# LaRa: Latents and Rays for Multi-Camera Bird's-Eye-View Semantic Segmentation — Supplementary Material —

**Florent Bartoccioni**
Valeo.ai Inria[*]

**Éloi Zablocki**
Valeo.ai

**Andrei Bursuc**
Valeo.ai

**Patrick Pérez**
Valeo.ai

**Matthieu Cord**
Valeo.ai
Sorbonne Université

**Karteek Alahari**
Inria[*]

## A Implementation details

Following common practice [1, 2, 3] we employ an EfficientNet [4] as our CNN image encoder $E$. In particular, we use an EfficientNet-B4 [4] with an output stride of 8. It extracts feature maps for each image $F_k = E(I_k) \in \mathbb{R}^{h \times w \times c}$. In practice, $h = 224/8 = 28$, $w = 480/8 = 60$ and we define $c = 128$.

For the BEV CNN, we follow Philion and Fidler [1] and use an encoder-decoder architecture with a ResNet-18 [5] as backbone. It produces features at three levels of resolutions (1:1, 1:2 and 1:8), which are progressively upsampled back to the input resolution with bilinear interpolation (first $\times 4$ for the 1:8*th* scale then $\times 2$ for the 1:2*th*). Skip connections are used between encoder and decoder stages of the same resolution.

Both MLP$_{\text{ray}}$ and MLP$_{\text{bev}}$ are 2-layer MLPs producing 128-dimensional features. Each consists of two linear transformations with a GELU [6] activation function:

$$\text{MLP}(x) = W_2 \text{GELU}(W_1 x + b_1) + b_2. \tag{5}$$

The exact specification of other modules will be available in our code upon publication.

### A.1 Attention modules

Following the original formulation and notations [7], the attention operation is defined as:

$$\text{Attn}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_K}})V \tag{6}$$

with its multi-headed extension:

$$\begin{aligned} \text{MultiheadAttn}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where head}_i &= \text{Attn}(QW_i^Q, KW_i^K, VW_i^V). \end{aligned} \tag{7}$$

with $d_q$, $d_v$, $d_k$ the dimensions of $Q$, $K$ and $V$. In practice, we use $d_{\text{model}}$, a hyperparameter, to define the dimension of the queries, keys and values for the inner attention (Equation 6) as well as $h$ the number of attention heads. More precisely, we linearly project queries, keys and values $h$ times with different projections, each with dimension $d_{\text{emb}} = d_{\text{model}}/h$. The learnable projection matrices of each head are defined as $W_i^Q \in \mathbb{R}^{d_q \times d_{\text{emb}}}$, $W_i^K \in \mathbb{R}^{d_k \times d_{\text{emb}}}$, $W_i^V \in \mathbb{R}^{d_v \times d_{\text{emb}}}$ and $W_i^O \in \mathbb{R}^{h \cdot d_{\text{emb}} \times d_v}$.

[*]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France

Our architecture integrates three attention modules [7]: (i) a cross-attention between latent vectors and input features; (ii) a sequence of self-attention on the latent vectors; (iii) a cross-attention between BEV query and latent vectors. More precisely, and with a slight abuse of notation:

**Latent-Input cross-attention**

$$\begin{aligned}&\text{latents} := \text{MultiheadAttn}(\text{LayerNorm}(\text{latents}), \text{LayerNorm}(\text{input}), \text{LayerNorm}(\text{input})) + \text{latents}\\&\text{latents} := \text{MLP}(\text{LayerNorm}(\text{latents})) + \text{latents}\end{aligned} \tag{8}$$

**Latent self-attention**

$$\begin{aligned}&\text{latents} := \text{MultiheadAttn}(\text{LayerNorm}(\text{latents}), \text{LayerNorm}(\text{latents}), \text{LayerNorm}(\text{latents})) + \text{latents}\\&\text{latents} := \text{MLP}(\text{LayerNorm}(\text{latents})) + \text{latents}\end{aligned} \tag{9}$$

**BEVquery-Latent cross-attention**

$$\begin{aligned}&\text{output} := \text{MultiheadAttn}(\text{LayerNorm}(\text{BEVquery}), \text{LayerNorm}(\text{latents}), \text{LayerNorm}(\text{latents}))\\&\text{output} := \text{MLP}(\text{LayerNorm}(\text{output})) + \text{output}\end{aligned} \tag{10}$$

In particular, the cross-attention between BEV query and latent vectors is not residual. Since the query is made of coordinates, imposing the network to predict segmentation as residual of coordinates does not make sense.

