Appendix

A Details of Network Architecture

We provide additional information about our network architecture. The backbone of our network is based on a standard PyTorch implementation of multi-scale grouping PointNet++. It is followed by a single MLP layer that extracts a D-dimensional feature for each point. In our implementation, we set $D = 128$.

The detailed structure of the transformer module can be seen in Fig. 5. The self-attention layer follows the official code of the point transformer layer, while the cross-attention layer employs a multi-head attention mechanism with position-wise feed-forward networks. In our configuration, we set the number of attention heads to $h = 8$, the head dimension to $d_h = 16$, and use k-nearest neighbor sampling with $k = 16$. The inner layer of the feed-forward network has a dimensionality of $d_i = 256$.

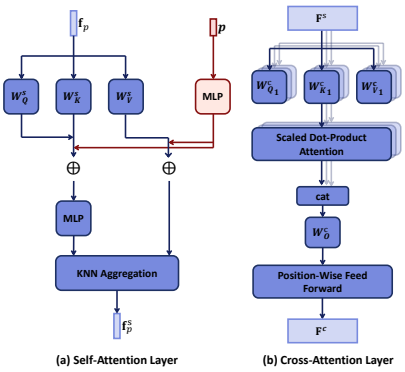

Figure 5: Detailed structure of the attention layers.

Regarding the primal-dual descriptor, we set its feature dimension to $d = 256$.

B Analysis on Primal-dual Descriptor

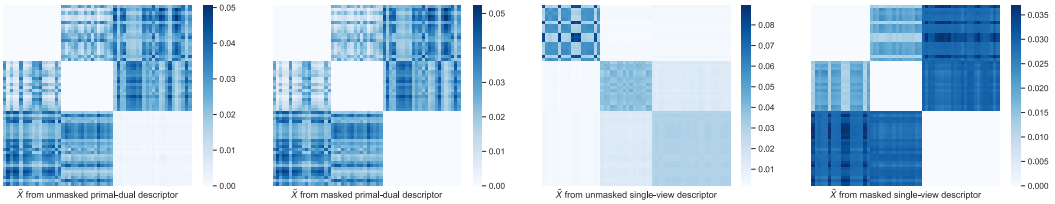

Figure 6: Comparison between the doubly-stochastic soft matching matrix $\tilde{X}$ computed by different types of descriptors.

We conducted an analysis to compare the results obtained using the primal-dual descriptor and the standard single-view descriptor. Before passing the affinity metric computed from the single-view descriptor to the Sinkhorn layer, a common practice is to mask out the diagonal sub-matrix to prevent self-self alignment. For clarity, we refer to the masked and unmasked versions of the primal-dual and single-view descriptors to denote whether this masking operation has been applied or not.

In Fig. 6, we provide a visualization of the doubly-stochastic matrix $\tilde{X}$ computed by the Sinkhorn algorithm using four types of descriptors: (1) the unmasked primal-dual descriptor, (2) the masked primal-dual descriptor, (3) the unmasked single-view descriptor, and (4) the masked single-view descriptor. The visualization depicts an example object with 3 pieces and 58 fracture points. The $\tilde{X}$ obtained from the primal-dual descriptor clearly demonstrates its ability to differentiate between the two viewpoints and avoid aligning a point to itself. In contrast, the single-view descriptor exhibits a diagonal peak, resulting in self-self alignment. Even when the diagonal sub-matrix is masked to prevent self-self alignment, the soft matching computed from the single-view descriptor is less distinct compared to that computed from the primal-dual descriptor.

Furthermore, we conducted evaluation on the `everyday` object subset of Breaking Bad dataset, using the unmasked primal-dual descriptor and the masked primal-dual descriptor. The results indicate that the unmasked primal-dual descriptor achieves comparable performance to the masked one, with only a slight difference: a decrease of $0.2°$ in rotation error (MAE(R)) and an improvement of 0.7×10^{-2} in translation error (MAE(T)). We attribute this minor difference to the fine characteristics of the primal-dual descriptor that prevent self-self alignment, thus resulting in consistent performance.

C Experiment Details

C.1 Dataset

We leverage Breaking Bad dataset [4] to evaluate our method and all baseline methods. As we have stated in section 4, the training of all methods were on the training subset of `everyday`, and testing were on the testing subset of both `everyday` and `artifact` object. Each object within the dataset has been fragmented into pieces, represented by triangle meshes, and all the pieces are in their original poses. By assembling these pieces together directly, the surface of the original object is seamlessly restored. The triangle meshes of the pieces solely consist of exterior faces that are visible from the outside.

