UMC: A Unified Bandwidth-efficient and Multi-resolution based Collaborative Perception Framework Supplementary Material

Tianhang Wang[1], Guang Chen[1,✉]*, Kai Chen[1], Zhengfa Liu[1], Bo Zhang[2], Alois Knoll[3], Changjun Jiang[1]
[1]Tongji University, [2]Shanghai Westwell Technology Co., Ltd, [3]Technische Universität München
{tianya_wang,guangchen,14sdck,1811466,cjjiang}@tongji.edu.cn
bob.zhang@westwell-lab.com knoll@in.tum.de

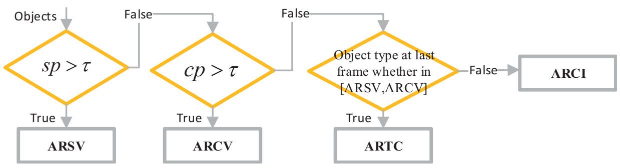

Figure 1. In the test set, we traverse all objects in each frame to obtain their corresponding types. Note that *sp*, *cp* are short for detected points from single and collaborative view, respectively.

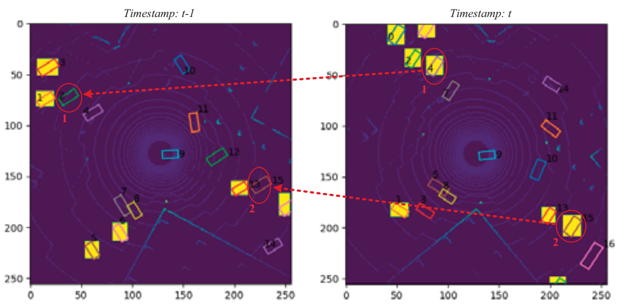

Figure 2. The detailed process of manually labelling.

1. Detailed information of Proposed Metrics

The detailed process of proposed metrics are shown in Figure 1. We traverse all objects in each frame to obtain their corresponding types. Note that the type of ARSV/ARCV can be automatically generated with code, as shown in Listing 1. And the type of ARCI/ARTC needs to be manually labeled, because whether it is visible at the last moment cannot be directly determined due to the movement of the vehicles from $t-1$ to t, as shown in Figure 2.

For detailed process of manually labelling, *e.g.*, at timestamp t, we firstly label vehicles with yellow background as ARCI type. Then, by comparing with time $t-1$, we find that agent 1,2 are visible at time $t-1$ (with no yellow background). Based on that, we change the type of agent 1,2 from ARCI to ARTC. The pseudo code are as follows:

```
single_view = np.load(...)
collab_view = np.load(...)
for object in len(single_view.objects):
    if singel_view[object].points > threhold:
        object.type = ARSV
    else:
        if collab_view[object].points >
   threhold:
            object.type = ARCV
        else:
            object.type = ARCI

% Finally, we will manually label partial ARCI
     to ARTC, as shown in Figure 2.
```

Listing 1. Pseudo code for proposed metrics

As for the threshold points τ, the proposed metrics distinguish between ARSV and ARCV based on whether they are visible. Nevertheless, whether they are visible depends on the number of points included in each object and the performance of the detector. To verify at how many points the detector can not detect the object, we conducted the following experiments on the *No Fusion* model.

Table 1. The ARSV and ARCV performance with different threshold points τ.

Points	$\text{ARSV}_{50/70}$	$\text{ARCV}_{50/70}$
10	81.48/77.48	23.72/19.34
7	79.44/75.05	15.34/11.88
5	77.73/73.21	6.08/4.07
4	76.90/72.35	4.62/3.45

Table 1 shows the ARSV and ARCV in terms of different points. We can see that i) as the number of points decreases, so does the ARCV. This is because, as

*Corresponding author: guangchen@tongji.edu.cn. Our code is available at:https://github.com/ispc-lab/UMC.

the points become more accurate, the no fusion model should theoretically be 0 in terms of ARCV; ii) when the points are greater than 5, the decline in ARCV is very large. When points are less than 5, the decline of the ARCV slows down, indicating it is close to the accurate points; iii) when points equal 4, the ARCV is 4.62% in IoU@0.5 and 3.45% in IoU@0.7, which are within the acceptable error range of 5%. So, to decide if an object is visible, we look at how many points it has and whether that number is greater than 4.

Note that the 'last frame' of ARTC's time interval varies based on the sampling frequency (5Hz in V2X-Sim, 10Hz in OPV2V). And, the ARSV and ARCV only take recall into account, while the AP is weighted by both recall and precision. Hence, a high AP does not necessarily mean high ARSV or ARCV values. Based on that, you may wonder why not utilize APSV/APCV as the new metric. It is because the each proposed model can only predict the classifications (background or foreground) and regressions (x,y,w,h) of each pixel. Therefore, there is no prediction of the corresponding types for each detected objects.

2. Experiments details

2.1. Basic parameters

Our experiments are all performed on the workstation with AMD Core Ryzen Threadripper 3960X CPU and Nvidia 3090 GPU with Pytorch v1.7.1, CUDA 11.0. The SRAR-based's transmitted collaborative feature map (TCF) has a dimension of $32 \times 32 \times 256$. Our proposed UMC's TCFs have the dimension of $32 \times 32 \times 256$, $64 \times 64 \times 128$. As for hyper-parameter tuning, we choose Adam as the optimizer and set the batch size to 4 for both V2X-Sim and OPV2V datasets. Also, we utilize the same number of scenes (total 80 scenes) for training. For testing stage, the V2X-sim utilizes 10 scenes, and OPV2V utilizes 15 scenes. Meanwhile, we use initial learning rate of 0.001 and set the random seed to 622.

2.2. Baseline setting

To ensure fairness, we fix the structure of the shared feature extractor MotionNet[9] and detector, and transplant the collaborative part of the different methods without modification. Meanwhile, all the models are trained 100 epoch with initial learning rate of 0.001 and set the learning rate update strategy as 'torch.optim.lr_scheduler.MultiStepLR(self. optimizer_head, milestones=[50, 100], gamma=0.5)'.