### A.2 Output embedding

In the main paper, we considered Fourier features and learned query as alternative BEV query embeddings. Here we detail both of them.

**Fourier features.** The Fourier encoding has been proven to be well suited for encoding fine positional features [7, 8, 9]. This is done by applying the following on an arbitrary input $z \in \mathbb{R}$:

$$\text{fourier}(z) = (z, \sin(f_1 \pi z), \cos(f_1 \pi z), \ldots, \sin(f_B \pi z), \cos(f_B \pi z)), \tag{11}$$

where $B$ is the number of Fourier bands, and $f_b$ is spaced linearly from 1 to a maximum frequency $f_B$ and typically set to the input's Nyquist frequency [8]. The maximum frequency $f_B$ and number of bands $B$ are hyper-parameters. This Fourier embedding is applied on the normalized coordinate grid such that:

$$Q_{\text{fourier}}[i,j] = \text{fourier}(Q_{\text{coords}}[i,j]_i) \oplus \text{fourier}(Q_{\text{coords}}[i,j]_j). \tag{12}$$

**Learned.** Another alternative, following common transformer practice [7, 10] and most notably proposed by CVT [3], is to let the network learn its query of dimension $d_{\text{bev-query}}$ from data. However, this is memory intensive as it introduces $h_{\text{bev}} \times w_{\text{bev}} \times d_{\text{bev-query}}$ additional parameters to be optimized. In other words, the number of parameters grows quadratically to the resolution of the BEV map. For experiments using learned output query embedding, we use $d_{\text{bev-query}} = 32$.

## B Evaluation details

With no established benchmarks to precisely compare model's performances, there are almost as many settings as there are previous works. Differences are found at three different levels:

- The **resolution** of the output grid where two main settings have been used: a grid of 100m×50m at a 25cm resolution [3, 11, 12, 13] and a grid of 100m×100m at a 50cm resolution [1, 3]. These settings are respectively referred as 'Setting 1' ($h_{\text{bev}} \times w_{\text{bev}} = 400 \times 200$) and 'Setting 2' ($h_{\text{bev}} \times w_{\text{bev}} = 200 \times 200$).
- The considered **classes**. There are slight differences in the classes used to train and evaluate the model. For instance, some models are trained with a multi-class objective to simultaneously segment objects such as `cars`, `pedestrian` or `cones` [11, 12, 13]. Some others

only train and evaluate in a binary semantic segmentation setting on a meta-class `vehicles` which includes `cars`, `bicycles`, `trucks`, *etc*. [1, 3]. Some works also use *instance* segmentation information to train their model where the centers of each distinct vehicle is known at train time [2]. In all of our experiments, we place ourselves in the binary semantic segmentation setting of the meta-class `vehicles`. This choice is made to have fair and consistent comparisons with our baselines [1, 3], however, it should be noted that our model is not constrained to this setting.

- The levels of **visibility** of objects. Objects selected as ground truth, both for training and evaluating the model, differ in terms of their levels of visibility. Three options have been considered: objects that are in line-of-sight with the ego car's LiDAR [11], or objects with a nuScenes visibility above a defined threshold, either 0% [1] or 40% [3].

