Supplementary Materials for SparseFusion: Fusing Multi-Modal Sparse Representations for Multi-Sensor 3D Object Detection

Yichen Xie[1,*], Chenfeng Xu[1,*], Marie-Julie Rakotosaona[2], Patrick Rim[3], Federico Tombari[2], Kurt Keutzer[1], Masayoshi Tomizuka[1], Wei Zhan[1]
[1] University of California, Berkeley [2] Google [3] California Institute of Technology

In Sec. A, we provide additional experimental results of SparseFusion and complementary details of the experiments presented in the main paper. Then, in Sec. B, we elaborate on the details of the architecture of SparseFusion.

A. Additional Experiments

A.1. Category-wise Results

In Tab. 1, we report the performance of SparseFusion and our LiDAR-only baseline [1] for each object category in nuScenes *validation set*. SparseFusion achieves significant performance improvement for all of the object categories. In particular, the introduction of camera inputs helps to distinguish objects with similar shapes like motorcycles and bicycles.

A.2. Qualitative Results

We provide additional qualitative results in Fig. 1, where SparseFusion effectively detects most objects in the scene with the correct classification.

A.3. Experiment Details

Cross-Modality Sparse Representation Interaction We provide more details about Fig. 6 in the main paper. The orange boxes refer to the high-confidence objects detected by the fusion branch. The blue and green dots denote all instances from the LiDAR and camera branches separately even if they only have very low confidence. The blue/green lines separately connect the orange boxes and blue/green dots. We only visualize the distribution of attention for high-confidence objects detected in the fusion branches (orange). The magnitude of relationships (*i.e.*, the attention value) is represented by the darkness and thickness of the lines. More examples are visualized in Fig. 2.

Optimal Transport for Sparse Fusion (main paper Tab. 3a) We explain some details of the optimal transport strategy for sparse fusion in our ablation study. We model the distribution of LiDAR candidates as follows.

$$\mathbf{p}_L(\mathbf{q}_{L,i}) = \frac{s_{L,i}}{\sum_{i=1}^{N_L} s_{L,i}}, i = 1, 2, \dots, N_L \tag{1}$$

where $s_{L,i}$ is the classification confidence (highest category) of the i-th instance for the LiDAR detector. Similarly, the distribution of camera candidates is modeled as follows.

$$\mathbf{p}_C(\mathbf{q}_{C,j}) = \frac{s_{C,j}}{\sum_{i=1}^{N_C} s_{C,j}}, i = 1, 2, \dots, N_C \tag{2}$$

where $s_{C,j}$ is the classification confidence (highest category) of the j-th instance for the camera detector (after view transformation). We construct a cost matrix $\mathbf{C} = [c_{ij}], i = 1, 2, \dots, N_L, j = 1, 2, \dots, N_C$, where c_{ij} is the euclidean distance between the centers of the i-th LiDAR instance and j-th camera instance on the BEV plane. We solve an optimal transport between $\mathbf{p}_L(\mathbf{q}_{L,i})$ and $p_C(\mathbf{q}_{C,i})$ using the IPOT algorithm [7] which outputs an optimal transport plan $\mathbf{T}^*$, where

$$\mathbf{T}^* = arg \min_{\mathbf{T} \in \mathbf{R}_+^{N_L \times N_C}} < \mathbf{C}, \mathbf{T} > \tag{3}$$

$$s.t. \quad \mathbf{T}\mathbf{1}_{N_C} = \mathbf{p}_L, \mathbf{T}^T \mathbf{1}_{N_L} = \mathbf{p}_C \tag{4}$$

We normalize $\mathbf{T}$ for each row as $\hat{\mathbf{T}}_{ij} = \mathbf{T}_{ij} / \sum_{j=1}^{N_C} \mathbf{T}_{ij}$. Then, we concatenate LiDAR candidates $\mathbf{Q}_L$ with the weighted camera candidates $\hat{\mathbf{T}}\mathbf{Q}_C$ (matrix product) in a channel-wise manner. The output features are fed into a feed-forward network to get the N_L fused instance features, then the prediction head can get the object categories and bounding boxes based on the instance features.

B. Architecture Details

In this section, we explain the detailed structure of each module in SparseFusion. In addition, we also illustrate the query initialization process for both LiDAR and camera detectors.