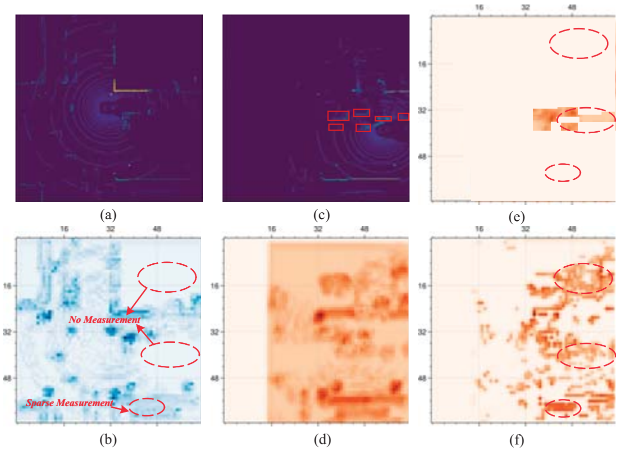

Figure 3. Difference between where2comm and UMC communication strategy. (a) The ego agent's observation. (b) The ego agent's observation at feature level. (c) The collaborator's observation, red box denoted for the detected agents. (d) The collaborator's observation at feature level. (e) Communication map of where2comm. (f) Communication map of UMC.

2.3. Communication Volume

The communication volume for each method is calculated by: $\text{mean}(\log(\sum_{i=1}^{agents}\sum_{t=1}^{T}(F.\text{size} + Q.\text{size})))$ during the test stage. F represents the transmitted feature map, and Q represents the query matrix, which is calculated in UMC, who2com, and when2com, and equals 0 in other methods.

2.4. Setting of δ_s, δ_c

Table 2. The performance-bandwidth trade off with different δ_s, δ_c values on V2X-Sim dataset.

δ_s	δ_c	$\text{ARSV}_{50/70}\uparrow$	$\text{ARCV}_{50/70}\uparrow$	$\text{ARCI}_{50/70}\downarrow$	$\text{ARTC}_{50/70}\uparrow$	$\text{AP}_{50/70}\uparrow$	C.V.$\downarrow$
50	100	85.66/81.87	70.76/63.15	2.41/1.7	11.88/8.12	68.97/61.35	20.58
50	50	84.78/80.73	67.46/60.72	2.50/1.78	11.88/8.12	67.83/60.02	19.92
50	10	80.62/75.39	31.48/23.53	2.17/1.37	12.03/7.20	59.57/50.50	18.4
20	100	87.30/84.08	74.08/66.77	16.68/12.24	22.15/17.41	66.98/59.24	19.68
20	50	83.23/78.98	56.49/48.87	2.12/1.51	11.23/8.08	64.77/56.79	19
20	10	80.53/75.38	19.02/12.33	2.14/1.38	9.16/6.74	58.00/49.46	17.63
10	100	86.57/82.97	59.37/51.68	14.57/10.51	19.09/15.15	63.59/55.84	19.06
10	50	82.12/77.44	38.52/31.28	2.16/1.47	10.77/8.35	61.57/53.43	18.376
10	10	80.78/75.51	13.29/8.97	2.21/1.39	10.77/8.35	57.62/49.27	17.171
5	100	86.04/82.59	47.19/40.44	14.17/9.9	19.25/15.14	61.04/53.99	18.375
5	20	80.87/75.72	15.09/10.76	2.12/1.36	10.77/8.35	57.67/49.45	17.17
1	100	85.44/81.49	34.59/28.72	13.47/9.88	20.49/17.35	58.07/50.72	17.15

We record the proposed UMC under different (δ_s, δ_c) in terms of the trade-off between performance and communication bandwidth, as shown in Table 2.

Meanwhile, we comprehensively analyze the communication strategy between where2comm[1] and our proposed UMC. As shown in Figure 3, where2comm takes the advantages of sparsity of foreground information and only transmits the regions that the agents have. However, as for different downstream tasks, such as segmentation[10] or scene completion[4], there are other no-measurement or sparse-measurement regions that need collaborator's communication, as shown in Figure 3.(b). Hence, our proposed UMC aim to optimize not only detection but also general downstream tasks based on the traditional information theory. From the Figure 3.(f), we can observe that UMC can transmit the necessary regions to ego agent for general downstream tasks.

Note that in addition to discussing the top-$\delta\%$ based filtering strategy of Eq.1 in manuscript, we also explored the mean based filtering strategy, the corresponding performance is shown as follows:

Table 3. The performance of mean based filtering strategy on V2X-Sim dataset.

δ_s	δ_c	$\text{ARSV}_{50/70}\uparrow$	$\text{ARCV}_{50/70}\uparrow$	$\text{ARCI}_{50/70}\downarrow$	$\text{ARTC}_{50/70}\uparrow$	$\text{AP}_{50/70}\uparrow$	C.V. $\downarrow$
mean		84.67/80.68	67.01/60.04	2.38/1.70	12.35/9.46	67.80/60.01	19.23

3. Performance analysis

3.1. Computation complexity

The configuration of the experiment platform has been described in Section 2.1. Based on that, in terms of computations, the proposed entropy-cs only requires about 0.136G FLOPS with 0.66 ms latency to process a $256\times\ 32\times32$ (C,H,W) feature map (more architecture details are shown in Section 4.1).

Compared to DiscoNet[3], the proposed C-GRU costs about 4.10G FLOPS more with 12.7 ms latency.

3.2. Is Early Fusion always be better?

We discuss the performance of early fusion model. As we all know, Early fusion aggregates the raw measurements from all collaborators, promoting a holistic perspective. From the Table 1 in manuscript, the early fusion performs extremely good, even better than all the other baselines in some metrics. However, V2VNet[8] actually shows early fusion is far from optimal due to noises in real sensors. To address the above issue, since the dataset of V2VNet is not open source, we conduct experiments on OPV2V[12] with Gaussian noises. As shown in Table 4, we agree that the performance of Early Fusion may be degraded by noises to some extent.

Table 4. Comparisons on OPV2V dataset.[**Best**, **Worst**]

Method	$\text{ARSV}_{50/70}$	$\text{ARCV}_{50/70}$	$\text{AP}_{50/70}$
No Fusion	69.33/**46.35**	**12.32**/**4.36**	**54.69**/**23.94**
Early Fusion	**64.62**/46.95	45.00/24.39	55.88/**25.89**
UMC	**76.56**/**47.68**	**47.82**/**25.06**	**61.90**/24.50

Table 5. Comparisons of single-grain selection.