## C Extension to driveable area segmentation task

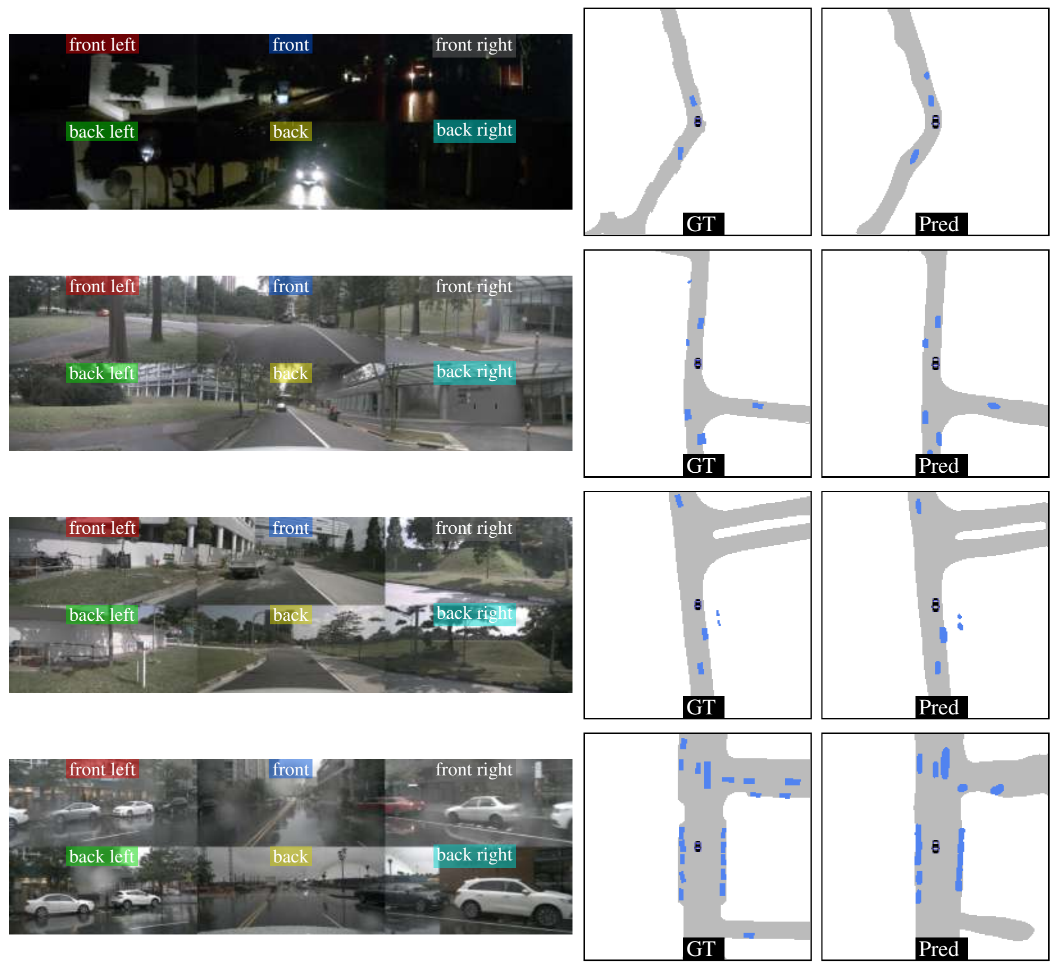

Figure 5: **Qualitative results on complex scenes.** We show the six camera views surrounding the vehicle along with segmentation ground truth for reference. Vehicles are shown in blue and driveable area in gray. Vehicles and driveable area predictions are from two different models trained independently for their respective ground-truth, the predictions are then merged for vizualization purpose. The ego vehicle is located in the center and facing downwards. Predictions of both driveable area and vehicle segmentation are thresholded at 0.5 for visualization purpose.

In this section, we also provide results for the driveable area segmentation task, also addressed by CVT [3]. Contrary to vehicle segmentation, this task requires the network to do "amodal completion" to a high degree, i.e., to correctly estimate regions of the road despite parts of it being severely occluded.

We followed the protocol of CVT [3] for this segmentation task; the ground truth is generated using HD-map's polygons from the dataset. We kept the same hyperparameters we used for the vehicle segmentation task, with a minor difference to the learning rate: we divide it by a factor 10 after 15 epochs (compared to a constant learning rate for vehicle segmentation).

Quantitative and qualitative results for this additional task are given respectively in Table 3 and Figure 5. When compared with CVT, we observe that LaRa achieves better performance (+0.9). Note that we do not do multi-tasking: following CVT [3], we train a model specifically for the task of driveable area segmentation. The qualitative examples in Figure 5 are produced by fusing predictions from two models.

Table 1: **Driveable area segmentation**. Results (in IoU) on nuScenes.

