Mask-Attention-Free Transformer for 3D Instance Segmentation
Supplementary Material

Introduction

This is the supplementary material, which is divided into the following sections.

1. More experiments are shown in Sec. 1.
2. Efficiency comparison is given in Sec. 2.
3. A memory-efficient implementation for RPE is introduced in Sec. 3.
4. We elaborate on the implementation details for the spatial distribution of the ground truths in Sec. 4.
5. We give more visualizations for the spatial distribution of the ground truths in Sec. 5.
6. More visual comparisons are shown in Sec. 6.
7. Limitation analysis and future work are shown in Sec. 7.

1. More Experiments

1.1. Effect of Different Weights on the Auxiliary Center Regression Task

weight	mAP	mAP_{50}	mAP_{25}
0.25	57.3	74.4	84.1
0.50	**58.4**	**75.9**	**84.5**
0.75	57.6	75.2	84.1
1.00	57.6	75.4	84.0

Table 1. Ablation study of the weights on the auxiliary center regression task.

As shown in Table 1, we analyze the effect of different weights on the auxiliary center regression task. It shows that the performance is highest when setting the weight to 0.5.

1.2. Effect of Different Grid Sizes on the Relative Position Encoding

grid size (m)	mAP	mAP_{50}	mAP_{25}
0.025	57.6	74.7	83.3
0.05	57.5	75.3	83.7
0.1	**58.4**	**75.9**	**84.5**
0.2	58.1	75.8	84.2

Table 2. Ablation study of the weights on the auxiliary center regression task.

Also, we investigate the effect of different grid sizes on the relative position encoding (RPE) in Table 2. It reveals that setting it to 0.1m performs best. This result indicates that setting the grid size too high may cause fine-grained information to be lost, while setting it too low may lead to a small space covered by the RPE.

1.3. Random Initializing and Freezing the Position Queries During Training

Method	mAP	mAP_{50}	mAP_{25}
frozen	58.1	75.7	84.5
learnable	**58.4**	**75.9**	**84.5**

Table 3. Ablation on making the position queries learnable or frozen during training.

As shown in Table 3, we conduct an ablation study on making the position queries learnable or frozen during training. The result shows that making them learnable only brings slight performance improvement. Given a random initialized position queries, just freezing them works sufficiently well. This finding demonstrates the robustness of our method.

2. Efficiency Comparison

Method	SPFormer	Ours
Inference Time (ms)	72	69

Table 4. Efficiency comparison with the previous state-of-the-art method (*e.g.*, SPFormer [1]). We randomly sample a scene from ScanNetv2 *val* set and list the inference time on a single RTX3090 GPU.

To show the superiority of our method, we compare the efficiency with previous work [1] as shown in Table 4. It is chosen because it has achieved the best performance on ScanNetv2 *val* set so far. We randomly select a scene and compare the inference time. The results show that our approach demonstrates inference speed on par with SPFormer.

3. A Memory-efficient Implementation of RPE

In Fig. 5 of the submission file, we illustrate the relative position encoding (RPE), where we utilize the relative positions to obtain the integer indices $\hat{\mathbf{r}}_x, \hat{\mathbf{r}}_y, \hat{\mathbf{r}}_z \in \mathbb{R}^{n\times N}$ to look up the tables of position encoding $\mathbf{t}_x, \mathbf{t}_y, \mathbf{t}_z \in \mathbb{R}^{L\times d}$, respectively. We then sum them up to yield $\mathbf{f}^{pos} \in \mathbb{R}^{n\times N\times d}$, which is followed by dot product with query features $\mathbf{f}^q \in \mathbb{R}^{n\times d}$ or key features $\mathbf{f}^k \in \mathbb{R}^{N\times d}$ to yield the final positional bias $\mathbf{pos\_bias} \in \mathbb{R}^{n\times N}$. However, the resultant space complexity is

$$O(3\times n\times N\times d) = O(3nNd), \tag{1}$$

which is impractical for current hardware (*i.e.*, GPUs) to accommodate.