Multi-Grains Selection $F^{e,t}_{i,1}$	$F^{e,t}_{i,2}$	$F^{e,t}_{i,3}$	$\text{AP}_{50/70}\uparrow$	C. V. $\downarrow$
✓			56.73/49.22	7.88
	✓		58.96/52.86	8.12
		✓	57.43/51.66	8.36

3.3. Grains selection

Table 3 in manuscript compares the performance of different selections of grain level. We also include comparisons of single-grain, as shown in Table 5. Note that the heavy memory burden of all resolution baseline is not applicable on our RTX 3090.

3.4. More details about ablations analysis

Table 4 in manuscript shows that a tremendous drop when adding Entropy-CS in variant 3 and 4. From our perspectives, variant 3 and 4 are based on single-resolution, then variant 3 (w/ entropy-cs)costs about $\frac{1}{4}$ communication of variant 4. Based on [1], when the communication is too small, the collaborative detection performance will suffer, resulting in a tremendous drop in variant 3. However, variant 3 still achieves detection gain compared with No Fusion (improved by $9.40\%/9.25\%\uparrow$ in $\text{AP}_{50/70}$, respectively).

Meanwhile, AP of variant 2 is worse than variant 4 with comparable ARSV and better ARCV, this is because The ARSV and ARCV only take recall into account, while the AP is weighted by both recall and precision. Therefore, in variants 2 and 4, a high AP does not necessarily mean high ARSV or ARCV values, more details about proposed metrics can be found in Section 1.

3.5. Unified framework design

We summarize the main contributions of recent collaborative algorithms in Table 6, , where ✓ indicates that a unique module is designed and – indicates that general operations are utilized. Our proposed UMC optimizes the communication, collaboration, and reconstruction process with multi-resolution technique.

Table 6. Contribution summary.

Method	Comm.	Collab.	Recons.
Who2com (ICRA 2020 [6])	✓	-	-
When2com (CVPR 2020 [5])	✓	-	-
V2VNet (ECCV 2020 [8])	-	✓	-
DiscoNet (NIPS 2021 [3])	-	✓	-
V2X-ViT (ECCV 2022 [11])	-	✓	-
Where2comm (NIPS 2022 [1])	✓	✓	-
UMC (ours)	✓	✓	✓

4. Detailed architecture of the model

Note that we will release the source code.

4.1. Architecture of entropy-CS

```python
def acc_entropy_selection(self, tg_agent, nb_agent, delta1, delta2, M=3, N=3):
    self.stack = stack_channel(1, 9, kernel_size=3, padding=1)
    w = nb_agent.shape[-2]
    h = nb_agent.shape[-1]
    batch_nb = nb_agent.reshape(-1, 1, 1, 1)
    stack = self.stack(nb_agent).permute(2,3,1,0).contiguous().reshape(-1, 9, 1, 1)

    p = F.sigmoid((stack - batch_nb)).mean(dim=1).reshape(w, h)
    entropy_tmp = p * torch.log(p)

    with torch.no_grad():
        top_delta = torch.sort(entropy_tmp.reshape(-1), descending=True)
        self_holder = top_delta[0][int(w*h*delta1)]

    masker = torch.where(entropy_tmp>=self_holder)

    stack_tg = self.stack(tg_agent).permute(2,3,1,0).contiguous().reshape(-1, 9, 1, 1)
    p_t = F.sigmoid((stack_tg - batch_nb)).mean(dim=1).reshape(w, h)
    entropy_t = p_t * torch.log(p_t)

    tmp_masker = - torch.ones_like(entropy_t)
    tmp_masker[masker] = entropy_t[masker]

    with torch.no_grad():
        top_delta2 = torch.sort(tmp_masker[tmp_masker!=-1].reshape(-1), descending=True)
        thresholds = top_delta2[0][int(w*h*delta2)]

    return torch.where(tmp_masker>=thresholds)


class stack_channel(nn.Conv2d):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1, padding=0, bias=False, interplate='none'):
        super(stack_channel, self).__init__(in_channels, out_channels, kernel_size=kernel_size, stride=stride, padding=padding, bias=bias)

        square_dis = np.zeros((out_channels, kernel_size, kernel_size))

        for i in range(out_channels):
            square_dis[i, i//3, i%3] = 1

        self.square_dis = nn.Parameter(torch.Tensor(square_dis), requires_grad=False)

    def forward(self, x):

        kernel = self.square_dis.detach().unsqueeze(1)
        stack = F.conv2d(x, kernel, stride=1, padding=1, groups=1)

        return stack
```

Listing 2. Entropy-CS code

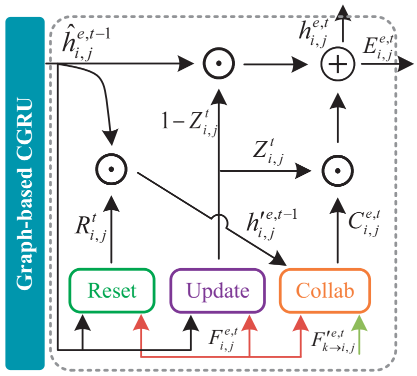

Figure 4. The architecture of G-CGRU.

Our contribution of entropy-based selection is both theoretical and practical. The intuition of entropy-cs is low computational complexity and high interpretability. The entropy-cs is no parameter and single-round communication to reduce heavy bandwidth burden brought by multi-resolution technique.

4.2. Architecture of G-CGRU

To facilitate understanding, we have simplified many formulas and steps in manuscript. Hence, we add more technique details about the section of `Graph-based Collaborative GRU`.

