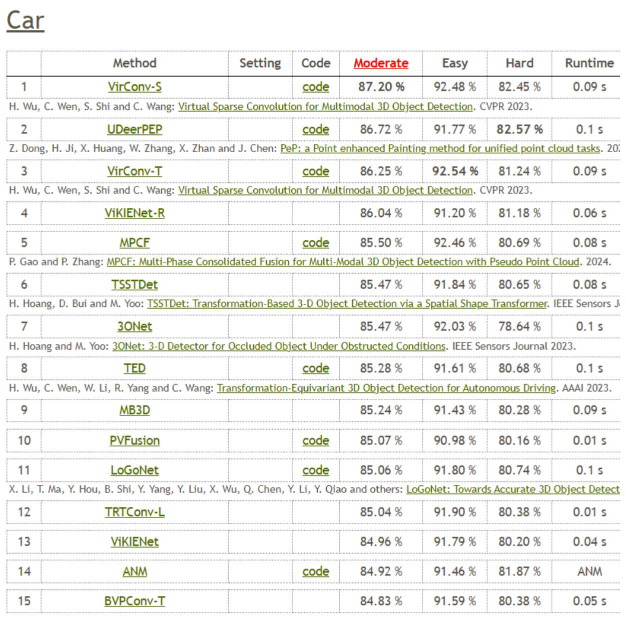
Supplementary Material:
ViKIENet: Towards Efficient 3D Object Detection with Virtual Key Instance Enhanced Network

Overview of this supplementary material:

- **Section 1: Performance Analysis on KITTI Dataset**
- **Section 2: More Results and Discussion**
- **Section 3: Computational Analysis**
- **Section 4: Visualization**

1. Performance Analysis on KITTI Dataset

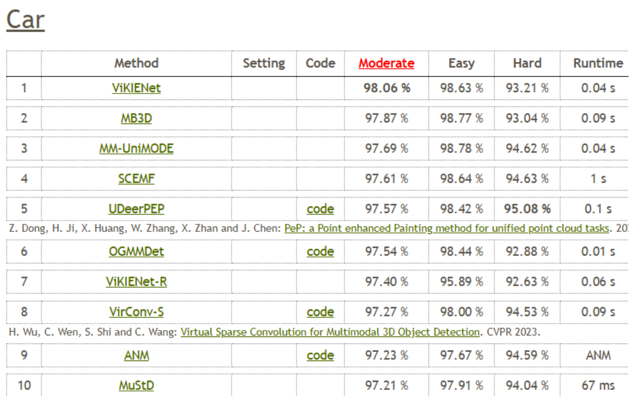
We submitted our results to the test dataset onto KITTI's official evaluation server. As listed in Fig. 1 [4], Fig. 2 [3], and Fig. 3 [3], ViKIENet and ViKIENet-R run at 22.7 and 15.0 FPS, respectively, on an NVIDIA GeForce RTX 3090. As of the CVPR submission deadline (Nov. 15th, 2024), we rank 1^{st} on the KITTI car detection and orientation estimation leaderboard and rank 2^{nd} on the 3D car detection leaderboard among officially published papers. We divide existing methods into two categories, with and without Rotation and Transformation Equivariant Networks (RTEN) [14] as they have been shown to be effective but cumbersome. On the 3D object detection leaderboard, ViKIENet-R, VirConv-T [15], TSSTDet [7], and TED [14] all utilize a similar RTEN. Among these models, due to the highly efficient design of our VKI-based model design, ViKIENet-R runs the fastest, even surpassing the other models in the top-ten list that do not utilize this mechanism, while achieving equivalent accuracy as the state-of-the-art model, VirConv-T [15]. As for VirConv-S, since it uses additional training data, we did not consider it for fair comparison. ViKIENet strikes a balance between speed and performance. Compared to other top-ranked methods such as 3ONet [6] (6.5 FPS on RTX 3090) and LoGoNet [11] (10.69 FPS on A100), our ViKIENet demonstrates excellent real-time performance.