| Method | IoU |
|---|---|
| CVT | 74.3 |
| LaRa (ours) | **75.2** |

## D A quantitative study of the influence of ray embedding on attention consistency across cameras

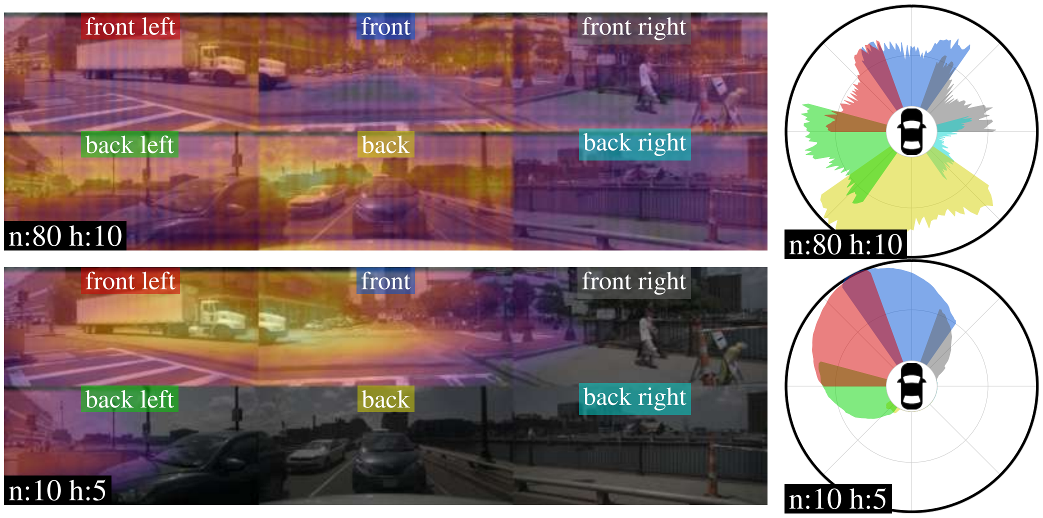

Figure 6: **Input-to-latent attention study.** Analysis of attention maps for two networks trained with different input embeddings. Top row is with 'Fourier + Cam. idx' and bottom row is with our proposed 'Cam. rays' embedding. The attention for one attention head and one latent is shown on the left superimposed with RGB images. The polar plots represent the directional attention intensity for one attention head with one latent vector. The radial distance is proportional to the attention level and shows the directions the network attends to the most.

In this section, we propose a quantitative analysis to support our claim that "our network is able to retrieve the pixel relationships between views thanks to our ray embedding" (Sec. 4.3 in the main paper).

To this end, we introduce a metric that directly quantifies the consistency and alignment of attention values across camera by analyzing behavior in "overlapping" regions, i.e., regions seen by two different cameras. We provide a visual description of this metric and its computation in Figure 7.

In short, knowing the orientation of each camera, we compute the Mean Squared Error (MSE) of the directional attention intensity between cameras on their overlapping regions. This score is averaged for all the overlapping regions, latents and attention heads, and examples in the validation set. A score of zero indicates a perfect match of the attention levels on overlapping regions (i.e., across

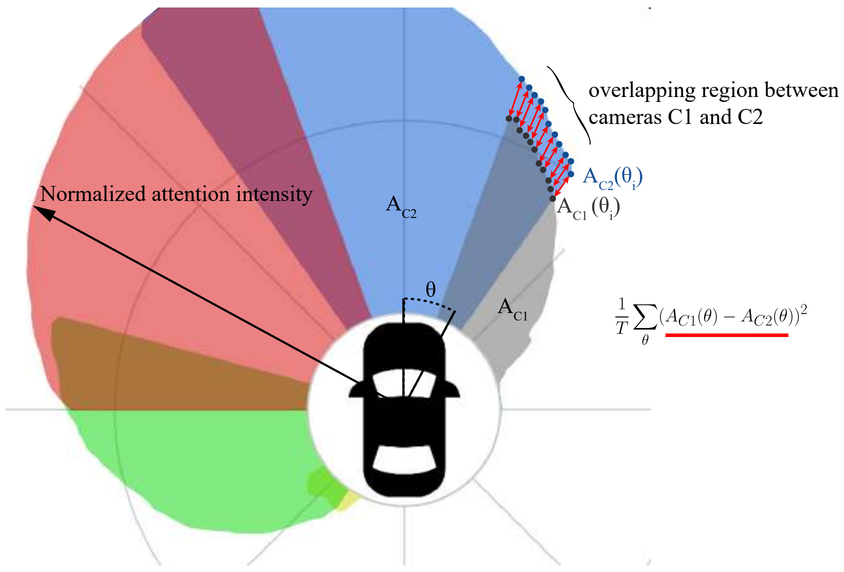
overlapping region between cameras C1 and C2
$A_{C2}(\theta_i)$
$A_{C1}(\theta_i)$
Normalized attention intensity
$A_{C2}$
$\theta$
$A_{C1}$
$\frac{1}{T}\sum_{\theta}(A_{C1}(\theta) - A_{C2}(\theta))^2$

Figure 7: **Measuring the attention consistency across cameras**. The proposed metric computes the Mean-Squared-Error (MSE) of the attention intensity on overlapping regions between cameras (as illustrated for two cameras and one latent and one attention head), and averages it over all cameras, latents, heads and scenes.

cameras). Results with this metric, reported in Table 2, show that our 'Cam. rays' embedding is 10 times more "consistent" across cameras than the baseline 'Fourier + Cam. idx'.