For the ego agent i of the j-th resolution intermediate feature maps, the inputs of G-CGRU are hidden states $\boldsymbol{h}_{i,j}^{e,t-1}$, the ego agent observation $\boldsymbol{F}_{i,j}^{e,t}$, and the supporters' selected feature maps $\{\boldsymbol{F}_{k\to i,j}^{'e,t}\}_{k\neq i}$, then the updates for G-CGRU at t-th step can be formulated as:

$$
\begin{aligned}
\hat{\boldsymbol{h}}_{i,j}^{e,t-1} &= \Lambda(\boldsymbol{h}_{i,j}^{e,t-1}, \boldsymbol{\xi}_i^{t-1\to t}) \\
\boldsymbol{R}_{i,j}^{t} &= Reset(\boldsymbol{F}_{i,j}^{e,t}, \hat{\boldsymbol{h}}_{i,j}^{e,t-1}) \\
\boldsymbol{Z}_{i,j}^{t} &= Update(\hat{\boldsymbol{h}}_{i,j}^{e,t-1}, \boldsymbol{F}_{i,j}^{e,t}) \\
\boldsymbol{h}_{i,j}^{'e,t-1} &= \hat{\boldsymbol{h}}_{i,j}^{e,t-1} \odot \boldsymbol{R}_{i,j}^{t} \\
\boldsymbol{C}_{i,j}^{e,t} &= Collab(\boldsymbol{h}_{i,j}^{'e,t-1}, \{\boldsymbol{F}_{k\to i,j}^{'e,t}\}_{k\neq i}, \boldsymbol{F}_{i,j}^{e,t}) \\
\boldsymbol{E}_{i,j}^{e,t} &= \boldsymbol{Z}_{i,j}^{t} \odot \boldsymbol{C}_{i,j}^{e,t} + (1-\boldsymbol{Z}_{i,j}^{t}) \odot \hat{\boldsymbol{h}}_{i,j}^{e,t-1} \\
\boldsymbol{h}_{i,j}^{e,t} &= \boldsymbol{W}_{3\times 3} * \boldsymbol{E}_{i,j}^{e,t}
\end{aligned} \tag{1}
$$

where $\odot$, $*$ represent dot product and 3×3 convolution operation, respectively. $\boldsymbol{W}_{3\times3}$ indicates trainable parameters. The last time hidden feature $\boldsymbol{h}_{i,j}^{e,t-1}$ needs conduct feature alignment operation from $t-1$ to t to get $\hat{\boldsymbol{h}}_{i,j}^{e,t-1}$, which ensure the $\hat{\boldsymbol{h}}_{i,j}^{e,t-1}$ and $\boldsymbol{F}_{i,j}^{e,t}$ are supported in the same coordinate system.

The *Reset* and *Update* gate modules share the same structure. Here we take *Reset* as an example:

$$\begin{aligned}\boldsymbol{W}_{ir} &= \sigma(\boldsymbol{W}_{3\times3} * ([\hat{\boldsymbol{h}}_{i,j}^{e,t-1}; \boldsymbol{F}_{i,j}^{e,t}])) \\ \boldsymbol{R}_{i,j}^{t} &= \sigma(\boldsymbol{W}_{ir} \odot \hat{\boldsymbol{h}}_{i,j}^{e,t-1} + (1-\boldsymbol{W}_{ir}) \odot \boldsymbol{F}_{i,j}^{e,t})\end{aligned} \tag{2}$$

where $\sigma(\cdot)$, $[\cdot;\cdot]$ represent Sigmoid function and concatenation operation along channel dimensions. The gate $\boldsymbol{R}_{i,j}^{t} \in \mathbb{R}^{K,K,C}$ learns where the hidden features $\boldsymbol{h}_{i,j}^{e,t-1}$ are conducive to the present.

Based on the above *Reset* and *Update* modules, we thus derive the *Collab* module. To make better collaborative feature integration, we construct a collaboration graph $\mathcal{G}_c^t(\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}})$ in *Collab* module, where node $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_i\}_{i=1,\ldots,N}$ is the set of collaborative agents in environment and $\boldsymbol{\mathcal{E}} = \{\boldsymbol{W}_{k\to i}\}_{i,k=1,\ldots,N}$ is the set of trainable edge matrix weights between agents and models the collaboration strength between two agents. Let $\mathcal{C}_{\mathcal{G}_c^t}(\cdot)$ be the collaboration process defined in the *Collab* module's graph $\mathcal{G}_c^t$. The j-th resolution enhanced maps of ego i agent after collaboration are $\boldsymbol{E}_{i,j}^{e,t} \leftarrow \mathcal{C}_{\mathcal{G}_c^t}(\boldsymbol{h}_{i,j}^{e,t-1}, \boldsymbol{F}_{k\to i,j}^{\prime e,t}, \boldsymbol{F}_{i,j}^{e,t})$. This process has two stages: message attention (**S1**) and message aggregation (**S2**).

$$\begin{aligned}\boldsymbol{W}_{k\to i} &= \Pi([\boldsymbol{h}_{i,j}^{\prime e,t-1}; \boldsymbol{F}_{k\to i,j}^{\prime e,t}; \boldsymbol{F}_{i,j}^{e,t}] \in \mathbb{R}^{K,K} \\ \bar{\boldsymbol{W}}_{k\to i} &= \frac{e^{\boldsymbol{W}_{k\to i}}}{\sum_{m=1}^{N} e^{\boldsymbol{W}_{m\to i}^{t}}} \\ \boldsymbol{C}_{i,j}^{e,t} &= \sum_{m=1}^{N} \bar{\boldsymbol{W}}_{m\to i} \circ \boldsymbol{F}_{m\to i,j}^{\prime e,t}\end{aligned} \tag{3}$$

In the *message attention stage* (**S1**), each agent determines the matrix-valued edge weights, which reflect the strength from one agent to another at each individual cell. To determine the edge weights, we firstly get the conductive history information from hidden features by *Reset* gates through $\boldsymbol{h}_{i,j}^{\prime e,t-1} \leftarrow \hat{\boldsymbol{h}}_{i,j}^{e,t-1} \odot \boldsymbol{R}_{i,j}^{t}$. Then, we utilize the edge encode Π to correlate the history information, the feature map from another agent and ego feature map; that is, the matrix-value edge weight from k-th agent to the i-th agent is $\boldsymbol{W}_{k\to i} = \Pi(\boldsymbol{h}_{i,j}^{\prime e,t-1}, \boldsymbol{F}_{k\to i,j}^{\prime e,t}, \boldsymbol{F}_{i,j}^{e,t}) \in \mathbb{R}^{K,K}$, where Π concatenates three feature maps along the channel dimension and then utilizes four 1×1 convolutional layers to gradually reduce the number of channels from $3C$ to 1, more details are shown in Section 4.6. Also, to normalize the edge weights across different agents, we implement a softmax operation on each cell of the feature map.

In the *message aggregation stage* (**S2**), each agent aggregates the feature maps from collaborators based on the normalized matrix-valued edge weights, the updated feature map $\boldsymbol{C}_{i,j}^{e,t}$ is utilized by $\sum_{k=1}^{N} \boldsymbol{W}_{k\to i} \circ \boldsymbol{F}_{k\to i,j}^{\prime e,t}$, where $\circ$ represents the dot production broadcasting along the channel dimension.