Car

	Method	Setting	Code	Moderate	Easy	Hard	Runtime
1	VirConv-S		code	**87.20 %**	92.48 %	82.45 %	0.09 s
H. Wu, C. Wen, S. Shi and C. Wang: Virtual Sparse Convolution for Multimodal 3D Object Detection. CVPR 2023.							
2	UDeerPEP		code	86.72 %	91.77 %	**82.57 %**	0.1 s
Z. Dong, H. Ji, X. Huang, W. Zhang, X. Zhan and J. Chen: PeP: a Point enhanced Painting method for unified point cloud tasks. 20							
3	VirConv-T		code	86.25 %	**92.54 %**	81.24 %	0.09 s
H. Wu, C. Wen, S. Shi and C. Wang: Virtual Sparse Convolution for Multimodal 3D Object Detection. CVPR 2023.							
4	ViKIENet-R			86.04 %	91.20 %	81.18 %	0.06 s
5	MPCF		code	85.50 %	92.46 %	80.69 %	0.08 s
P. Gao and P. Zhang: MPCF: Multi-Phase Consolidated Fusion for Multi-Modal 3D Object Detection with Pseudo Point Cloud. 2024.							
6	TSSTDet			85.47 %	91.84 %	80.65 %	0.08 s
H. Hoang, D. Bui and M. Yoo: TSSTDet: Transformation-Based 3-D Object Detection via a Spatial Shape Transformer. IEEE Sensors J							
7	3ONet			85.47 %	92.03 %	78.64 %	0.1 s
H. Hoang and M. Yoo: 3ONet: 3-D Detector for Occluded Object Under Obstructed Conditions. IEEE Sensors Journal 2023.							
8	TED		code	85.28 %	91.61 %	80.68 %	0.1 s
H. Wu, C. Wen, W. Li, R. Yang and C. Wang: Transformation-Equivariant 3D Object Detection for Autonomous Driving. AAAI 2023.							
9	MB3D			85.24 %	91.43 %	80.28 %	0.09 s
10	PVFusion		code	85.07 %	90.98 %	80.16 %	0.01 s
11	LoGoNet		code	85.06 %	91.80 %	80.74 %	0.1 s
X. Li, T. Ma, Y. Hou, B. Shi, Y. Yang, Y. Liu, X. Wu, Q. Chen, Y. Li, Y. Qiao and others: LoGoNet: Towards Accurate 3D Object Detect							
12	TRTConv-L			85.04 %	91.90 %	80.38 %	0.01 s
13	ViKIENet			84.96 %	91.79 %	80.20 %	0.04 s
14	ANM		code	84.92 %	91.46 %	81.87 %	ANM
15	BVPConv-T			84.83 %	91.59 %	80.38 %	0.05 s

Figure 1. Top 15 entries of the KITTI 3D car detection leaderboard as of Nov. 20, 2024.



Car

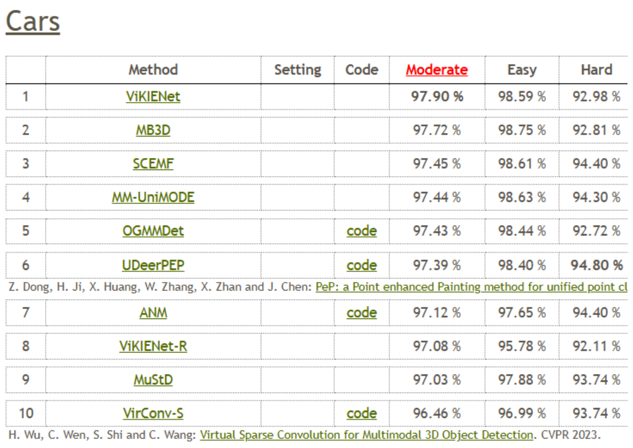
	Method	Setting	Code	Moderate	Easy	Hard	Runtime
1	ViKIENet			**98.06 %**	98.63 %	93.21 %	0.04 s
2	MB3D			97.87 %	98.77 %	93.04 %	0.09 s
3	MM-UniMODE			97.69 %	98.78 %	94.62 %	0.04 s
4	SCEMF			97.61 %	98.64 %	94.63 %	1 s
5	UDeerPEP		code	97.57 %	98.42 %	**95.08 %**	0.1 s
Z. Dong, H. Ji, X. Huang, W. Zhang, X. Zhan and J. Chen: PeP: a Point enhanced Painting method for unified point cloud tasks. 20							
6	OGMMDet		code	97.54 %	98.44 %	92.88 %	0.01 s
7	ViKIENet-R			97.40 %	95.89 %	92.63 %	0.06 s
8	VirConv-S		code	97.27 %	98.00 %	94.53 %	0.09 s
H. Wu, C. Wen, S. Shi and C. Wang: Virtual Sparse Convolution for Multimodal 3D Object Detection. CVPR 2023.							
9	ANM		code	97.23 %	97.67 %	94.59 %	ANM
10	MuStD			97.21 %	97.91 %	94.04 %	67 ms

Figure 2. Top 10 entries of the KITTI 2D car detection leaderboard as of Nov. 20, 2024.