Table 2: **Impact of ray embedding on cross-camera attention consistency.** Cross-camera attention consistency (measured with proposed MSE metric, see Fig. 6) on nuScene.

| Embedding | MSE on overlap |
|---|---|
| 2D Fourier + Cam. idx | 0.0896 |
| Cam. rays (ours) | **0.0068** |

Additionally, we provide qualitative examples of the 'Fourier + Cam. idx' embedding to compare against our ray embedding in Figure 6. Contrary to the attention yield by our ray embedding, the one derived from the 'Fourier + Cam. idx' embedding is much more spread out and less consistent across cameras.

## E Comparison to PETR encoding

In PETR [14], the embedding of each pixel is computed by sampling its ray given $D$ predefined depths. The 3D coordinates of the $D$ sampled points along the ray are normalized, concatenated, processed by an MLP and summed with the visual features. Conceptually, the embedding is a way to indicate to the network "this pixel can observe these 3D points in the camera frustum space".

The embedding in PETR differs in that it is limited by the sampling resolution (i.e., the $D$ predefined depths), as computation and memory footprint increase linearly with respect to $D$. In contrast, we showed that our constant-complexity embedding is effective as a 3D positional embedding.

In addition, we include quantitative results to compare PETR embedding against our ray embedding in Table 3. We trained our model with PETR input embedding in place of ours. The results show that our ray embedding performs better (+72%).

Table 3: **Impact of ray embedding on performance.** Vehicle segmentation performance (in IoU) for vehicle segmentation on nuScenes.

| Embedding | IoU |
|---|---|
| PETR [14] | 34.8 |
| Cam. rays (ours) | **35.4** |

## F Additional attention qualitative analysis

We also provide additional analysis of attention maps for the multi-camera input shown in Figure 8 with a network using 256 latents and 32 attention heads. As in the main paper, the polar plots represent the directional attention intensity, showing the directions the network attends the most. The contribution of each camera is indicated by a color code coherent with Figure 8. Each polar plot is oriented in an upward direction (i.e., the front of the car points upward).

## G Additional qualitative examples

We also provide videos of our segmentation results on complex scenes in various visual conditions (daylight, rain, night). In these videos, we compare against our two baselines CVT [3] and Lift-Splat [1]. For a fair comparison, we use our model trained with visibility $> 40\%$ against CVT and $> 0\%$ against Lift-Splat.

## References

[1] J. Philion and S. Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3D. In *ECCV*, 2020.

[2] A. Hu, Z. Murez, N. Mohan, S. Dudas, J. Hawke, V. Badrinarayanan, R. Cipolla, and A. Kendall. FIERY: Future instance segmentation in bird's-eye view from surround monocular cameras. In *ICCV*, 2021.

[3] B. Zhou and P. Krähenbühl. Cross-view transformers for real-time map-view semantic segmentation. In *CVPR*, 2022.

[4] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[6] D. Hendrycks and K. Gimpel. Gaussian error linear units (GELUs). *arXiv 1606.08415*, 2016.

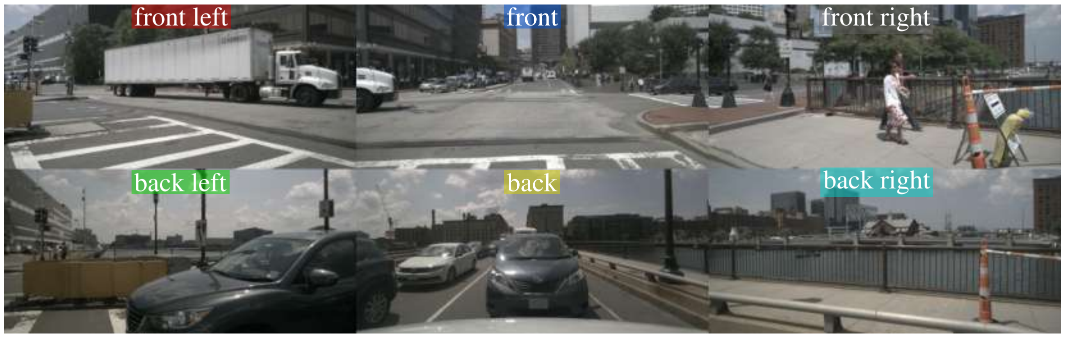

Figure 8: Six input camera images coming from the 360-degree camera rig of nuScenes. Note small overlaps between views, e.g., the front of the white truck is both seen in the front-left and front cams.

