EN-b4' refers to EfficientNet-b4 [43], and 'RN-50' to ResNet-50 [14].

B. Further memory analysis

To complement the memory analysis described in the main paper (Fig. 1, Fig. 7), we estimate the maximum training and validation batch sizes of various models on a 40GB A100 by studying the out-of-memory boundary in several forwards and backwards. The aim is to study how the models scale and at what point they reach memory saturation. During validation, the results in Tab. 10 indicate that PointBeV scales better than other models using an EfficientNet-b4 [43], achieving a 2× increase in maximum validation batch size compared to LaRa [2] and a 4× increase compared to CVT [52]. Therefore, the model is more suited for high-resolution tasks than the preceding models.

For training, as seen in Tab. 11, PointBeV can handle significantly larger batch sizes and scales better in terms of resolution compared to all other models.

C. Sparse Feature Pulling

In the context of BeV projection, the feature pulling module takes as input a grid of image features and a list of coordinates. It outputs a list of image features, corresponding to the provided coordinates. The problem is that existing interpolation modules only work with a fixed number of points per batch and per camera. This enforces models to consider more points than the number of visible points per camera. We propose a custom interpolation module that removes this limitation by introducing a batch reference table. Tab. 12 shows the benefits in terms of speed and memory of our module in standard conditions for BeV methods. We compared the native and custom modules under the standard use case. In details, we consider a 3D pillar BeV of $X \times Y \times Z = 200 \times 200 \times 8$ points, and 6 feature images, one per camera. This setting is the one we encounter in nuScenes [3]. Each feature image has 128 channels and their resolution is the down-sampled one after the backbone, i.e., a 224×480 original resolution leads to a down-sampled 28×60 resolution. Compared to its torch-based counterpart (Tab. 12), our sparse interpolation module does not calculate the features of points not visible in the cameras, resulting in faster computation and a smaller memory footprint. Note that conventional use of the torch module requires to apply after the interpolation a masking operation to remove unused features. We took this operation into account in our table. Our module demonstrates significantly lower memory and time usage. It is 2.3× faster for forward and 4.2× for backward, with over 2× less memory consumption in both cases.

Method	**Backbone**	**Resolution**		
		224 × 480	448 × 800	640 × 1600
LaRa [2]	EN-b4	5	3	1
CVT [52]	EN-b4	5*	1*	1*
Simple-BEV [13]	RN-50	8	5	2
BEVFormer [31]	RN-50	10	8	3
PointBeV	EN-b4	13	4	1
PointBeV	RN-50	26	9	3

Table 11. **Memory analysis of various models by comparing maximum batch sizes during *training*** on a 40GB A100. 'EN-b4' refers to EfficientNet-b4 [43], and 'RN-50' to ResNet-50 [14]. '*' indicates that the model has been trained without checkpointing at the backbone level for a fairer comparison between models. Checkpointing has a direct influence on the memory footprint.

	Forward		*Backward*	
Module	Mem (GiB)	Time (ms)	Mem (GiB)	Time (ms)
Naive Feature Pulling	1.9	4.3	2.8	26.0
Sparse Feature Pulling	0.9	1.9	1.4	6.2

Table 12. **Sparse Feature Pulling module memory and time footprints.** Results are for a batch composed of a single example.

D. Temporal model

One goal of the submanifold temporal attention module is to apply attention to a reduced combination of points. Therefore, we established a threshold τ_{temp} at which a point is considered temporally active or inactive. To asses the number of points filtered at each time step, we used a static model and analyzed the distribution of logits in the prediction map

(Tab. 13). To be conservative, we set our threshold at the inflection point of the static model, i.e., at $\tau_{temp} = \text{sigm}(-5)$. Given the threshold considered, our module processes on average only one tenth of the points in the past, which divides the calculations by 10 compared with naive temporal attention in torch.

τ_{temp}	sigm(0)	sigm(−3)	sigm(−5)	sigm(−6)	sigm(−7)	sigm(−8)	sigm(−9)
# Points	725	1932	3895	6091	11266	27084	38442
IoU	37.0	39.1	39.9	40.0	40.2	40.2	40.2

Table 13. **Analysis of the number of activated points according to the applied temporal threshold τ_{temp}.** To do this, we take a static model trained at 224×480 image resolution without visibility filtering having 39.9 IoU and calculated how many points were above the considered threshold. The corresponding IoU is the temporal model evaluated using the temporal threshold.