2. More Results and Discussion

2.1. Results of Different Pre-trained Segmentation Models on ViKIENet

We tested a few different segmentation models to study the impact of semantic segmentation results on our model's final 3D detection results. As shown in Tab. 1, there is not much difference among them. Therefore we can choose a segmentation model that prioritizes in efficiency such as BiSeNetV2 which can achieve up to 156 FPS on an NVIDIA GeForce 1080Ti card [17]. We also visualized the differences of the segmentation masks over the input images and observed that they only differ near the edge of the instances but almost all of the pixels within the instances are predicted correctly and thus our SKIS module is able to identify and include most of the virtual points of the instances, and we noticed that the inaccuracy of the final detection results often come from the depth's dimension hence future work could be done to employ state-of-the-art depth completion models.



Cars

	Method	Setting	Code	Moderate	Easy	Hard
1	ViKIENet			97.90 %	98.59 %	92.98 %
2	MB3D			97.72 %	98.75 %	92.81 %
3	SCEMF			97.45 %	98.61 %	94.40 %
4	MM-UniMODE			97.44 %	98.63 %	94.30 %
5	OGMMDet		code	97.43 %	98.44 %	92.72 %
6	UDeerPEP		code	97.39 %	98.40 %	94.80 %
Z. Dong, H. Ji, X. Huang, W. Zhang, X. Zhan and J. Chen: PeP: a Point enhanced Painting method for unified point cl						
7	ANM		code	97.12 %	97.65 %	94.40 %
8	ViKIENet-R			97.08 %	95.78 %	92.11 %
9	MuStD			97.03 %	97.88 %	93.74 %
10	VirConv-S		code	96.46 %	96.99 %	93.74 %
H. Wu, C. Wen, S. Shi and C. Wang: Virtual Sparse Convolution for Multimodal 3D Object Detection. CVPR 2023.						

Figure 3. Top 10 entries of the KITTI car orientation estimation leaderboard as of Nov. 20, 2024.

Method	Segmentation Model	3D AP (R40)		
		Easy	Moderate	Hard
ViKIENet	Mask-RCNN [5]	95.64	88.40	85.98
	Bisenet V2 [17]	95.58	88.49	86.07
	DeepLab V3 [1]	95.35	88.45	86.04
	FCN [12]	95.34	88.43	86.06
	LRASPP [9]	95.29	88.36	86.00

Table 1. Performance comparison of using different segmentation models for ViKIENet on the KITTI validation set.

2.2. Discussion on Different Depth Completion Approaches for ViKIENet.

Following VirConv [15] and other virtual point based methods [14, 15], for a fair performance comparison with them, we followed their convention and did not include the inference time of the depth completion model and used the same pre-trained model, PENet [8], as them for this task. Additionally, our ViKIENet is compatible with different depth completion approaches for a diverse range of use cases. We used IP-Basic [10] for depth completion when working with the JRDB dataset. Its performance is being compared with PENet in Tab. 2. This fast depth completion model can run at 90 Hz on an Intel Core i7-7700K processor [10]. These results suggest that, for applications that focus on close-range perception, for example, autonomous mobile robots (AMRs) or vehicles with low-speed needs such as street sweeper trucks or tour buses, ViKIENet can be deployed with depth completion approaches with only small computational overhead, while achieving significantly higher detection results compared to the baseline. RGB-D cameras can also be used to work with ViKIENet for real-time applications.

Method	RMSE	MAE	iRMSE	iMAE
PENet [8]	730.08	210.55	2.17	0.94
IP-Basic [10]	1288.46	302.60	3.78	1.29

Table 2. Performance comparison between PENet [8] and IP-Basic [17]

VIFF		VIFF-R	
Attention mechanism	AP (R40)	Attention mechanism	AP (R40)
None	85.79	None	87.21
Unidirectional cross-attention: A to B	88.07	Unidirectional cross-attention: A to B	88.14
Unidirectional cross-attention: B to A	88.10	Unidirectional cross-attention: B to A	89.05
Bi-directional cross-attention	88.49	Bi-directional cross-attention	89.52

Table 3. Ablation study on the KITTI validation set. A to B refers to VKI feature as Query. B to A refers to LiDAR feature as Query.

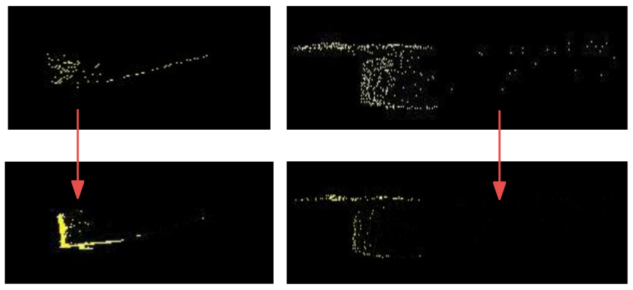


Figure 4. Noise suppressed and contour strengthened by VIRA in feature map visualization.