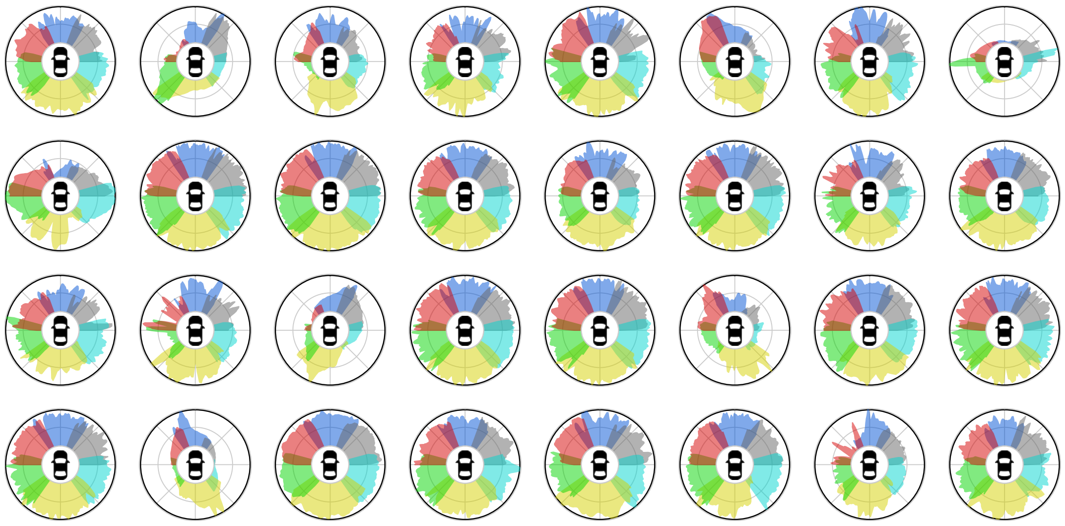

Figure 9: **Input-to-latent attention study — average over latents.** These polar plots represent the directional attention intensity averaged over all the 256 latent vectors for each attention head. When averaging over latent vectors, we observe that each attention head generally covers all directions. This suggests that the latent vectors contain most of the directional information and that the whole scene is attended across the latent. More rarely, an attention head's polar plot will be directional but will maintain a level of generality by being symmetrical.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[8] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, O. J. Henaff, M. Botvinick, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *ICLR*, 2022.

[9] W. Yifan, C. Doersch, R. Arandjelović, J. Carreira, and A. Zisserman. Input-level inductive biases for 3D reconstruction. In *CVPR*, 2022.

[10] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[11] T. Roddick and R. Cipolla. Predicting semantic map representations from images using pyramid occupancy networks. In *CVPR*, 2020.

[12] B. Pan, J. Sun, H. Y. T. Leung, A. Andonian, and B. Zhou. Cross-view semantic segmentation for sensing surroundings. In *IROS*, 2020.

[13] A. Saha, O. Mendez, C. Russell, and R. Bowden. Enabling spatio-temporal aggregation in birds-eye-view vehicle estimation. In *ICRA*, 2021.

[14] Y. Liu, T. Wang, X. Zhang, and J. Sun. Petr: Position embedding transformation for multi-view 3d object detection. *arXiv*, 2022.