E. Sparse Inference

The introduction of sparse inference highlighted several parameters, such as the size of the densification patch and the threshold for considering a coarse point as an anchor point. To thoroughly analyze the impact of these metrics on sparse evaluation, we retrieved the checkpoint of a model trained without the visibility filter at a resolution of 224×480 with an associated 38.09 IoU, and then we varied the aforementioned parameters during inference. By default, the number of coarse points associated with its model is $N_{coarse} = 2500$, the threshold $\tau = 0.1$, the fine patch size is $k_{fine} = 9$.

E.1. Fine patch size

When varying the size of the densification patch, we observe that smaller patches lead to lower results, mainly due to having only 1/16 of the total points in the coarse pass (Tab. 14). However, considering the geometric pattern of the coarse pass, the IoU plateaus beyond a certain window size. This is logical, as the spacing between two coarse points becomes less than half of the window size.

Patch size	1	3	5	7	9	11	13
N_{fine}	92	826	1,743	2,277	2,839	3,414	4,009
Memory (MB)	461	462	472	481	493	507	522
IoU vehicle (↑)	2.6	22.0	37.3	38.0	38.1	38.1	38.1

Table 14. **Sparse evaluation on the nuScenes [3] validation set** of our model at resolution 224×480 without visibility filtering, using different patch sizes for the fine pass. Coarse sampling is a regular 50×50 grid (2,500 points). The reported memory is the maximum memory allocated calculated without the backbone.

Qualitatively, the larger the patch size k_{fine}, the greater the number of activated points in the fine pass. Beyond a certain point, when half the size of the patch exceeds the distance between two neighboring points in the first pass, we have a prediction that nearly does not change anymore. Considering the trade-off between the number of points considered, the patch size, and the final IoU, we have thus selected a default patch size of $k_{fine} = 9$, resulting in an IoU of 38.09 as indicated in Tab. 1.

E.2. Anchor threshold

We also vary the anchor threshold τ at which a point is considered an anchor point (Tab. 15). It is noteworthy that the distribution of activation scores for the points exhibits a relatively long tail. Many points have an activation threshold between 0 and 0.1, and at 0.1, already 71% of the points have been filtered. If the threshold is too high, there is a risk that certain regions will not be densified, leading to a drop in IoU. If we densify the entire map, we lose the memory value of sparse inference. This tradeoff is exhibited in Tab. 15.

Threshold τ	0.0	0.1	0.2	0.3	0.4	0.5
N_{fine}	40,000	2,839	2,212	1,868	1,627	1,430
Memory (MB)	1577	493	479	472	468	466
IoU vehicle (↑)	38.1	38.1	38.0	37.9	37.7	37.5

Table 15. **Sparse evaluation of our model on the nuScenes [3] validation set** at resolution 224×480 without visibility filtering, using different threshold to activate anchor points before the fine pass. The reported memory is the maximum memory allocated without the backbone.

Qualitatively, as the anchor point threshold increases, fewer points are considered active during the second pass, which may result in missing important regions in the BeV (see Fig. 9). Conversely, a threshold set too low tends to regard too many points as significant, thereby losing the memory efficiency of the approach. Note that when the threshold is $\tau = 0$, even if the image is sub-sampled, as the patch size is greater than half the spacing, we end up with an image that has made predictions over the entire BEV. Considering the trade-off between the number of points considered, the anchor threshold and the final IoU, we have thus selected a default threshold factor of $\tau = 0.1$, resulting in an IoU of 38.09 as indicated in Tab. 1.

E.3. Sparse adaptative inference

We also examine the influence of the reduction factor in the coarse pass by adjusting the kernel size to cover regions between two neighboring points (Tab. 16). This analysis is directly related to Fig. 7 in the paper. It demonstrates that subsampling 1/16 of the points is sufficient to achieve results similar to a model evaluating the entire grid in a single pass. Beyond a certain reduction factor, which is related to the size of the considered objects, performance decreases.