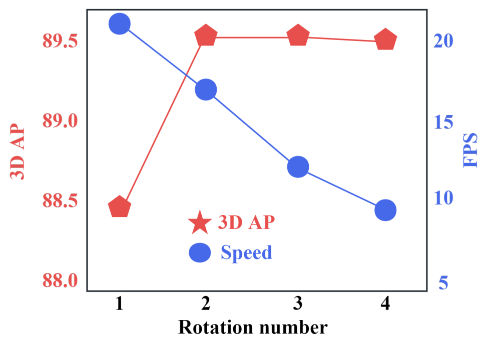


Figure 5. 3D AP and speed trade-off by using different rotation numbers.

2.3. More Qualitative and Quantitative Experiments on VIFF and VIRA.

Fig. 4 illustrates that VIRA can utilize lidar points to emphasize the contour of actual objects, both (a) and (b) show VIRA and VIFF can help to reduce boundary noise issues by suppressing depth-inaccurate features, resulting in more accurate bounding boxing predictions. To justify the effectiveness of the bi-directional (named after [13]) cross-attention, we conduct ablation studies in Tab. 3, comparing VIFF and VIFF-R with their unidirectional variants.

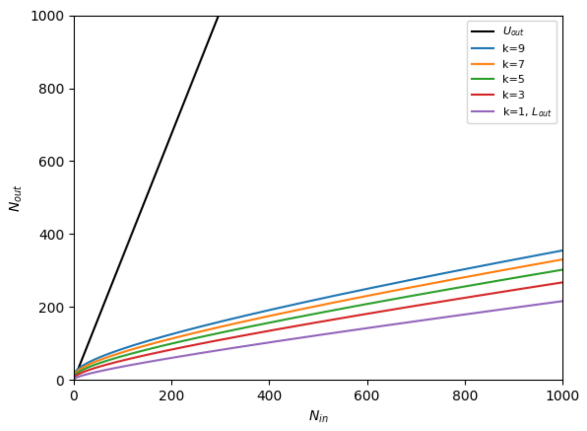


Figure 6. Upper bound of N_{out} and lower bounds of N_{out} compared to N_{in} with k at 1, 3, 5, 7, 9

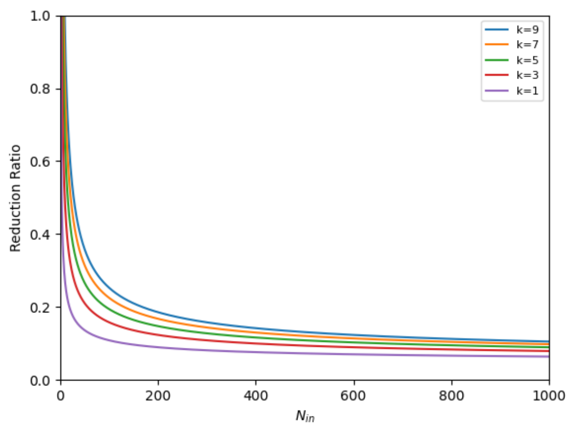


Figure 7. Further reduction ration with the increase in number of inputs

2.4. Discussion of ViKIENet-R

As introduced in the paper, ViKIENet-R utilizes VIFF-R module for feature fusion. Inspired by the approach in TED [14], we also included rotation-transformation equivariant features for our VKIs. The key difference between VIFF-R and VIFF resides in replacing the self-attention module with cross-attention across different rotational features, enabling better fusion of multiple rotational features. Fig. 5 illustrates the precision under different rotation counts. We observe that ViKIENet-R achieves its highest accuracy with a rotation number of 2 and 3, we chose 2 rotations to maintain its high efficiency since it is much faster. To further evaluate the performance of our ViKIENet-R in terms of speed and GPU memory usage, we tested Voxel-RCNN, VirConv-T, and ViKIENet-R on the KITTI validation set using an Nvidia RTX 3090 GPU, with the batch size set to be 16. As shown in Tab. 4, ViKIENet-R achieves significant improvements in both memory usage and inference speed compared to VirConv-T [15].

Method	Memory Usage	FPS	Device	AP (R40)
Voxel-RCNN [2]	6.3G	24.3	RTX 3090	85.29
VirConv-T [15]	13.3G	9.9	RTX 3090	89.87
ViKIENet-R (Ours)	11.5G	15.0	RTX 3090	89.52

Table 4. Performance comparison with Voxel-RCNN [2] and VirConv-T [15] on Nvidia RTX 3090.

3. Computational Analysis

Matrix multiplications over a large input dimension can build up the FLOPs quickly, which can hinge a model's inference speed. With rotation employed for additional feature enrichment, albeit its general ability to further improve the 3D detection accuracy, the benefits comes with a price of increased time consumptions [7, 14, 15]. Our VKI-based approach can significantly mitigate this issue.