Finally, the collaborative map is $\boldsymbol{E}_{i,j}^{e,t} = \boldsymbol{Z}_{i,j}^{t} \odot \boldsymbol{C}_{i,j}^{e,t} + (1-\boldsymbol{Z}_{i,j}^{t}) \odot \boldsymbol{h}_{i,j}^{e,t-1}$. Note that the $\boldsymbol{Z}_{i,j}^{t}$ is generated by *Update* gate and $\odot$ is the dot product. And, the hidden state is updated as $\boldsymbol{h}_{i,j}^{e,t} \leftarrow \boldsymbol{W}_{3\times3} * \boldsymbol{E}_{i,j}^{e,t}$.

4.3. Architecture of shared encoder

We use the main architecture of MotionNet[9] as our shared encoder. The input BEV map's dimension is $(c, w, h) = (13, 256, 256)$. We describe the architecture of the encoder below:

```
nn.Sequential(
      nn.Conv2d(13, 32, 3, stride=1, padding=1)
      nn.BatchNorm2d(32)
      nn.ReLU()
      nn.Conv2d(32, 32, 3, stride=1, padding=1)
      nn.BatchNorm2d(32)
      nn.ReLU()
      nn.Conv3D(64, 64, (1,1,1), stride=1)
      nn.Conv3D(128, 128, (1,1,1), stride=1)
      nn.Conv2d(32, 64, 3, stride=2, padding=1)
      nn.BatchNorm(64)
      nn.ReLU()
      nn.Conv2d(64, 128, 3, stride=2, padding=1)
      nn.BatchNorm(128)
      nn.ReLU()
      nn.Conv2d(128, 128, 3, stride=1, padding
      =1)
      nn.BatchNorm(128)
      nn.ReLU()
      nn.Conv2d(128, 256, 3, stride=2, padding
      =1)
      nn.BatchNorm(256)
      nn.ReLU()
      nn.Conv2d(256, 256, 3, stride=1, padding
      =1)
      nn.BatchNorm(256)
      nn.ReLU()
      nn.Conv2d(256, 512, 3, stride=2, padding
      =1)
      nn.BatchNorm(512)
      nn.ReLU()
      nn.Conv2d(512, 512, 3, stride=1, padding
      =1)
      nn.BatchNorm(512)
      nn.ReLU())
```

Listing 3. Shared encoder code

4.4. Architecture of SRAR-based shared decoder

The input of the SRAR-based shared decoder is the intermediate feature output by each layer of the encoder. Its architecture is shown below:

```
nn.Sequential(
    nn.Conv2d(512 + 256, 256, 3, 1, 1)
    nn.BatchNorm2d(256)
    nn.ReLU()
    nn.Conv2d(256, 256, 3, 1, 1)
    nn.BatchNorm2d(256)
    nn.ReLU()
    nn.Conv2d(256 + 128, 128, 3, 1, 1)
    nn.BatchNorm2d(128)
    nn.ReLU()
    nn.Conv2d(128, 128, 3, 1, 1)
    nn.BatchNorm2d(128)
    nn.ReLU()
    nn.Conv2d(128 + 64, 64, 3, 1, 1)
    nn.BatchNorm2d(64)
    nn.ReLU()
    nn.Conv2d(64, 64, 3, 1, 1)
    nn.BatchNorm2d(64)
    nn.ReLU()
    nn.Conv2d(64 + 32, 32, 3, 1, 1)
    nn.BatchNorm2d(32)
    nn.ReLU()
    nn.Conv2d(32, 32, 3, 1, 1)
    nn.BatchNorm2d(32)
    nn.ReLU())
```

4.5. Architecture of query generator

The entropy-CS compresses the intermediate feature to generate query matrix by query generator, which is for light communication. Its architecture is shown below:

```
nn.Sequential(
    nn.Conv2d(256, 64, 1, 1, 0)
    nn.BatchNorm2d(64)
    nn.ReLU()
    nn.Conv2d(64, 1, 1, 1, 0)
    nn.ReLU())
```

Listing 4. Query generator code

4.6. Architecture of edge encoder II

```
class EdgeEncoder(nn.Module):
    def __init__(self,channel):
        super(EdgeEncoder, self).__init__()

        self.conv1_1 = nn.Conv2d(channel, 128, kernel_size=1, stride=1, padding=0)
        self.bn1_1 = nn.BatchNorm2d(128)

        self.conv1_2 = nn.Conv2d(128, 32, kernel_size=1, stride=1, padding=0)
        self.bn1_2 = nn.BatchNorm2d(32)

        self.conv1_3 = nn.Conv2d(32, 8, kernel_size=1, stride=1, padding=0)
        self.bn1_3 = nn.BatchNorm2d(8)

        self.conv1_4 = nn.Conv2d(8, 1, kernel_size=1, stride=1, padding=0)

    def forward(self, x):
        x = x.view(-1, x.size(-3), x.size(-2), x.size(-1))
        x_1 = F.relu(self.bn1_1(self.conv1_1(x)))
        x_1 = F.relu(self.bn1_2(self.conv1_2(x_1)))
        x_1 = F.relu(self.bn1_3(self.conv1_3(x_1)))
        x_1 = F.relu(self.conv1_4(x_1))

        return x_1
```