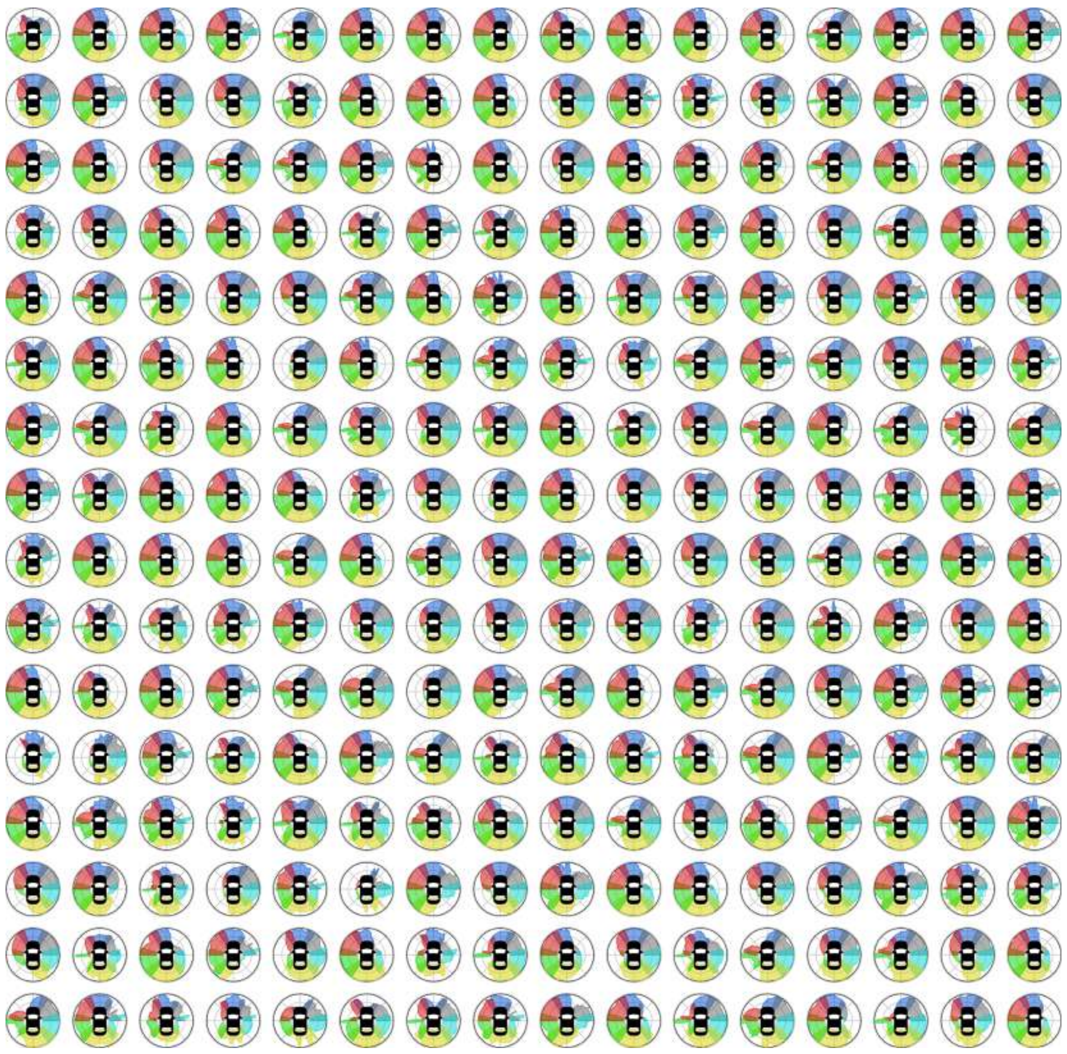

Figure 10: **Input-to-latent attention study — average over heads.** These polar plots represent the directional attention intensity averaged over all attention heads for the 32 attention heads. When averaging over attention heads, we observe that the average attention spans over half of the scene. This allows latent vectors to extract long-range context between views with the capacity to disambiguate them.

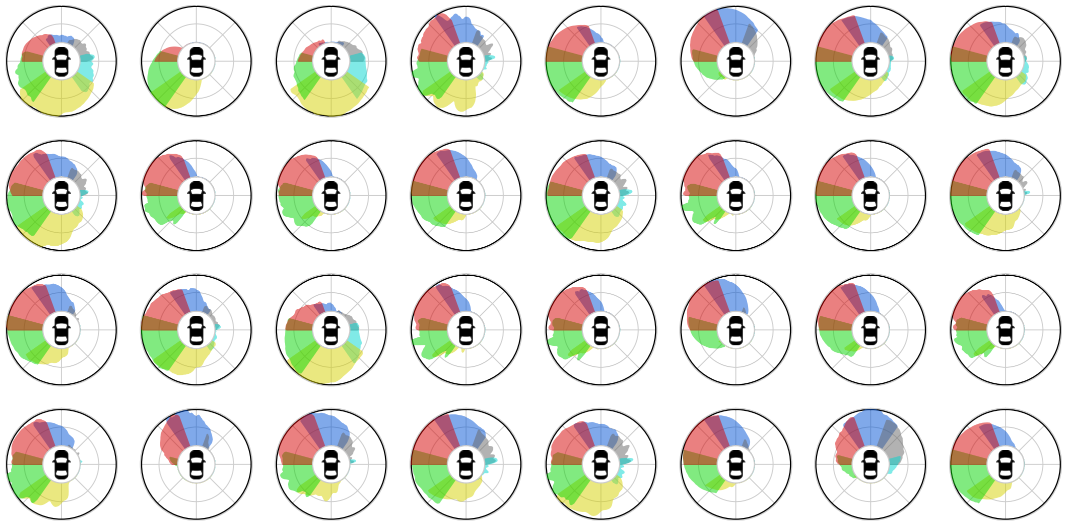
Figure 11: **Input-to-latent attention study — all the attention heads of a latent vector.** These polar plots represent the directional attention intensity of the 32 attention heads for a randomly chosen latent vector (latent vector #10). As shown in Figure 10, one latent vector approximately covers half of the scene over its attention heads.