To measure how many FLOPs can we save, we attempted with several open-sourced FLOPs analyzing tools but we were unable to apply them due to the customized sparse tensor object [16] being used as inputs throughout our model. Therefore we did our own calculations and obtained interesting findings particularly in the sparse convolution stage since the calculation is not trivial as compared to regular dense convolutions.

Two types of sparse convolution have become popular in recent years: sparse convolution (SparseConv) and submanifold sparse convolution (SubMConv) [16]. They are

	Number of Voxels (in millions)		
	ViKIENet	Random Discard	VirConv-T [15]
Input Layer	19.53	19.53	53.60
After SparseConv1	25.67	59.00	140.81
After SparseConv2	16.09	68.32	129.30
After SparseConv3	8.19	45.27	67.63
	Growth Factor		
Input Layer	-	-	-
After SparseConv1	1.31	3.02	2.63
After SparseConv2	0.82	3.50	2.41
After SparseConv3	0.42	2.32	1.26

Table 5. Number of voxels before and after each sparse convolution layer, and growth factor compared across ViKIENet, random discard and Virconv-T [15]

	MAC	ViKIENet	Random Discard	VirConv-T [15]
SubM0+SpConv1	$\mathrm{N}_{in} * 36 * 76$	405.00	405.00	1111.40
SubM1+SpConv2	$\mathrm{N}_{out_1} * 140 * 576$	2070.39	4757.89	11355.20
SubM2+SpConv3	$\mathrm{N}_{out_2} * 368 * 576$	3410.07	14482.39	27408.13
SubM3	$\mathrm{N}_{out_3} * 176 * 576$	829.88	4589.42	6856.46
Total	-	6715.34	24234.66	46731.19
MAC Ratio	-	0.14	0.52	1.00

Table 6. Performance on different distances and occlusion degrees.

usually employed together with a sparse convolution layer followed by several submanifold convolution layers s to enlarge receptive filed while maintaining sparsity. In sparse convolution, it calculates if any part of the kernel covers an active (non-zero) input site, which may increase the density in the output matrix. In submanifold sparse convolution, it only calculates when the center of the kernel covers an active input site, which allows the output to have the same sparsity with the input matrix, exactly the same when stride is one and same size zero padding is applied.

According to the implementation of SparseConv, the number of multiplication is approximately the same regardless of the distribution when the number of active sites in the input are the same and when stride is one, when stride is more than one it is more random. However, the number of outputs will differ when the input distributions are different, which will then affect the number of multiplications in all of its subsequent submanifold sparse convolution, and other subsequent processing layers. The difference becomes more significant when rotation is applied. Take Virconv [15] for example, there are about 12 SubMConv layers and 3 SparseConv after the first SparseConv layer. If using 2 additional rotation, it would have a total of 36 SubMConv and 9 SparseConv layers. Regarding to the input distribution, the number of outputs of a sparse convolution is minimum when all the pixels or voxels gather together closest to a square or cube shape and maximum when each of them is scattered and no two pixels or voxels are within one kernel's range. In our case of VKIs, since they are dense virtual points obtained for each instance, the distribution will look like several clusters of voxels. Therefore its number of outputs will be much lower than same amount of input voxels but evenly distributed in the space, which means even though our number of input voxels is close one third (36.4% to be exact) of those using all virtual points as [15] right before entering the 3D backbone stage, the actual performance speedup in terms of the number of matrix multiplications will be even lower than one third.

Taking a simple example of 2 non-zero elements that are not near the edge as input in a 1D sparse convolution of kernel size 3 and stride 1, when they are adjacent to each other, the number of output non-zero elements, N_{out}, will be 4; when they are at least 2 spaces apart, N_{out} will be 6; and when they are one space apart, N_{out} will be 5. Formally, in 3D sparse convolution, the lower bound of N_{out}, termed L_{out} can be approximated at

$$\left(\frac{\sqrt[3]{N_{\text{in}}}-k+p}{s}+1\right)^3, \tag{1}$$

and the upper bound of N_{out}, termed U_{out} can be approximated at

$$N_{\text{in}} \times \left(\frac{1-k+p}{s}+1\right)^3. \tag{2}$$

When stride is 1 and kernel size is 3, if the cube is near the edge, L_{out} will be even smaller, assuming the cube is not close to the edge for the purpose of getting a tighter bound, L_{out} can be approximated at

$$\left(\sqrt[3]{N_{\text{in}}}+2\right)^3 \tag{3}$$

which is equal to

$$N_{\text{in}} + 6N_{\text{in}}^{\frac{2}{3}} + 12N_{\text{in}}^{\frac{1}{3}} + 8 \tag{4}$$

and U_{out} can be approximated at $27N_{\text{in}}$, which is quite large and hence stride 1 is not often used.