Listing 5. Edge encoder code

4.7. Architecture of MGFE

```
class MGFE(nn.Module):
    def __init__(self, input_channel):
        super(MGFE, self).__init__()

        self.guide_v1 = nn.Conv2d(128, 128, kernel_size=1, stride=1, padding=0)
        self.guide_v1_bn = nn.BatchNorm2d(128)
        self.guide = nn.Conv2d(256, 256, kernel_size=1, stride=1, padding=0)
        self.guide_bn = nn.BatchNorm2d(256)

        self.conv5_1 = nn.Conv2d(512 + 256 + 256, 256, kernel_size=3, stride=1, padding=1)
        self.bn5_1 = nn.BatchNorm2d(256)

        self.conv5_2 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
        self.bn5_2 = nn.BatchNorm2d(256)

        self.conv6_1 = nn.Conv2d(256 + 128 + 128, 128, kernel_size=3, stride=1, padding=1)
        self.bn6_1 = nn.BatchNorm2d(128)

        self.conv6_2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1)
        self.bn6_2 = nn.BatchNorm2d(128)

        self.conv7_1 = nn.Conv2d(128 + 64, 64, kernel_size=3, stride=1, padding=1)
        self.conv7_2 = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)

        self.conv8_1 = nn.Conv2d(64 + 32, 32, kernel_size=3, stride=1, padding=1)
        self.conv8_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)

        self.bn7_1 = nn.BatchNorm2d(64)
        self.bn7_2 = nn.BatchNorm2d(64)

        self.bn8_1 = nn.BatchNorm2d(32)
        self.bn8_2 = nn.BatchNorm2d(32)

        self.norm1 = L2Norm(512)
        self.norm2 = L2Norm(256)
```

```
        self.norm3 = L2Norm(128)
        self.norm4 = L2Norm(64)
        self.norm5 = L2Norm(32)

    def forward(self, x, x_1, x_2, x_3, x_4, enhance_v1, enhance, batch, kd_flag = 0):

        enhance_v1 = enhance_v1.view(batch, -1, enhance_v1.size(1), enhance_v1.size(2), enhance_v1.size(3))
        enhance_v1 = enhance_v1.permute(0, 2, 1, 3, 4).contiguous()
        enhance_v1 = enhance_v1.permute(0, 2, 1, 3, 4).contiguous()
        enhance_v1 = enhance_v1.view(-1, enhance_v1.size(2), enhance_v1.size(3), enhance_v1.size(4)).contiguous()

        guide_v1 = torch.max(F.relu(self.guide_v1_bn(self.guide_v1(enhance_v1))), dim=1, keepdim=True)[0]
        guide = torch.max(F.relu(self.guide_bn(self.guide(enhance))), dim=1, keepdim=True)[0]
        x_3_guide = guide * x_3

        x_5 = F.relu(self.bn5_1(self.conv5_1(torch.cat((self.norm1(F.interpolate(x_4, scale_factor=(2, 2))), self.norm2(x_3_guide), self.norm2(enhance)), dim=1))))
        x_5 = F.relu(self.bn5_2(self.conv5_2(x_5)))

        x_2 = x_2.view(batch, -1, x_2.size(1), x_2.size(2), x_2.size(3))
        x_2 = x_2.permute(0, 2, 1, 3, 4).contiguous()
        x_2 = x_2.permute(0, 2, 1, 3, 4).contiguous()
        x_2 = x_2.view(-1, x_2.size(2), x_2.size(3), x_2.size(4)).contiguous()

        x_2_guide = guide_v1 * x_2

        x_6 = F.relu(self.bn6_1(self.conv6_1(torch.cat((self.norm2(F.interpolate(x_5, scale_factor=(2, 2))), self.norm3(x_2_guide), self.norm3(enhance_v1)), dim=1))))

        x_6 = F.relu(self.bn6_2(self.conv6_2(x_6)))

        x_1 = x_1.view(batch, -1, x_1.size(1), x_1.size(2), x_1.size(3))
        x_1 = x_1.permute(0, 2, 1, 3, 4).contiguous()
        x_1 = x_1.permute(0, 2, 1, 3, 4).contiguous()
        x_1 = x_1.view(-1, x_1.size(2), x_1.size(3), x_1.size(4)).contiguous()

        x_7 = F.relu(self.bn7_1(self.conv7_1(torch.cat((self.norm3(F.interpolate(x_6, scale_factor=(2, 2))), self.norm4(x_1)), dim=1))))
        x_7 = F.relu(self.bn7_2(self.conv7_2(x_7)))


        x = x.view(batch, -1, x.size(1), x.size(2), x.size(3))
        x = x.permute(0, 2, 1, 3, 4).contiguous()
        x = x.permute(0, 2, 1, 3, 4).contiguous()
        x = x.view(-1, x.size(2), x.size(3), x.size(4)).contiguous()

        x_8 = F.relu(self.bn8_1(self.conv8_1(torch.cat((self.norm4(F.interpolate(x_7, scale_factor=(2, 2))), self.norm5(x)), dim=1))))
        res_x = F.relu(self.bn8_2(self.conv8_2(x_8)))

        return res_x

class L2Norm(nn.Module):
    def __init__(self, n_channels, scale=10.0):
        super(L2Norm, self).__init__()
        self.n_channels = n_channels
        self.scale = scale
        self.eps = 1e-10
        self.weight = nn.Parameter(torch.Tensor(self.n_channels))
        self.weight.data *= 0.0
        self.weight.data += self.scale

    def forward(self, x):
        norm = x.pow(2).sum(dim=1, keepdim=True).sqrt() + self.eps
        x = x / norm * self.weight.view(1, -1, 1, 1)

        return x
```

Listing 6. MGFE code

5. Detailed information of Interpolation

We utilize the main architecture of ADP-C[7] as our interpolate function in entropy-CS module. We assume the input feature as $\boldsymbol{f}_{in} \in \mathbb{R}^{K,K,C}$ and suppose the pixels of the $\boldsymbol{f}_{in}$ are indexed by $\boldsymbol{p}$. Then, we form a mask $\boldsymbol{M} \in \mathbb{R}^{K,K}$:

$$\boldsymbol{M}(\boldsymbol{p}) = \begin{cases} 0, & \text{if } \boldsymbol{f}_{in}(\boldsymbol{p}) = \boldsymbol{0} \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

Assuming C is a convolution layer with input $\boldsymbol{f}_{in}$, the by applying the mask, the output $\boldsymbol{f}_{out}$ at position $\boldsymbol{p}$ becomes:

$$\boldsymbol{f}_{out}(\boldsymbol{p}) = \begin{cases} C(\boldsymbol{f}_{in})(\boldsymbol{p}), & \text{if } \boldsymbol{M}(\boldsymbol{p}) = 1, \\ \boldsymbol{0}, & \text{if } \boldsymbol{M}(\boldsymbol{p}) = 0. \end{cases} \tag{5}$$

Denoting the interpolation operation as $\boldsymbol{I}$, the final output feature $\boldsymbol{f}^{*}_{out}$ is:

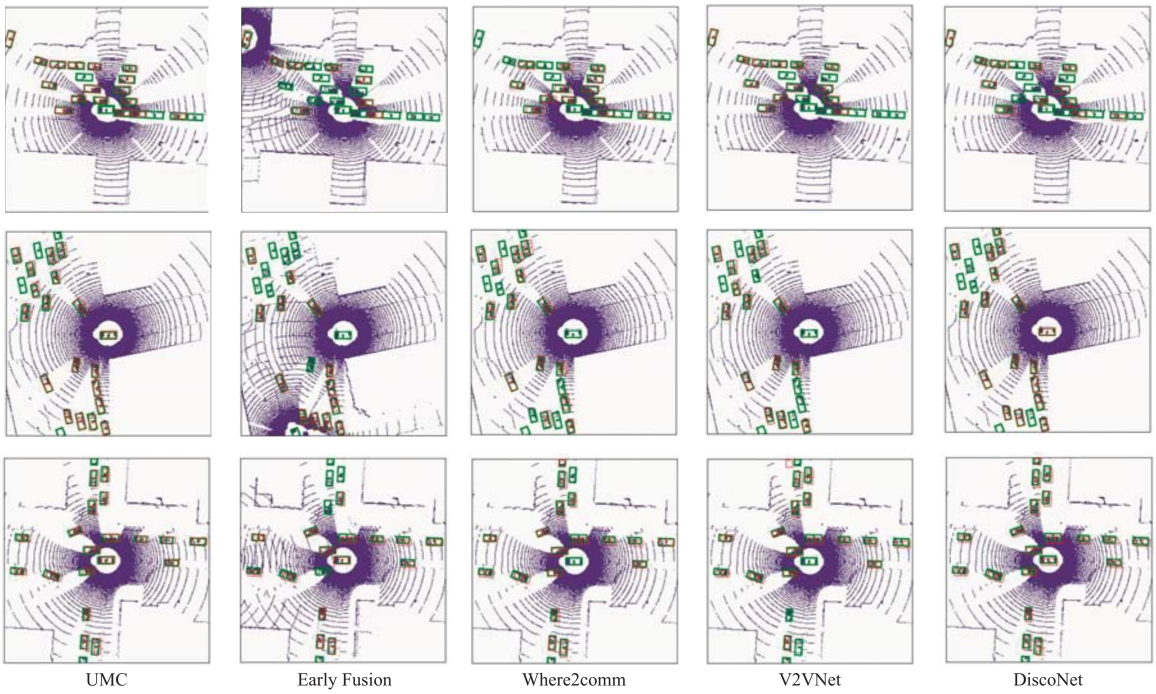

Figure 5. Detection results of UMC, Early Fusion, Where2comm, V2VNet and DiscoNet on OPV2V dataset.

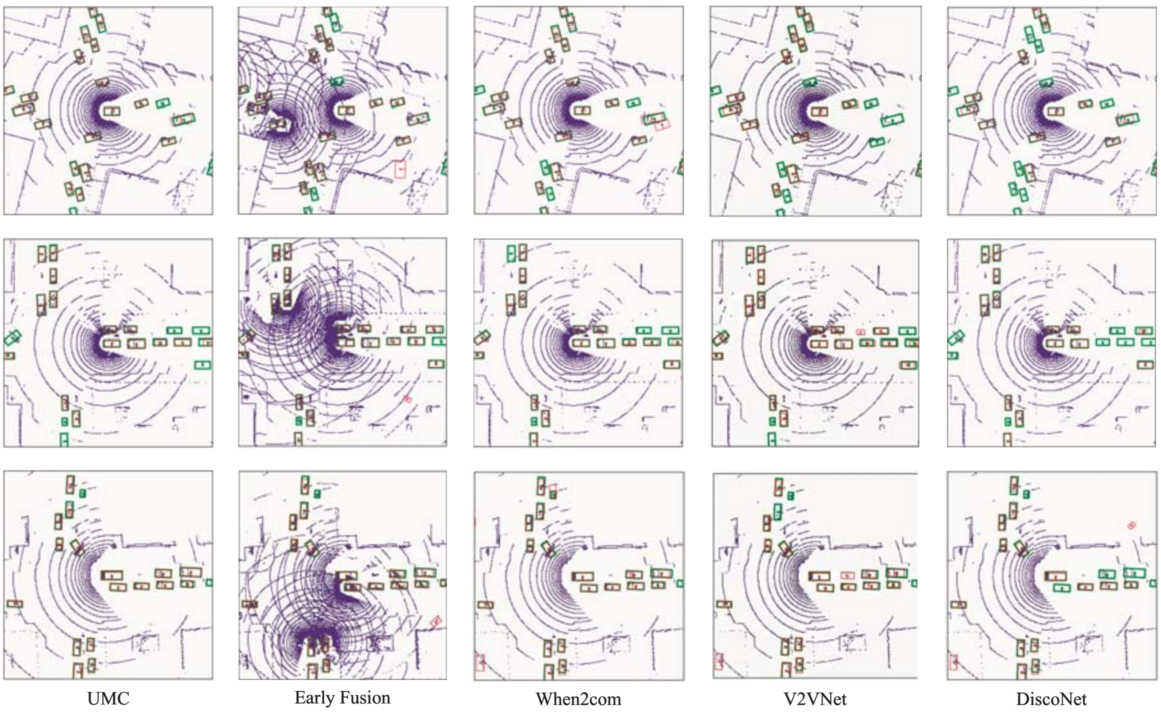

Figure 6. Detection results of UMC, Early Fusion, When2com[5], V2VNet and DiscoNet on V2X-Sim dataset.

$$\boldsymbol{f}^{*}_{out}(p) = \begin{cases} \boldsymbol{f}_{out}(\boldsymbol{p}), & \text{if } \boldsymbol{M}(\boldsymbol{p}) = 1, \\ \boldsymbol{I}(\boldsymbol{f}_{out})(\boldsymbol{p}), & \text{if } \boldsymbol{M}(\boldsymbol{p}) = 0. \end{cases} \tag{6}$$