Therefore, practically, we design a memory-efficient implementation for RPE. We first use the position encoding tables $\mathbf{t}_x, \mathbf{t}_y, \mathbf{t}_z \in \mathbb{R}^{L\times d}$ to perform dot product with key features $\mathbf{f}^k \in \mathbb{R}^{N\times d}$ to yield positional bias tables $\tilde{\mathbf{t}}_x, \tilde{\mathbf{t}}_y, \tilde{\mathbf{t}}_z \in \mathbb{R}^{L\times N}$. (The same operation is for query features except that we yield positional bias tables of the shape $\mathbb{R}^{L\times n}$.) We then let the integer relative positions $\hat{\mathbf{r}}_x, \hat{\mathbf{r}}_y, \hat{\mathbf{r}}_z \in \mathbb{R}^{n\times N}$ to look up the positional bias tables respectively, and sum them up to obtain the final positional bias $\mathbf{pos\_bias} \in \mathbb{R}^{n\times N}$. The result is equivalent to that of the vanilla implementation, since

$$(\mathbf{t}_x[\hat{\mathbf{r}}_x] + \mathbf{t}_y[\hat{\mathbf{r}}_y] + \mathbf{t}_z[\hat{\mathbf{r}}_z]) \cdot \mathbf{f}^k = (\mathbf{t}_x \cdot \mathbf{f}^k)[\hat{\mathbf{r}}_x] + (\mathbf{t}_y \cdot \mathbf{f}^k)[\hat{\mathbf{r}}_y] + (\mathbf{t}_z \cdot \mathbf{f}^k)[\hat{\mathbf{r}}_z]. \tag{2}$$

Compared to the vanilla implementation, it reduces the space complexity to

$$O(L\times N\times d) = O(LNd). \tag{3}$$

In our setting, L is set to 48 while n can be several hundreds, therefore, we have

$$O(LNd) \ll O(3nNd). \tag{4}$$

4. Implementation Details of Spatial Distribution Visualization

As shown in Fig.6 in the submission file, we demonstrate the spatial distribution of the ground truths that are matched to the same query. Specifically, we randomly choose 10 queries and record their matched ground truths at different training iterations. Then, the three plots for each method are the x, y, and z-axis of the center coordinates of the matched ground truths, respectively. The darker point denotes the later training iteration. It is worth noting that a query may often match no ground truth at some training iterations. For this situation, we simply do not draw a scatter point for the query at that training iteration. As a result, different queries may correspond to different numbers of scatter points.