When kernel size is 3 and stride is 2, and assuming equal probabilities of the voxel being at odd or even position in its coordinates, i.e. each combination of odd or even position in the x, y, and z axis has 1/8 of probability. U_{out} can be approximated at

$$N_{\text{in}} \times \frac{8+4\times C_3^1+2\times C_3^2+1}{8}, \tag{5}$$

which is equal to $3.375N_{\text{in}}$. We define growth factor as the number of output voxels divded by the number of input voxels at the first input layer of the 3D backbone. As shown in table 13, after the first Sparse convolution layer, the random discard method grows 3.02 times of its input number of voxels, close to the 3.375 as our hypothesis. VirConv-T [15], which has more voxels but with a similar distribution as the random discard method, grows 2.63 times compared to its input voxels while our method only grows 1.31 times. After the second and third sparse convolution layer, though less drastically, the other two methods still increased the amount of voxels compared to their corresponding input voxels while ours reduced more than half of its input voxels.

Similar to our previous approximation, assuming equal probabilities of even and odd size of edge and equal probabilities of even and odd position in each axis, L_{out} can be approximated at

$$\left(\frac{\sqrt[3]{N_{\text{in}}}}{2}+1\right)^3 \tag{6}$$

or

$$0.125N_{\text{in}} + 0.75N_{\text{in}}^{\frac{2}{3}} + 1.5N_{\text{in}}^{\frac{1}{3}} + 1. \tag{7}$$

Conveniently, if there are k instances, assuming equal sizes, N_{out} can be approximated at

$$k\left(\frac{\sqrt[3]{\frac{Nin}{k}}}{2}+1\right)^3 \tag{8}$$

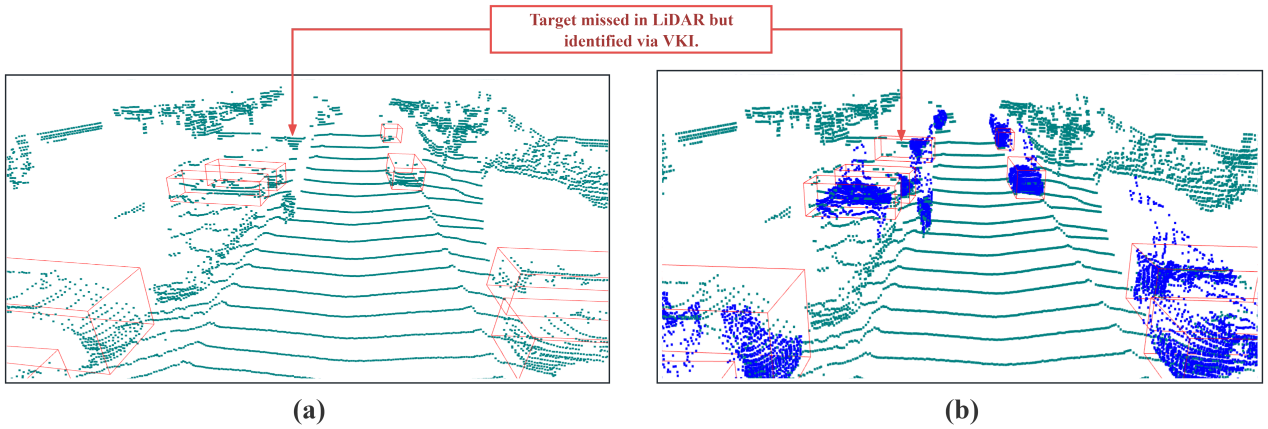


Figure 8. Visualization of detection results on KITTI validation set. Fig. 8 (a) shows the detection results of Voxel-RCNN [2], while Fig. 8 (b) presents the detection results of our ViKIENet.

Practically, k has an average of 5 in the KITTI dataset.

As shown in Fig. 6, L_{out} and N_{out} with k being at different values are significantly lower than U_{out}.

Suppose the additional reduction ratio r is defined as L_{out} / U_{out}, at k=1, max of r can be approximated as

$$\frac{1}{27}\left(1+6x^{-\frac{1}{3}}+12x^{-\frac{2}{3}}+\frac{8}{x}\right) \tag{9}$$

Similarly we can approximate when k is not 1. As shown in Fig. 7, theoretically r is superlinear. However in reality our number of voxels are in the magnitude of 10^2 or 10^3, thus the max additional reduction ratio is approximately between 0.1 to 0.05.