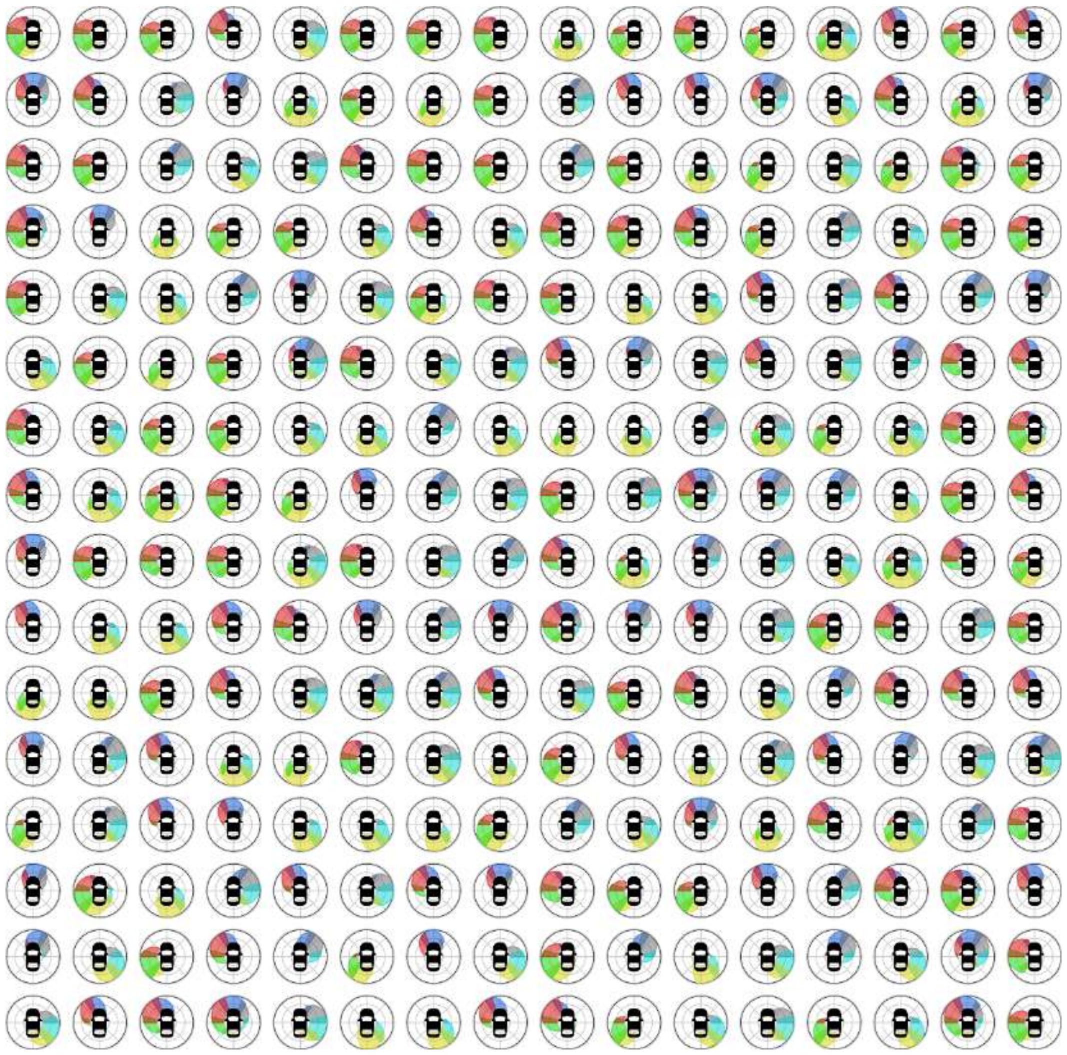

Figure 12: **Input-to-latent attention study — all the latent vectors for an attention head.** These polar plots represent the directional attention intensity of the 256 latent vectors for a randomly chosen attention head (head #4). As shown in Figure 9, one attention head generally covers the full scene over the latent vectors.