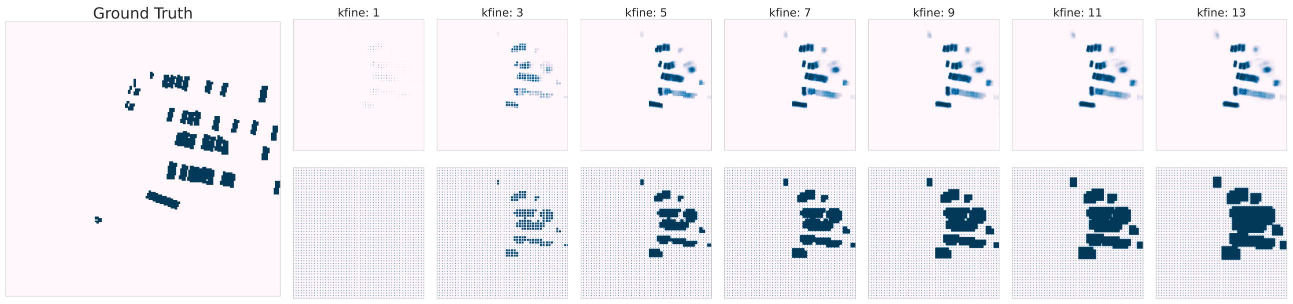

Figure 8. **Qualitative representation of different fine patch size k_{fine}.** Only the patch size applied around anchor points varies: the higher, the greater the number of points that are activated during the fine pass. The first row represents the predictions, while the second row depicts the associated binary masks. All white points outside the mask have a zero prediction. The model only considers active points in the mask.

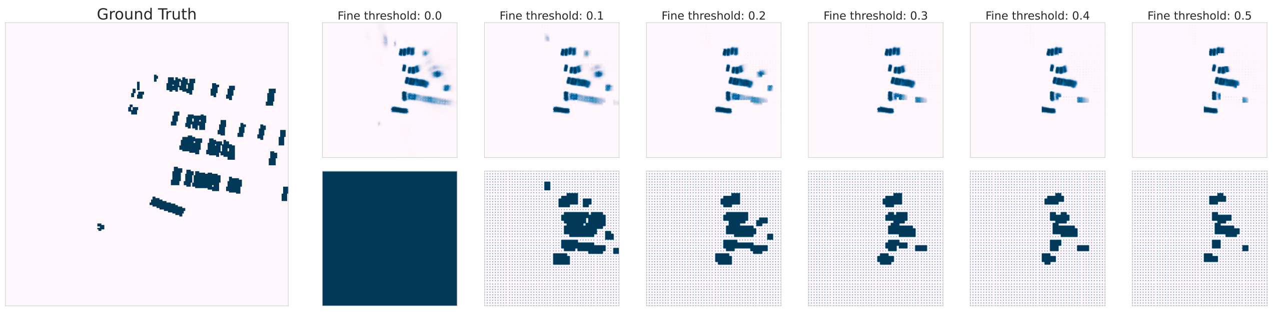

Figure 9. **Qualitative representation of different fine threshold τ_{fine}.** Only the threshold for activating anchor points varies: a lower threshold leads to a higher number of points designated as anchor points. The first row represents the predictions, while the second row depicts the associated binary masks. All white points outside the mask have a zero prediction. The model only considers active points in the mask.

Subsample factor S_k **Densification size** k_{fine}	1 —	2 3	4 5	8 7	16 9	32 13	64 17
N_{coarse}	40,000	19,880	10,000	4,900	2,500	1,255	625
N_{fine}	0	1,270	1,640	1,917	2,017	2,245	2,349
Memory (MB)	2379	1267	640	442	442	442	442
IoU vehicle (↑)	44.0	44.1	44.0	44.0	43.7	42.5	39.1

Table 16. **Sparse evaluation of our model on the nuScenes [3] validation set** at resolution 224×480 without visibility filtering, using different reduction factors to sample regular coarse points on the BeV grid while adapting the kernel patch size. Backbone is an EfficientNet-b4 [43] as in Fig. 7.

Qualitatively, it is observed that subsampling helps to correct predictions in certain areas of uncertainty (Fig. 10). However, if it is too high, there is a risk of missing important regions in the BeV. This is particularly the case when the factor is 64. Considering the trade-off between the number of points considered, and the final IoU, we have thus selected a default subsampling factor of $S_k = 16$, resulting in an IoU of 43.73 as indicated in Tab. 1.

E.4. LiDAR inference

We discuss in the paper an initialization of the coarse pass using LiDAR points retrieved from a sweep (Fig. 7). In the context of evaluation with a visibility filter, we demonstrate that this approach leads to better results than the standard approach or other sampling patterns (Tab. 17). Visually, it can be observed that the LiDAR pattern activates more regions than the standard pattern, even though the same activation threshold for anchor points is used, see Fig. 11. This is mainly because LiDAR point locations are better candidates than regular locations, which have to rely on the receptive field of features to contain local information.