Interestingly, when N_{in} is 132 (if voxel size is 0.05m of each edge, this is approximately a cube of 0.25m as its edge, which is similar to the size of a small document box) L_{out} is 1/10 of U_{out} meaning up to 3.6% compared to worst case distribution in addition to the 36.4% reduction in number of input voxels we already have at the beginning of the 3D backbone stage, and when N_{in} is 1,000, L_{out} is 0.064 of U_{out}, meaning up to 2.3% of worst case distribution. Obviously, it is unlikely for that distribution to happen but it should lean towards L_{out} rather than U_{out} in the beginning stages, and then as the strides increases, the voxels will become closer to each other and thus lean more towards U_{out}. Note that channel size will also increases in the later stages, meaning increased MAC (Multiply-Accumulate Operations) and FLOPs (Floating Point Operations). To test our hypothesis, we applied random voxel discard on the voxels generated from the entire virtual point cloud with a discard rate at 0.36 to match the number of input voxels with our VKIs, averaged over 10 times and over all the 3.7k training samples from the KITTI dataset and obtained the results in Table 14. Though ViKieNet has about 1/3 of the number of input voxels compared to VirConv-T [15], we only need 14% of its MAC in the 3D backbone stage. We also calculated the FLOPs taking account into the batch norm layers and the percentage remains the same. The significance of this calculation is that the more rotation, reflection or other transformation is added, the more advantage we will have compared to VirConv-T [15] or other models with transformation-equivariant mechanisms over the entire virtual point clouds. To confirm this conclusion, we tested the inference time with rotation numbers from 1 to 4 comparing ViKIENet and VirConv-T [15], results are shown in Tab. 7 below. We only need 20, 23, 20ms for each additional rotation while VirConv-T need 32, 30, and 40 ms for each additional rotation.

Number of Rotations	**ViKIENet-R (Ours)**	**VirConv-T [15]**
1	46 ms	78 ms
2	66 ms	110 ms
3	89 ms	140 ms
4	109 ms	180 ms

Table 7. Inference time (ms) over 1, 2, 3, and 4 rotations on a single Nvidia RTX 3090

Another interesting finding is that during the hashing process for output locations, when N_{in} is the same but distribution is less scattered, the reduced number of active outputs equals the increased number of addition required, because we need to aggregate the values on the same output location if there are more than one, which means if there are p fewer output locations, there is a slight time reduction of $p\Delta t$ where $\Delta t = h_t - a_t$, h_t is the time of building hash table for each output location, and a_t is the number of addition, building hash table has to be run sequentially while addition can be run in parallel and addition is gener-

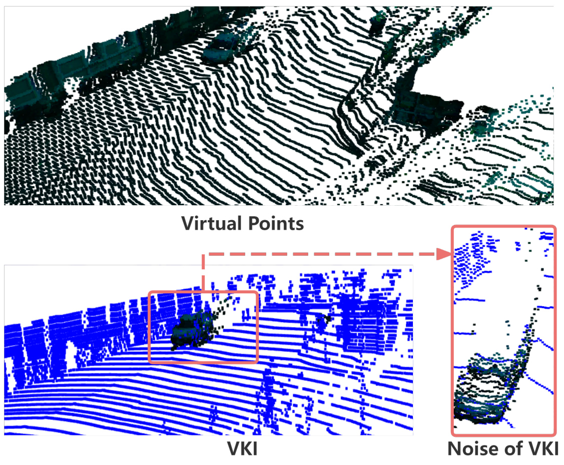


Figure 9. The virtual points and LiDAR points are shown in black and blue, respectively. The image below shows a vehicle with noise caused by inaccurate depth completion.

ally very fast hence though very small, Δt should be greater than zero.

4. Visualization

Fig. 8 compares results between our ViKIENet and the baseline model (Voxel-RCNN) [2] and demonstrates that since our model contains semantic information and denser geometric information, it can detect occluded targets which tend to be missed. Fig. 9 gives an illustration of the noise generated with the VKIs that we have been discussing throughout the paper. While VKIs effectively compensate for missing regions in the point cloud, they exhibit two main drawbacks which are also common among other virtual-points-based methods: excessive density and noise caused by inaccurate depth completion. Fig. 10 presents an example which shows the effectiveness of our proposed ViKIENet. In Fig. 10 (a), the object can be easily identified from the image, whereas in Fig. 10 (b), it is mistakenly detected by Voxel-RCNN [2]. Nonetheless, in Fig. 10 (c) our model successfully recognized that it is not a vehicle, because our VKIs include semantic information, ViKIENet is able to distinguish between similarly shaped objects from its texture and color pattern, enabling higher precision in 3D object detection results.