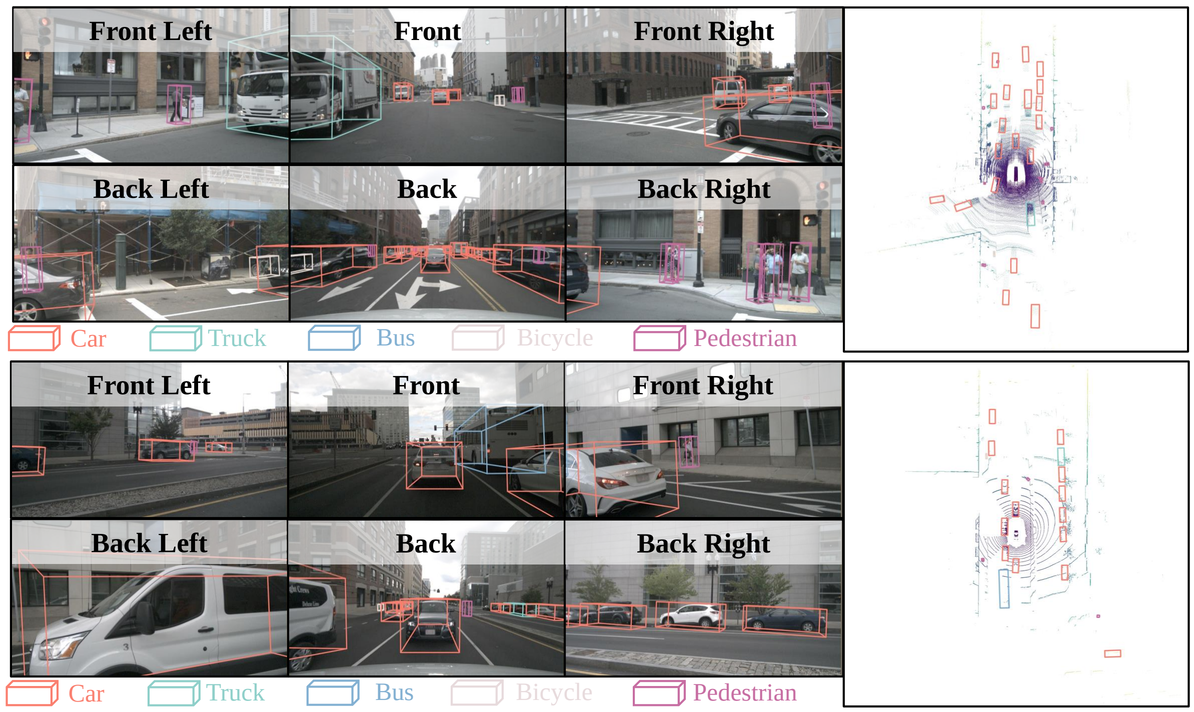

Figure 1: Qualitative results of SparseFusion on nuScenes *validation set*.

Table 1: Category-wise performance on nuScenes *validation set* including the overall NDS, mAP, and AP for each category.

Methods	Modality	NDS	mAP	car	truck	bus	trailer	const. vehicle	pedestrian	motorcycle	bicycle	traffic cone	barrier
TransFusion-L [1]	L	70.2	65.1	86.5	59.6	74.4	42.2	25.4	86.6	72.1	56.0	74.1	74.1
SparseFusion	L+C	**72.8**	**70.4**	**88.5**	**64.4**	**77.1**	**44.3**	**30.3**	**89.8**	**81.5**	**71.0**	**80.6**	**76.6**

B.1. Network Architecture

LiDAR Detector We follow TransFusion-L [1] to adopt a transformer-based LiDAR detector. The initial LiDAR queries $\mathbf{Q}_L^0$ (Sec. B.3) are passed through a self-attention module, then cross-attention is conducted with the BEV features from the LiDAR backbone. The output queries are fed into a feed-forward network to get the LiDAR candidates $\mathbf{Q}_L$. In both the self-attention and cross-attention modules, we add a positional encoding to all of the queries, keys, and values. Instead of the fixed sine positional embedding [6], we apply the learned embeddings by inputting the 2D XY locations of the queries, keys, and values on the BEV plane to an MLP encoder. A LiDAR view prediction head (Sec. B.2) is attached to the LiDAR candidates $\mathbf{Q}_L$ to get the object category as well as the 3D bounding box in LiDAR coordinates.