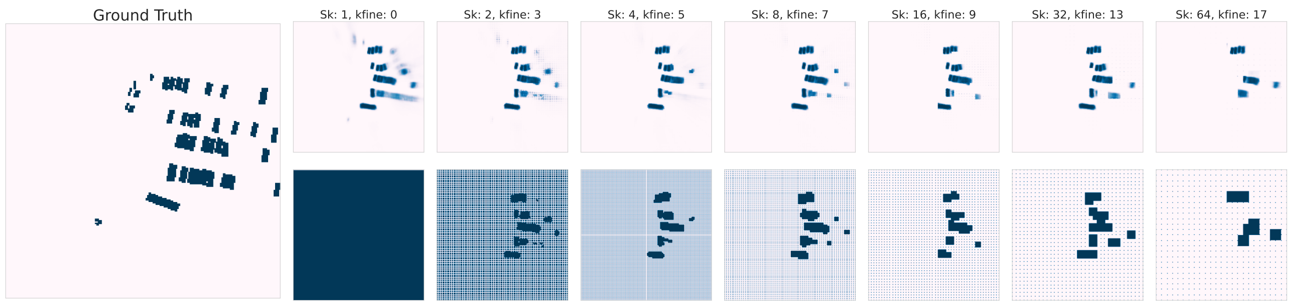

Figure 10. **Qualitative representation of different regular subsampling patterns** according to the reduction factor S_k and with adaptation of the densification patch size k_{fine}. The first row represents the predictions, while the second row depicts the associated binary masks. All white points outside the mask have a zero prediction. The model only considers active points in the mask.

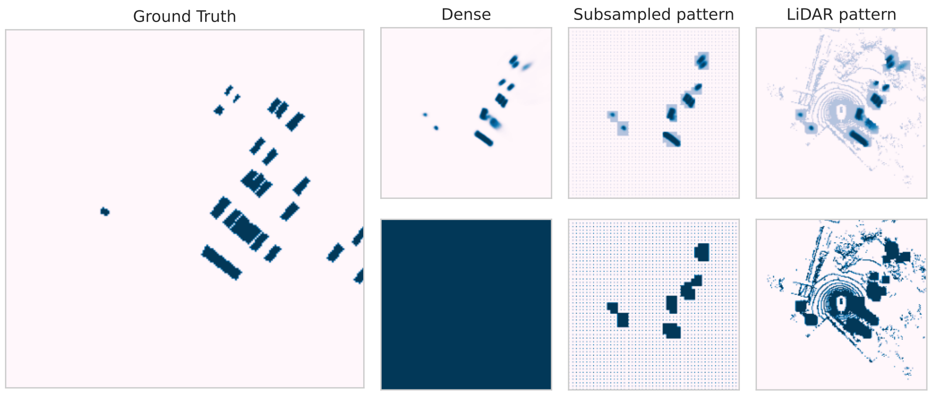

Figure 11. **Qualitative comparison of a subsampled pattern with a pattern initialized using LiDAR points.** The LiDAR points correspond to those from the sweep considered at the current timestep. The first row represents the predictions, while the second row depicts the associated binary masks. All white points outside the mask have a zero prediction. The model only considers active points in the mask.

IoU (↑) vehicle	LiDAR	Random	Regular	Dense
N_{point} (fine + coarse)	5.4k	2.7k	2.7k	40k
PointBeV	44.5	42.2	43.7	44.0

Table 17. **Comparison of different sub-sampling patterns**, showing that the lidar pattern leads to the best results. Models are trained using EfficientNet-b4 with low visibility filtering.

F. Qualitative examples

We present visualizations of PointBeV vehicle occupancy map predictions on the nuScenes validation set (without visibility filtering) in (Fig. 12) and with various lighting and weather conditions (nighttime, rainy weather, and clear weather) in Fig. 13.

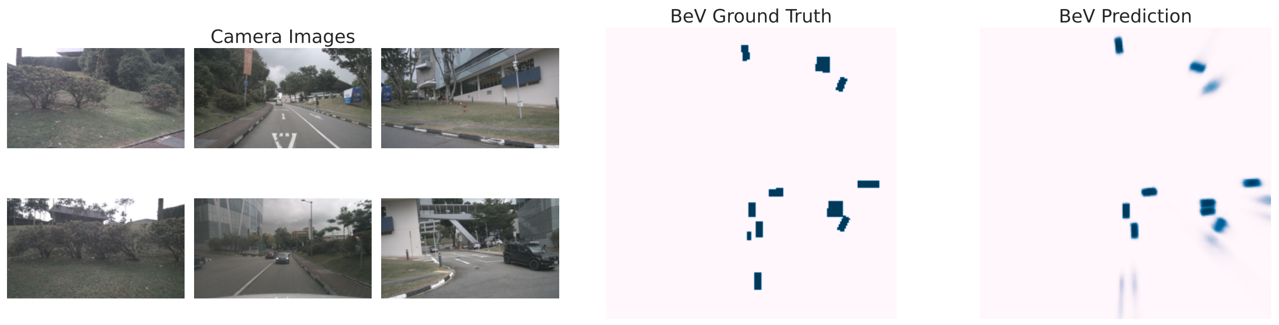

Figure 12. **Qualitative results of PointBeV's prediction on a random sample from the nuScenes validation (not cherry-picked).** The model inputs are the six cameras displayed on the left, respectively the front-left camera, front camera, front-right camera, back-left camera, back camera, and back-right camera. The ground truth and then the prediction are displayed.