In generating the point cloud for our experiments, we employ two sampling strategies for the compared methods: "sampling by piece" and "sampling by object". The "sampling by piece" strategy, originally utilized in the Breaking Bad benchmark [4], involves sampling an equal number of points within each fragment. However, this approach leads to excessively dense sampling on small fragments, while larger fragments suffer from sparser point distributions, resulting in an imbalance in the representation of point density across fragments. To better mirror real-world scanning technology, we opt for the "sampling by object" strategy. With this approach, we sample a fixed number of points within each object, and the number of sampled points for each fragment is determined based on its surface area. In essence, smaller fragments receive fewer points, whereas larger ones receive more, ensuring a more realistic representation. Additionally, we ensure that each fragment is sampled with a minimum of 30 points to include even the tiniest fragments in multi-part matching. Detailed parameter settings can be found in Table 1. Our analysis of the average fragment numbers and experiments conducted using DGL [8] indicate that the choice between the two sampling strategies has negligible impact on the baseline performance.

For each sampled point p on P_i, its fracture label c_p is determined by the distance from p to its nearest neighbor q among points from all other pieces:

$$c_p = \mathbb{1}\left(\min_{q \in O \setminus P_i} \|p - q\|_2 < \eta\right) \tag{15}$$

where η is set to 0.02 for all the objects. The ground-truth matching point $\hat{q}$ of a fracture point $\hat{p} \in \hat{P}_i$ is set to

$$\hat{q} = \underset{\hat{q}' \in \hat{O} \setminus \hat{P}_i}{\operatorname{argmin}} \|\hat{p} - \hat{q}'\|_2. \tag{16}$$

where $\hat{O}, \hat{P}_i$ denotes the set of fracture points in O and P_i. During the training process, c_p was applied to segment the fracture points and during testing the predicted $\tilde{c}_p$ was used instead. An example is shown in Fig. 7.

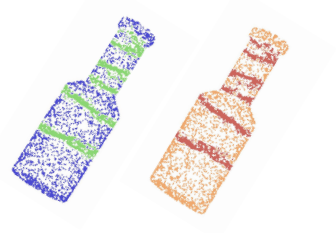

Figure 7: An example of surface segmentation over the point cloud of a bottle broken into 7 major pieces. Fracture points are marked with green in the predicted result of our method (left) and red in the ground-truth (right).

After the ground-truth labeling and matching were computed based on the pieces at their original pose, all pieces were recentered to the origin and a random rotation was applied to each piece.

To ensure a fair comparison with baseline methods [26, 10, 27, 8], we adopted the same implementation as of the benchmark code of [4], which sampled the same number of point each piece. For PREDATOR, we applied the same sample routine as the baseline methods, and we paired the adjacent pieces for training and testing.

C.2 Evaluation Metrics

The mean absolute error MAE(R) and square-rooted mean squared error RMSE(R) of rotation were computed as

$$\begin{aligned} \text{MAE(R)} &= \frac{1}{3}||\tilde{R} - R^{gt}||_1, \\ \text{RMSE(R)} &= \frac{1}{\sqrt{3}}||\tilde{R} - R^{gt}||_2, \end{aligned} \tag{17}$$

Table 4: Comparison of training and influence time with Tesla V100 GPUs. For Jigsaw, the time used in the forward is 1.30s/batch and the rest of the time is used on Hungarian and global alignment.

	Jigsaw	Predator[16]	DGL [33]	LSTM [33]	Global [33]
Training Time / GPUs	120H / 4	96H / 4	11H / 1	16H /1	21H / 1
Influence Speed (s/batch) / batch size	7.67(1.30) / 8	1.28 / 28	2.46 / 32	1.63 / 32	1.65 / 32

Table 5: Detailed quantitative results of Jigsaw on Breaking Bad dataset (mean and STD by 3 runs).

Method	RMSE (R) ↓ degree	MAE (R) ↓ degree	RMSE(T) ↓ $\times 10^{-2}$	MAE (T) ↓ $\times 10^{-2}$	PA ↑ %
Results on the `everyday` object subset.					
Jigsaw	42.3 ± 0.03	36.3 ± 0.06	10.7 ± 0.02	8.7 ± 0.02	57.3 ± 0.012
Results on the `artifact` object subset.					
Jigsaw	52.4 ± 0.09	45.4 ± 0.14	22.2 ± 0.05	19.3 ± 0.04	45.6 ± 0.015

for each piece, where $\tilde{R}$ and R^{gt} were predicted rotation and ground-truth rotation represented in Euler angles. The error over each object was computed as the mean error of all the pieces and the total error was the mean error of all the object. For translation, MAE(T) and RMSE(T) were computed in the same way

$$\begin{aligned} \text{MAE(T)} &= \frac{1}{3}||\tilde{\boldsymbol{t}} - \boldsymbol{t}^{gt}||_1, \\ \text{RMSE(T)} &= \frac{1}{\sqrt{3}}||\tilde{\boldsymbol{t}} - \boldsymbol{t}^{gt}||_2, \end{aligned} \tag{18}$$

where $\tilde{\boldsymbol{t}}$ and $\boldsymbol{t}^{gt}$ were the predicted translation and ground-truth translation.