5. More Spatial Distribution Visualizations

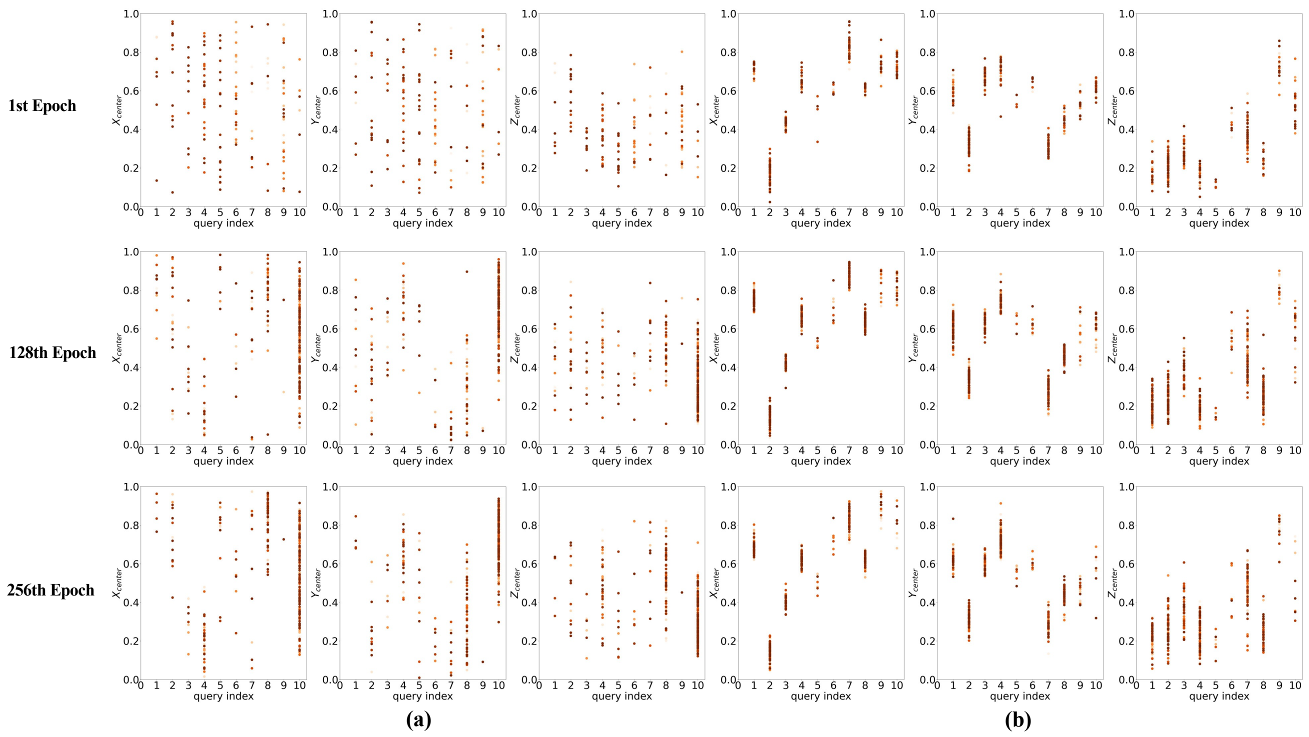
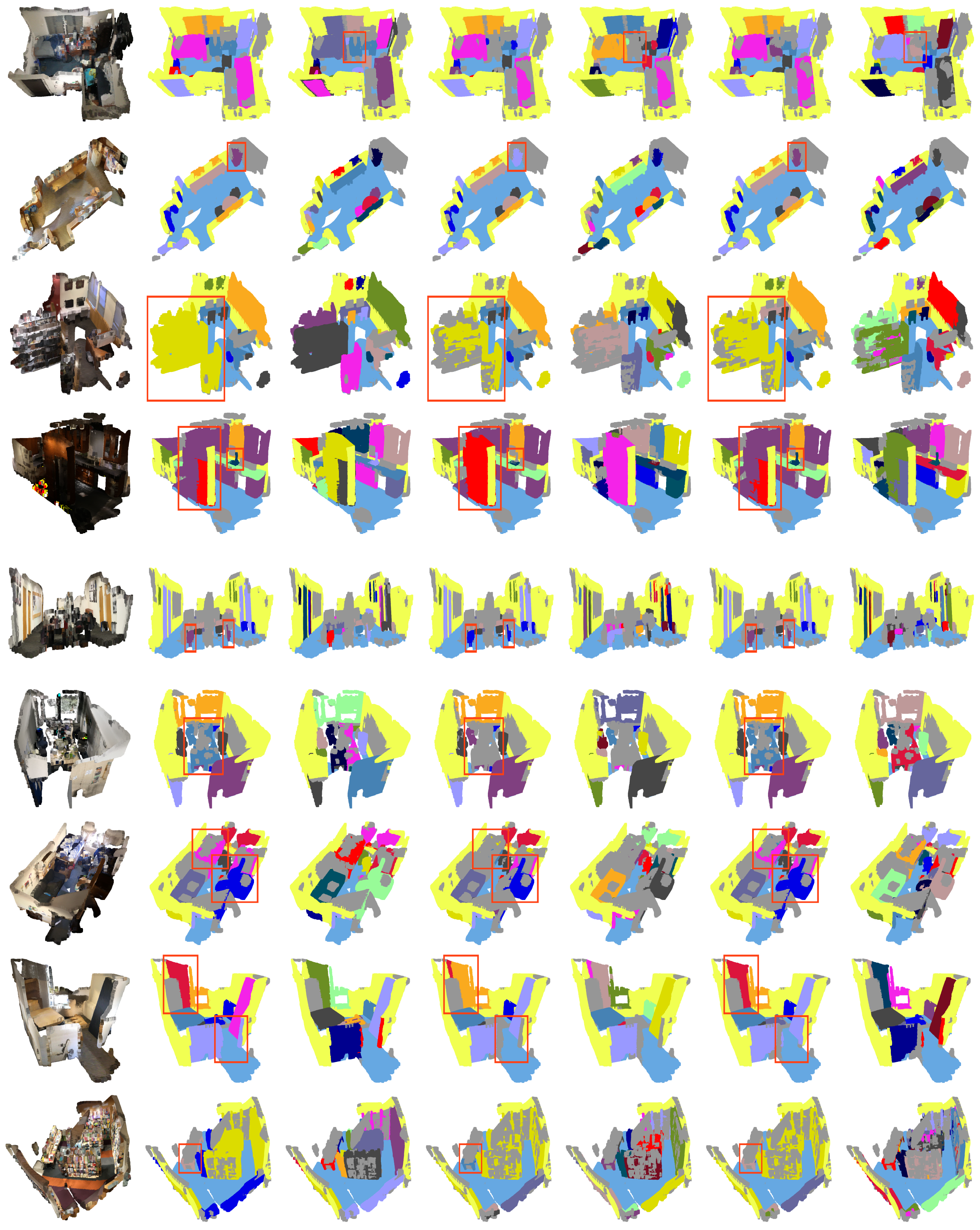


Figure 1. More spatial distribution of ground truths that are matched to the same query at different training iterations. (a): Baseline. (b): Ours. Top: the 1st epoch result. Middle: the 128th epoch result. Bottom: the 256th epoch result. We randomly select 10 object queries and record their matched ground-truth instances in the early training. The three scatter plots for each method represent the center XYZ coordinates of their matched ground-truth instances at different training iterations (the scatter points for a query correspond to different training iterations)..