Camera Detector We extend Deformable-DETR [10] to the 3D object detection task. The initial camera queries $\mathbf{Q}_C^0$ (Sec. B.3) go through a self-attention module, then deformable attention is conducted with the image features, where we aggregate multi-scale image features from FPN [4] through deformable attention. In deformable attention, each query only interacts with its corresponding single-view image features. The output queries are fed into a feed-forward network to get the perspective view camera candidates $\mathbf{Q}_C^P$. As we do with the LiDAR detector, we add positional embeddings to all of the queries, keys, and values, which indicate their 2D locations on the image of the corresponding view. A perspective view prediction head (Sec. B.2) is attached to the perspective view camera candidates $\mathbf{Q}_C^P$ to get the object category as well as the 3D bounding box in camera coordinates.

View Transformation Our view transformation module consists of two parts: feature projection and multi-view aggregation. The feature projection is already described in Eq. 1 of the main paper, which encodes the camera parameters and projected boxes with two MLPs and combines them with the original instance features. The multi-view aggrega-

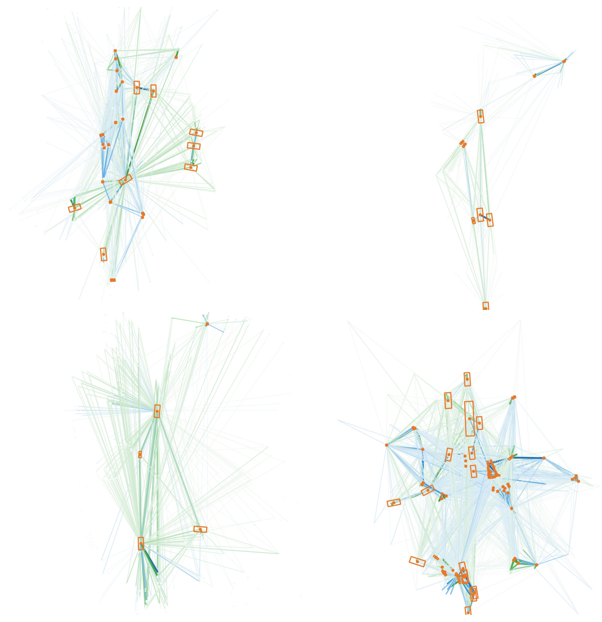
Figure 2: Instance-level feature interaction in the fusion stage. Orange boxes are objects detected after the fusion stage with high confidence in the BEV space. Blue and green dots denote all instances from the LiDAR and camera branches separately. Orange boxes are connected with blue/green dots with blue/green lines. The strength of attention is represented by the darkness and thickness of the lines.

tion is based on a self-attention module. The output instance features belonging to all the different views are put together as $\mathbf{Q}_C^L = \{\mathbf{q}_{C,i}^L\}_{i=1}^{N_C}$. They are fed into a self-attention module and feed-forward layer. For positional embeddings added to each instance feature, we take into account both the predicted box center on the image from the camera detector and the box center on the BEV plane after bounding box coordinate transformation. The 4-dimensional inputs are passed through an MLP to get the positional embedding for each instance feature. The updated queries serve as the camera candidates $\mathbf{Q}_C = \{\mathbf{q}_{C,i}\}_{i=1}^{N_C}$. We also attach a LiDAR view prediction head to the candidates to predict the object category and 3D bounding boxes in the LiDAR coordinates.

Fusion Branch We process the LiDAR candidates $\mathbf{Q}_L = \{\mathbf{q}_{L,i}\}_{i=1}^{N_L}$ and camera candidates $\mathbf{Q}_C = \{\mathbf{q}_{C,i}\}_{i=1}^{N_C}$ with two separate modules $f_L(\cdot), f_C(\cdot)$, each consisting of a fully-connected layer and layer normalization. Then, we concatenate the candidates as $\mathbf{Q}_{LC} = \{\mathbf{q}_{LC,i}\}_{i=1}^{N_L+N_C}$. Afterward, $\mathbf{Q}_{LC}$ is fed into a self-attention module and a feed-forward network to get the final fused instance features $\mathbf{Q}_F$. In the self-attention module, we also add a learned positional embedding to the instance features by encoding the XY box centers on the BEV with an MLP. Finally, we attach a LiDAR view prediction head to $\mathbf{Q}_F$ to predict the object category and 3D LiDAR view bounding boxes as the final results.