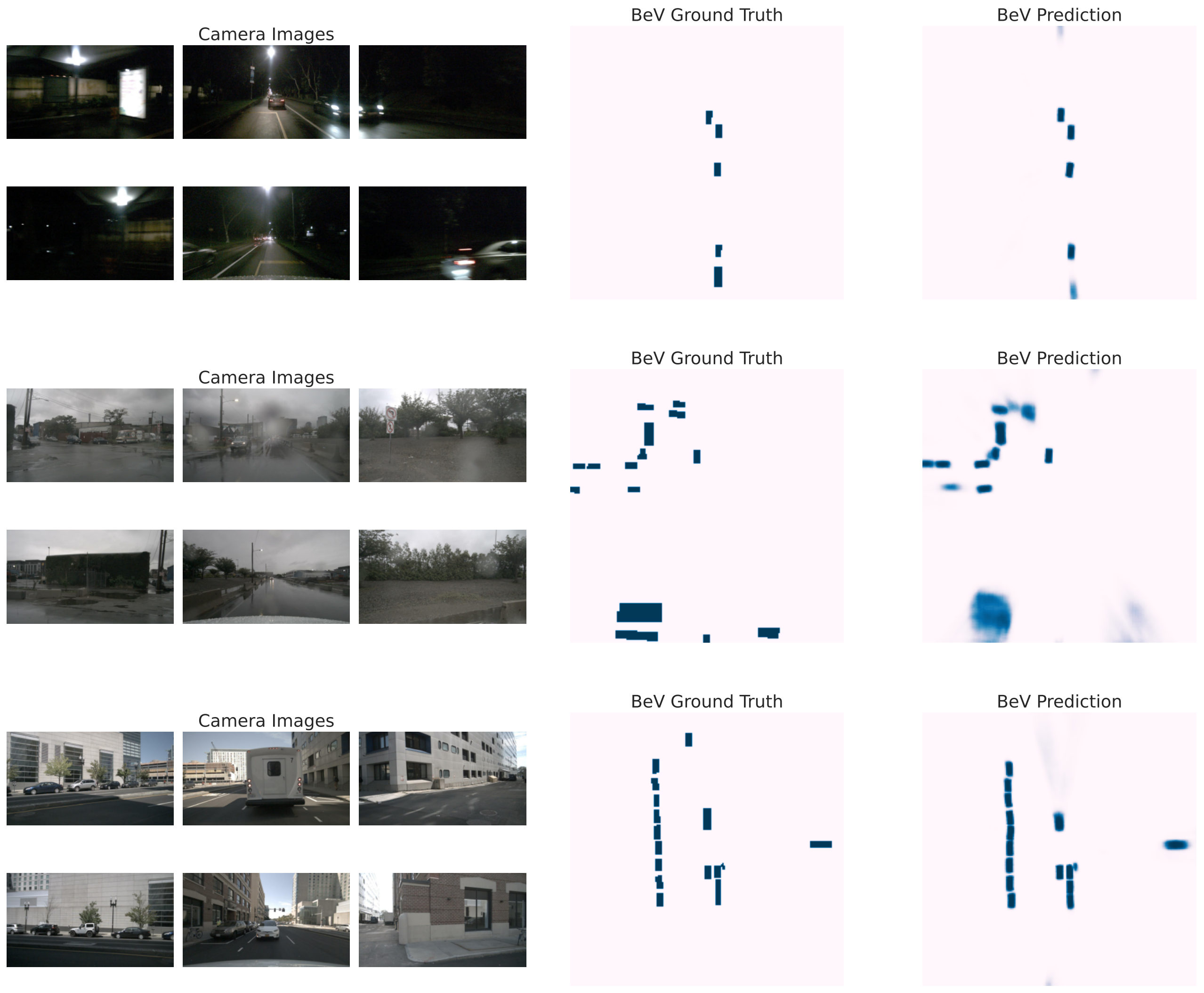

Figure 13. **Qualitative results of PointBeV's prediction on a nighttime, rainy and sunny sample from the nuScenes validation.** The model inputs are the six cameras displayed on the left, respectively the front-left camera, front camera, front-right camera, back-left camera, back camera, and back-right camera. The ground truth and then the prediction are displayed.