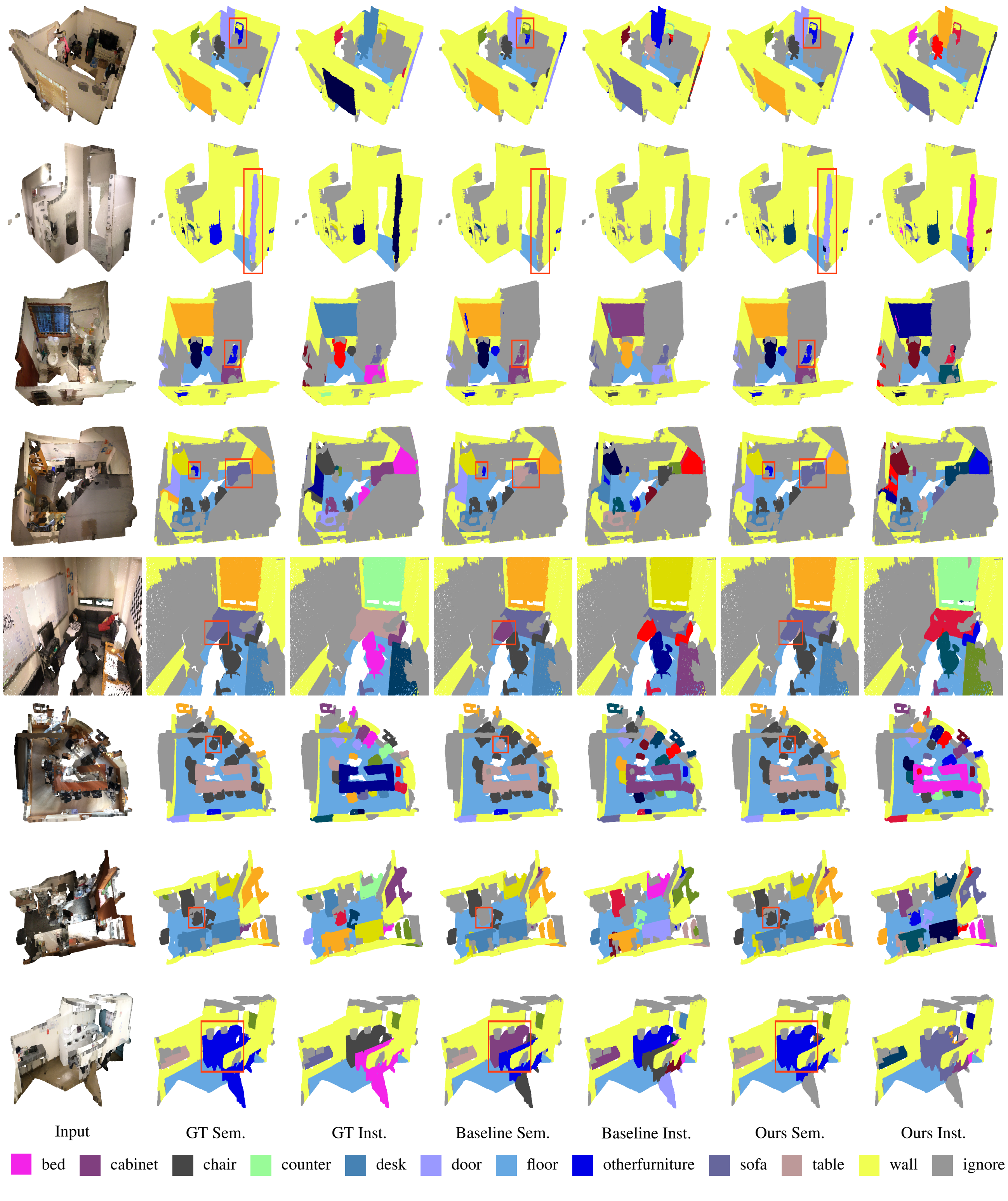
As shown in Fig. 1, we also show more spatial distribution visualizations (*i.e.*, at the 128th and 256th epochs). It demonstrates that the positional matching consistency is maintained in our method throughout the training. Even when training is mature (*e.g.*, 256th epoch), our approach still manifests more consistent matching results than the baseline model.