Algorithm 1: Geometric Transfer

```
Input: Multi-scale image feature map of view $v$:
       $\mathbf{F}_{C,v} = [\mathbf{F}_{C,v}^l]_{l=0}^L$, sparse depth map of view
       $v$: $\mathbf{D}_v$.
Output: Multi-scale depth-aware image feature
        map of view $v$: $\hat{\mathbf{F}}_{C,v} = [\hat{\mathbf{F}}_{C,v}^l]_{l=0}^L$
1 $\mathbf{F}_D = \text{Stem}(\mathbf{D}_v)$
2 $\hat{\mathbf{F}}_{C,v} = []$
3 for $l = 1, 2, \dots, L$ do
4     $\mathbf{F}_D = \text{Residual-Block}_l(\mathbf{F}_D)$
5     $\mathbf{F}_D = \text{Concatenate}(\mathbf{F}_D, \mathbf{F}_{C,v}^l)$
      /* channel-wise concatenation */
6     $\mathbf{F}_D = Conv_{3\times3}^l(\mathbf{F}_D)$
7     Append $\mathbf{F}_D$ to $\hat{\mathbf{F}}_{C,v}$ as $\hat{\mathbf{F}}_{C,v}^l$.
8 Return $\hat{\mathbf{F}}_{C,v}$
```

Geometric Transfer We project the LiDAR point clouds to multi-view images with camera parameters to get the sparse depth maps (200×112 for nuScenes) for each view. We combine the original multi-level image features from FPN [4] with the sparse depth map to obtain the multi-level depth-aware image features as shown in Alg. 1, where: L is the scale level number ($L = 4$ in our experiments); Stem$(\cdot)$ is a stem block composed of a 3×3 convolution, batch normalization, and a ReLU activation; Residual-Block$(\cdot)$ is the basic residual block in ResNet-18 [2] with stride 2 for downsampling. Since we have multi-view images describing the surrounding scene, we run Alg. 1 separately for each view with the shared network parameters.

Semantic Transfer Given the dense BEV features $\mathbf{F}_L \in \mathbb{R}^{H\times W\times C}$, only a few positions are indeed covered by the LiDAR point clouds. For a position $(x_j, y_j), x_j \in \{1, 2, \dots, W\}, y_j \in \{1, 2, \dots, H\}$ on the BEV feature map occupied by point clouds, we denote the median height of the points in this pillar (x_j, y_j) as z_j. We project all these $\{(x_j, y_j, z_j)\}$ from LiDAR coordinates to the multi-view images. We fetch these image features at these positions (max-pooling to aggregate multi-scale image features), and we combine them with the original corresponding BEV features through element-wise addition. The added features serve as the queries to interact with the multi-scale image features through a deformable-attention module and a feed-forward network. We add the positional embeddings, which are the 2D locations on the images, to the queries, keys, and values. This process is run separately for images of each view. If (x_j, y_j, z_j) can be projected to multiple views, we perform max-pooling to aggregate the updated queries from multiple views. Each updated query replaces the original BEV features $\mathbf{F}_L$ at (x_j, y_j) to obtain the semantic-aware BEV features $\hat{\mathbf{F}}_L$, which will be used for the query initialization of the LiDAR detector (Sec. B.3).

B.2. Prediction Head

We use two different prediction heads for 3D objects in the perspective view and the LiDAR view.

Perspective View Head The perspective view prediction head is designed for the camera detector to detect objects in the camera coordinates. The head includes six independent MLPs as follows:

1. It predicts the category of each object. The output dimension is the number of object categories, denoting the confidence of each category.
2. For the image of each view, it regresses the offset of the projected center of each object in the image from the reference points indicated by the positional embedding. The output dimension is two, denoting the XY coordinate separately.
3. For the image of each view, it estimates the depths of each object. The output dimension is one.
4. It regresses the logarithms of the XYZ scale of the 3D bounding box. The output dimension is three.
5. It predicts the angle of each object around the vertical axis (Y-axis in the camera coordinate). The output dimension is two, denoting the $\sin$ and $\cos$ of this angle.
6. It predicts the velocity in the horizontal plane (XZ-plane in the camera coordinate space). The output dimension is two,denoting the velocities along the X-axis and the Z-axis.