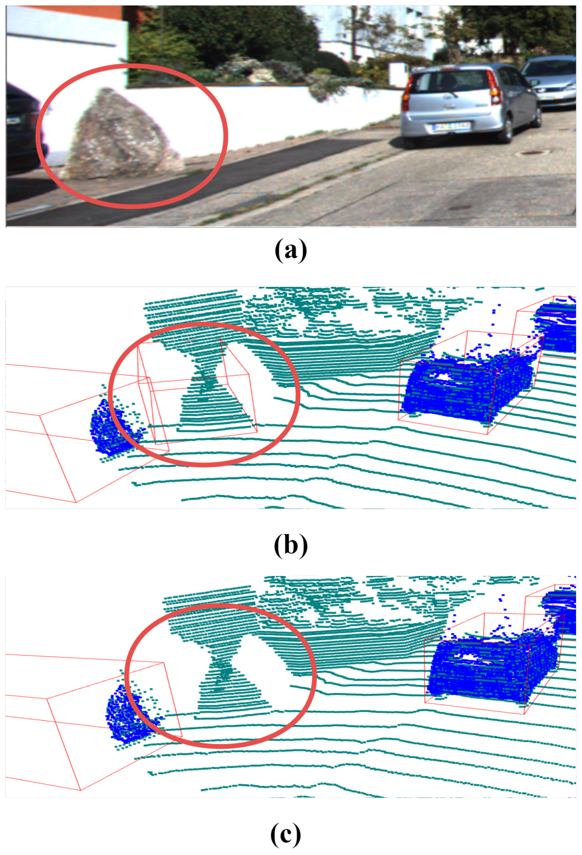


Figure 10. Visualization of detection results on the KITTI validation set. Fig. 10 (a) presents a sample image, Fig. 10 (b) shows the detection results from Voxel-RCNN [2], and Fig. 10 (c) illustrates the results from ViKIENet.

References

[1] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017. 2

[2] Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wengang Zhou, Yanyong Zhang, and Houqiang Li. Voxel R-CNN: Towards high performance voxel-based 3D object detection. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1201–1209, 2021. 3, 5, 6

[3] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Kitti vision benchmark suite. `https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d`, . Accessed: November 20, 2024. 1

[4] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Kitti vision benchmark suite. `https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d`, . Accessed: November 20, 2024. 1

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proc. IEEE International Conf. on Computer Vision (ICCV)*, pages 2980–2988. IEEE, 2017. 2

[6] Hiep Anh Hoang and Myungsik Yoo. 3onet: 3D detector for occluded object under obstructed conditions. *IEEE Sensors Journal*, 2023. 1

[7] Hiep Anh Hoang, Duy Cuong Bui, and Myungsik Yoo. TSSTDet: Transformation-based 3-D object detection via a spatial shape transformer. *IEEE Sensors Journal*, 2024. 1, 3

[8] Mu Hu, Shuling Wang, Bin Li, Shiyu Ning, Li Fan, and Xiaojin Gong. PENet: Towards precise and efficient image guided depth completion. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, pages 13656–13662. IEEE, 2021. 2

[9] Brett Koonce and Brett Koonce. Mobilenetv3. *Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization*, pages 125–144, 2021. 2

[10] Jason Ku, Ali Harakeh, and Steven L Waslander. In defense of classical image processing: Fast depth completion on the CPU. In *Proc. Conf. on Computer and Robot Vision (CRV)*, pages 16–22. IEEE, 2018. 2

[11] Xin Li, Tao Ma, Yuenan Hou, Botian Shi, Yuchen Yang, Youquan Liu, Xingjiao Wu, Qin Chen, Yikang Li, Yu Qiao, and Liang He. LoGoNet: Towards accurate 3D object detection with local-to-global cross-modal fusion. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 17524–17534, 2023. 1

[12] Yi Lu, Yaran Chen, Dongbin Zhao, and Jianxin Chen. Graphfcn for image semantic segmentation. In *Proc. International Symposium on Neural Networks*, pages 97–105. Springer, 2019. 2

[13] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016. 2

[14] Hai Wu, Chenglu Wen, Wei Li, Xin Li, Ruigang Yang, and Cheng Wang. Transformation-equivariant 3D object detection for autonomous driving. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2795–2802, 2023. 1, 2, 3

[15] Hai Wu, Chenglu Wen, Shaoshuai Shi, Xin Li, and Cheng Wang. Virtual sparse convolution for multimodal 3D object detection. In *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, pages 21653–21662, 2023. 1, 2, 3, 4, 5

[16] Yan Yan, Yuxing Mao, and Bo Li. SECOND: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 3

[17] Changqian Yu, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. BiSeNet V2: Bilateral network with guided aggregation for real-time semantic segmentation. *International Journal of Computer Vision*, 129: 3051–3068, 2021. 1, 2