6. More Examples of Visual Comparison

As shown in Fig. 2, more examples of visual comparison are demonstrated. It is notable that our method tends to make correct predictions on the semantic class of the instances. Moreover, our method is able to predict instances that are missed with the baseline method. Again, they reveal the superiority of our approach consistently.



🟪	bed	🟪	cabinet	⬛	chair	🟩	counter	🟦	desk	🟪	door	🟦	floor	🟦	otherfurniture	🟪	sofa	🟫	table	🟨	wall	⬜	ignore


Input
GT Sem.
GT Inst.
Baseline Sem.
Baseline Inst.
Ours Sem.
Ours Inst.
bed
cabinet
chair
counter
desk
door
floor
otherfurniture
sofa
table
wall
ignore

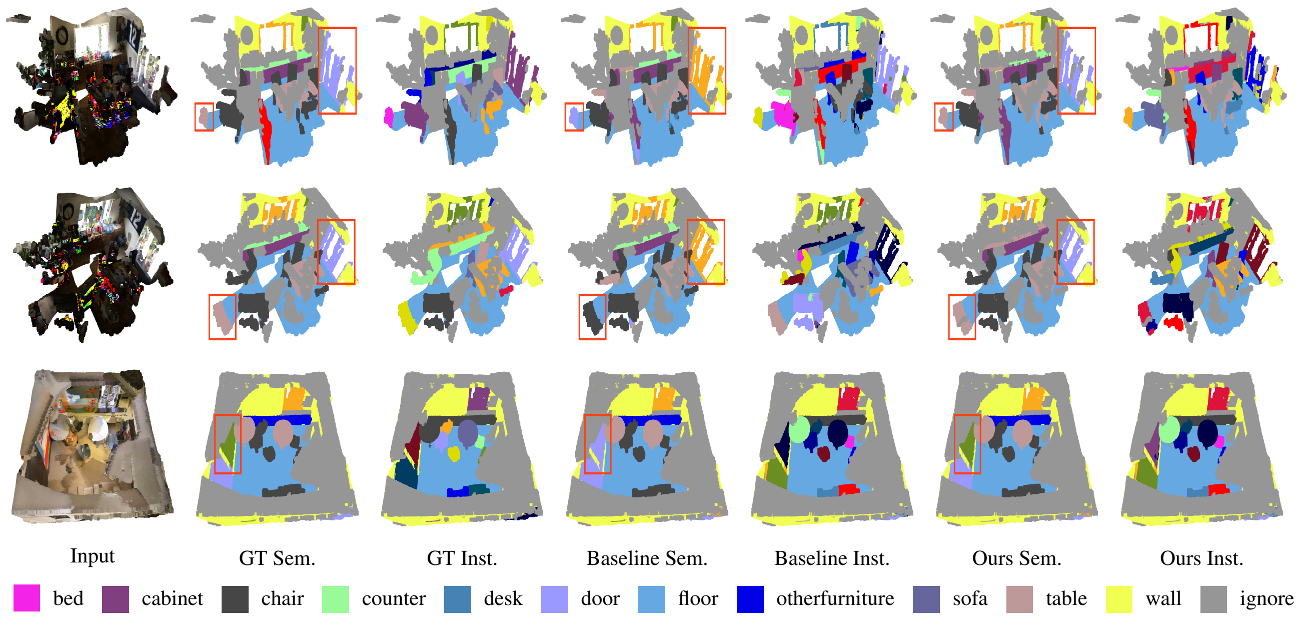


Figure 2. Visual comparison between baseline and ours (best viewed in color and by zoom-in). GT: Ground Truth. Sem.: Semantic labels. Inst.: Instance labels. The main difference is highlighted with a red bounding box. The bottom color map is for semantic labels.

7. Limitations and Future Work

Limitations. Firstly, since we employ initial position queries to match the corresponding ground-truth instances, this pipeline works well for the instance-level tasks (*e.g*., instance segmentation), but it is hard to extend to point-level tasks (*e.g*., semantic segmentation). Because there may be many instances for a single semantic class, and their center coordinates make no sense. Secondly, the memory-efficient implementation of relative position encoding (RPE) relies on our customized CUDA implementation, which needs further processing and optimization for practical use and deployment.

Furture work. Firstly, we plan to make our work extend to outdoor scenes (*e.g*., panoptic segmentation task) since currently we only focus on the indoor scenario. Moreover, we will optimize the CUDA implementation for RPE with more advanced techniques including shared memory, coalesced memory, etc.

References

[1] Jiahao Sun, Chunmei Qing, Junpeng Tan, and Xiangmin Xu. Superpoint transformer for 3d scene instance segmentation. *AAAI*, 2023. 2