LiDAR View Head The LiDAR view prediction head is designed to detect objects in the perspective view. The same head is used for the LiDAR detector, view transformation, and the fusion branch with different network weights. The head includes six independent MLPs as follows:

1. It predicts the category of each object. The output dimension is the number of object categories, denoting the confidence of each category.
2. It regresses the offset of the center of each object on the BEV plane from the reference points indicated by the positional embedding. The output dimension is two, denoting the XY coordinate separately.
3. It regresses the height of each object center. The output dimension is one.
4. It regresses the logarithms of the XYZ scale of the 3D bounding box. The output dimension is three.
5. It predicts the angle of each object around the vertical axis (Z-axis in the LiDAR coordinate space). The output dimension is two, denoting the $\sin$ and $\cos$ of this angle.
6. It predicts the velocity in the horizontal plane (XY-plane in the LiDAR coordinate). The output dimension is two, denoting the velocities along the X-axis and the Y-axis.

B.3. Query Initialization

We follow CenterFormer [9] and TransFusion [1] to initialize our queries using a heatmap, which helps to accelerate the convergence and reduce the number of queries.

Initialization for LiDAR Detector We splatter the bounding box centers on the BEV onto a category-aware heatmap $\mathbf{Y} \in [0,1]^{H\times W\times K}$ [3, 8], where K is the category number, with a Gaussian kernel $\mathbf{Y}_{x,y,k_i} = \exp\left[\frac{(x-c^L_{x,i})^2+(y-c^L_{y,i})^2}{2\sigma_i^2}\right]$, where k_i is the category of the i-th object, $(c_{x,i}, c_{y,i})$ is its center on the BEV, and σ_i is a standard deviation related to the object scale as done in [1]. The heatmap is calculated for each object separately, and we combine the multiple-object heatmaps by using the maximal value at each location. A dense head composed of 3×3 convolutions are attached to the BEV features $\hat{\mathbf{F}}_L$, which is augmented by the semantic transfer. Positions $\mathbf{p}^0_{L,i} \in \mathbb{R}^2, i = 1, \ldots, N_L$ on the BEV with the highest confidence scores in $\max_{k_i} \hat{\mathbf{Y}}_{x_i,y_i,k_i}$ are selected as the reference points on BEV plane, along with their categories $\{k_i\}_{i=0}^{N_L}$. The local BEV features at these positions from $\mathbf{F}_L$ are fetched. We add the local feature from $\mathbf{F}_L$ and a learnable category embedding $\{\mathbf{e}^L_{k_i}\}_{i=0}^{N_L}$ to get the initial LiDAR query features $\mathbf{Q}^0_L = \{\mathbf{q}^0_{L,i}\}_{i=1}^{N_L}$.

Initialization for Camera Detector For the camera modality, 3D box centers are projected into the multi-view images. We follow FCOS [5] to divide the objects of different sizes after projection into certain levels of multi-scale image features. We set the size thresholds to $0, 48, 96, 192, +\infty$. For each bounding box projected on the image plane, if its $\max(length, width)$ falls between the i-th and $i+1$-th threshold, the object is assigned to the i-th scale level. As we do in the LiDAR modality, corresponding projected centers of each feature level are splattered onto a heatmap. We also get the reference points

$\mathbf{p}_C^0 = \{\mathbf{p}_{C,i}^0\}_{i=1}^{N_C}$ with top confidence scores from the multi-view image features, as well as the corresponding categories $\{k_i\}_{i=0}^{N_C}$. The corresponding features from the depth-aware image features $\hat{\mathbf{F}}_C$ are added with the learnable category embedding $\{\mathbf{e}_{k_i}^C\}_{i=0}^{N_C}$ to get the initial camera query features $\mathbf{Q}_C^0 = \{\mathbf{q}_{L,i}^0\}_{i=1}^{N_L}$. It is worth mentioning that 3D objects are projected to multi-scale multi-view images, so initial queries come from the image features from different views and different scales.

References

[1] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1090–1099, 2022. 1, 2, 4

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[3] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European conference on computer vision (ECCV)*, pages 734–750, 2018. 4

[4] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 2, 3

[5] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9627–9636, 2019. 4

[6] 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. 2

[7] Yue Wang and Justin M Solomon. Deep closest point: Learning representations for point cloud registration. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3523–3532, 2019. 1

[8] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 4

[9] Zixiang Zhou, Xiangchen Zhao, Yu Wang, Panqu Wang, and Hassan Foroosh. Centerformer: Center-based transformer for 3d object detection. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVIII*, pages 496–513. Springer, 2022. 4

[10] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020. 2