The value of $I(\boldsymbol{f}_{out})(\boldsymbol{p})$ is weighted average of all the neighboring pixels centered at $\boldsymbol{p}$ within a radius r:

$$\boldsymbol{I}(\boldsymbol{f}_{out})(\boldsymbol{p}) = \frac{\sum_{\boldsymbol{s}\in\Psi(\boldsymbol{p})} \boldsymbol{W}_{(\boldsymbol{p},\boldsymbol{s})} \boldsymbol{f}_{out}(\boldsymbol{s})}{\sum_{\boldsymbol{s}\in\Psi(\boldsymbol{p})} \boldsymbol{W}_{(\boldsymbol{p},\boldsymbol{s})}} \tag{7}$$

where s indicates p's neighboring pixels and $\Psi(\boldsymbol{p}) = \{\boldsymbol{s} \| \|\boldsymbol{s} - \boldsymbol{p}\|_\infty \leqslant r, \boldsymbol{s} \neq \boldsymbol{p}\}$, the neighborhood of $\boldsymbol{p}$. In UMC, we set radius $r = 7$. $W_{(\boldsymbol{p},\boldsymbol{s})}$ is the weight assigned to point $\boldsymbol{s}$ for interpolating at $\boldsymbol{p}$, for which we utilize the RBF kernel, a distance-based exponential decaying weighting scheme:

$$\boldsymbol{W}_{(\boldsymbol{p},\boldsymbol{s})} = \exp(-\lambda^2 \|\boldsymbol{p} - \boldsymbol{s}\|_2^2) \tag{8}$$

with λ being a trainable parameter. This indicates that the closer $\boldsymbol{s}$ is to $\boldsymbol{p}$, the larger its assigned weight will be. Note that masked-out features $\boldsymbol{M}(\boldsymbol{p}) = 0$ still participate in the interpolation process as inputs with values of $\mathbf{0}$.

6. Detailed Qualitative results

We visualize the detection results between different collaborative approaches on V2X-sim[2] and OPV2V[12] datasets, as shown in Figure 5 and 6.

7. Loss curve

We visualize the loss curve of UMC, Early Fusion, When2com, Where2comm, V2VNet and DiscoNet on V2X-Sim and OPV2V on Figure 7 and 8, respectively.

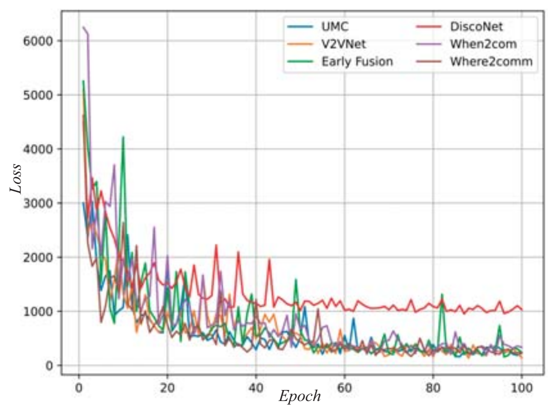

Figure 7. Epoch vs. loss on V2X-Sim dataset.

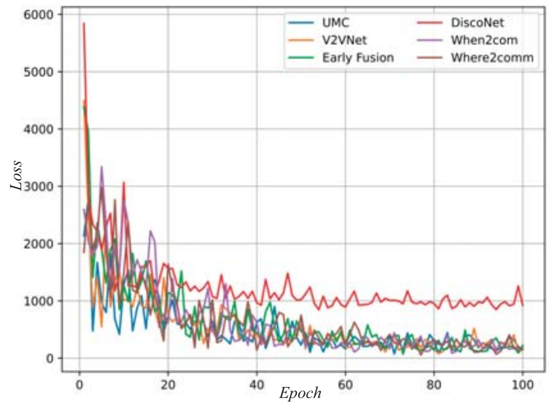

Figure 8. Epoch vs. loss on OPV2V dataset.

References

[1] Yue Hu, Shaoheng Fang, Zixing Lei, Yiqi Zhong, and Siheng Chen. Where2comm: Communication-efficient collaborative perception via spatial confidence maps. In *Thirty-sixth Conference on Neural Information Processing Systems (Neurips)*, November 2022. 2, 3

[2] Yiming Li, Dekun Ma, Ziyan An, Zixun Wang, Yiqi Zhong, Siheng Chen, and Chen Feng. V2x-sim: Multi-agent collaborative perception dataset and benchmark for autonomous driving. *IEEE Robotics and Automation Letters*, 7:10914–10921, 2022. 9

[3] Yiming Li, Shunli Ren, Pengxiang Wu, Siheng Chen, Chen Feng, and Wenjun Zhang. Learning distilled collaboration graph for multi-agent perception. In *NeurIPS*, 2021. 3

[4] Yiming Li, Juexiao Zhang, Dekun Ma, Yue Wang, and Chen Feng. Multi-robot scene completion: Towards task-agnostic collaborative perception. In *6th Annual Conference on Robot Learning*, 2022. 3

[5] Yen-Cheng Liu, Junjiao Tian, Nathan Glaser, and Zsolt Kira. When2com: Multi-agent perception via communication graph grouping. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4105–4114, 2020. 3, 8

[6] Yen-Cheng Liu, Junjiao Tian, Chih-Yao Ma, Nathan Glaser, Chia-Wen Kuo, and Zsolt Kira. Who2com: Collaborative perception via learnable handshake communication. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6876–6883, 2020. 3

[7] Zhuang Liu, Zhiqiu Xu, Hung-Ju Wang, Trevor Darrell, and Evan Shelhamer. Anytime dense prediction with confidence adaptivity. *International Conference on Learning Representations (ICLR)*, 2022. 7

[8] Tsun-Hsuan Wang, Sivabalan Manivasagam, Ming Liang, Binh Yang, Wenyuan Zeng, James Tu, and Raquel Urtasun. V2vnet: Vehicle-to-vehicle communication for joint perception and prediction. In *ECCV*, 2020. 3

[9] Pengxiang Wu and Siheng Chen. Motionnet: Joint perception and motion prediction for autonomous driving based on bird's eye view maps. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11382–11392, June 2020. 2, 5

[10] Runsheng Xu, Zhengzhong Tu, Hao Xiang, Wei Shao, Bolei Zhou, and Jiaqi Ma. Cobevt: Cooperative bird's eye view semantic segmentation with sparse transformers. In *Conference on Robot Learning (CoRL)*, 2022. 3

[11] Runsheng Xu, Hao Xiang, Zhengzhong Tu, Xin Xia, Ming-Hsuan Yang, and Jiaqi Ma. V2x-vit: Vehicle-to-everything cooperative perception with vision transformer. *ArXiv*, abs/2203.10638, 2022. 3

[12] Runsheng Xu, Hao Xiang, Xin Xia, Xu Han, Jinlong Li, and Jiaqi Ma. Opv2v: An open benchmark dataset and fusion pipeline for perception with vehicle-to-vehicle communication. In *ICRA*, 2022. 3, 9