D Implementation Details

D.1 Additional Configuration of parameters

For PREDATOR [16], we carefully follow the parameters presented in its open sourced code, and the threshold for overlap recall to add saliency loss is set to be 0.3. For Jigsaw, we start training with only the segmentation loss. We add matching loss after first 10 epochs, and rigidity loss after 200 epochs.

D.2 Running time

Our framework is implemented in Pytorch. All methods are trained over Tesla V100-SXM2-32GB GPUs and distributed data parallel strategy is applied for multi-GPU training. The training and influence time for Jigsaw and baseline methods are shown in Table 4.

E Additional Results

E.1 Detailed Quantitative Results

Fig. 8 gives a detailed distribution of each error metric with respect to the number of fractured pieces. Our approach exhibits a clear advantage in each of the metrics evaluated, and this advantage becomes more pronounced as the number of pieces decreases.

In addition, we present the mean and standard deviation (STD) of three runs of our method in Table 5. These values demonstrate the stability of our method across different random seeds and variations in the sampled point cloud. The consistent performance and low standard deviation affirm the robustness of our proposed approach.

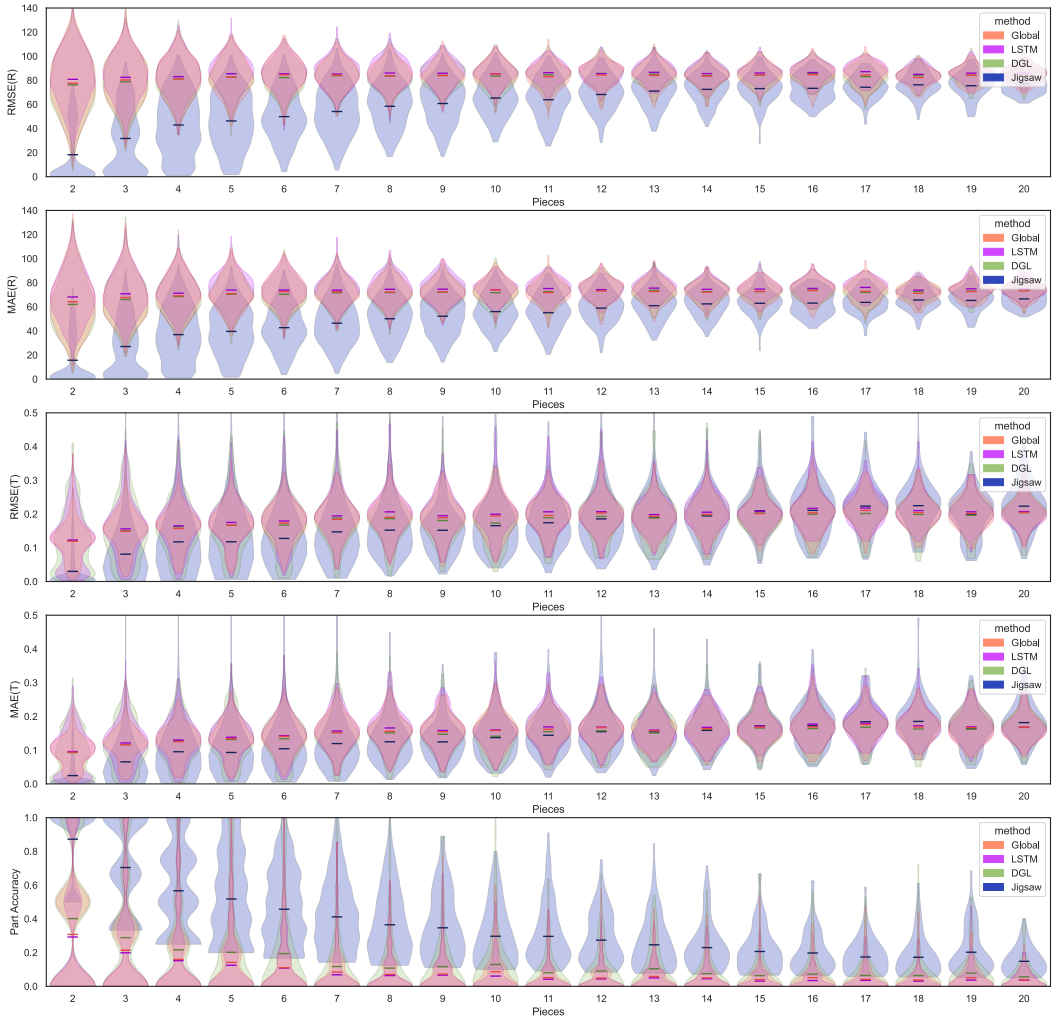

Figure 8: Detailed analysis of the quantitative results obtained by each method on the `everyday` object subset. We provide results for each metric, categorized by the number of pieces involved. The shadowed area in the figures represents the distribution of each metric across the corresponding number of pieces. The horizontal line indicates the mean value of the metric for that specific number of pieces. Different hues represent different methods.

E.2 More Qualitative Results

We collect additional visualizations of the assembled objects in Fig. 9. Even with a large number of pieces to assemble, our method remains robust in restoring major pieces to their original pose.

E.3 Additional Registration Baseline

We have received advice to consider GeoTransformer [30] as a state-of-the-art baseline for low overlap registration. However, training GeoTransformer proves to be exceedingly slow (about 15 days over 4 V100 GPUs), primarily due to the significant scale of Breaking-Bad in comparison to the 3D registration datasets on which it was originally tested. On pairwise transformations GeoTransformer achieves MAE(R)=72.4(degree), RMSE(R)=84.8(degree), RMSE(T)=14.3($\times 10^{-2}$), MAE(T)=11.6($\times 10^{-2}$), PA=3.1%. These findings align with the claims we've put forth in our paper, and we believe that presenting the full results of PREDATOR adequately reflects how registration baselines perform on the assembly task.

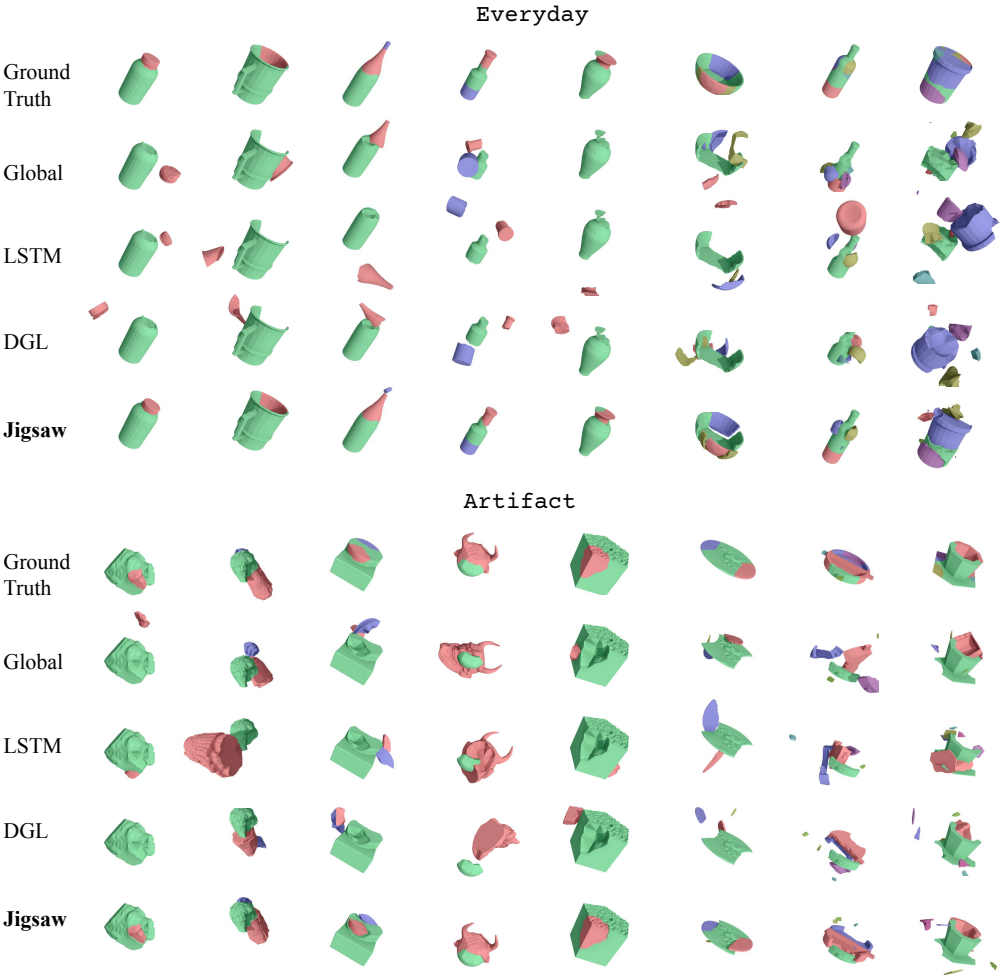

Figure 9: More visualization of assembly results.