O2O-Afford: Annotation-Free Large-Scale Object-Object Affordance Learning (Supplementary Material)

Kaichun Mo[1], **Yuzhe Qin**[2], **Fanbo Xiang**[2], **Hao Su**[2], **Leonidas Guibas**[1]

[1]Stanford University [2]UCSD

https://cs.stanford.edu/~kaichun/o2oafford

COVID-19 Impact: Our paper proposes a perception framework on learning object-object interaction affordance. While the acting object is usually held by human or robot hands, the problem of object-object affordance by itself is agent-free. In addition, literature work demonstrated that visual affordable can be used for several robotics tasks. Thus, we conducted large-scale learning and evaluation on PartNet data, as well as tested our learned model directly on some real-world scans. We think that these evaluations can sufficiently prove the effectiveness of the proposed perception learning framework.

A Overview

This document provides more details, results and visualizations accompanying the main paper. In summary, we include

- more data details and visualization;
- more details about the environment settings;
- more details about the network architecture;
- more details about the training;
- more results of point-wise affordance predictions;
- more results tested on real-world data;
- failure cases and discussions.

We also submit a **video presentation** that provides quick overview and summary of our paper.

We further release our **code with a small amount of data** for our data generation, network training, and network evaluation. Please refer to the ReadMe file in the code submission for more details.

B More Data Details and Visualization

In Table B.1, we summarize our data statistics. In Fig. B.1, we visualize our simulation assets from ShapeNet [1] and PartNet [2] that we use in this work.

There are two kinds of object categories: big heavy objects $\mathscr{C}_{\textbf{heavy}}$ and small item objects $\mathscr{C}_{\textbf{item}}$. In our experiments, $\mathscr{C}_{\textbf{heavy}}$ include cabinets, microwaves, tables, refrigerators, safes, and washing machines, while $\mathscr{C}_{\textbf{item}}$ contains baskets, bottles, bowls, boxes, cans, pots, mugs, trash cans, buckets, dispensers, jars, and kettles. For the *placement* and *fitting* tasks, we use the object categories in $\mathscr{C}_{\textbf{heavy}}$ to serve as the main scene object. And, we use the $\mathscr{C}_{\textbf{item}}$ categories as the scene objects for the other two task environments, as well as employ them as the acting objects for all the four tasks. Some objects may contain articulated parts. For the objects in $\mathscr{C}_{\textbf{heavy}}$, we sample a random starting part pose that is either fully closed or randomly opened to random degree with equal probabilities. For the acting objects, we fix the part articulation at the rest state during the entire simulated interaction.

5th Conference on Robot Learning (CoRL 2021), London, UK.

Train-Cats	All	Basket	Bottle	Bowl	Box	Can	Pot
Train-Data	867	77	16	128	17	65	16
Test-Data	281	43	44	5	18	5	

		Mug	Fridge	Cabinet	Table	Trash	Wash
		134	34	272	70	25	13
		46	9	73	25	10	3

Test-Cats	All	Bucket	Disp	Jar	Kettle	Micro	Safe
Test-Data	637	33	9	528	26	12	29

Table B.1: **Dataset Statistics.** Our experiments use 1,785 ShapeNet [1] models in total, covering 18 commonly seen indoor object categories. We use 12 training categories, which are split into 867 training shapes and 281 test shapes, and 6 test categories with 637 shapes that networks have never seen during training. In the table, Disp, micro, wash, and trash are respectively short for dispenser, microwave, washing machine, and trash can.

Figure B.1: **Data Visualization.** We visualize one example shape in our simulation assets for each of the 18 object categories we use in this paper. Twelve of them are training object categories while the other six are test categories. We mark with underscore the object categories in $\mathscr{C}_{\textbf{heavy}}$ that are assumed to be static (heavy).

C More Details on Settings

For the physical simulation in SAPIEN [3], we use the default setting of frame rate 500 frame-per-second, solver iterations 20, standard gravity 9.81, static friction coefficient 4.0, dynamic friction coefficient 4.0, and restitution coefficient 0.01. The perspective camera is located at a random position determined by a random azimuth $[0°, 360°)$ and a random altitude $[30°, 60°]$, facing towards the center of the scene point cloud with 5 unit length distanced away. It has field-of-view $35°$, near plane 0.1, far plane 100, and resolution 448×448. We use the Three-point lighting with additional ambient lighting. The scene point cloud S samples $n = 10,000$ points using Furthest Point Sampling (FPS) from the back-projected depth scan of the camera.

D More Details on Networks

For the feature extraction backbones $\textbf{Enc}_{\textbf{scene}}$ and $\textbf{Enc}_{\textbf{object}}$, we use the segmentation-version PointNet++ [4] with a hierarchical encoding stage, which gradually decreases point cloud resolution by several set abstraction layers, and a hierarchical decoding stage, which gradually expands back the

resolution until reaching the original point cloud with feature propagation layers. There are skip links between the encoder and decoder. We use the single-scale grouping version of PointNet++. The two PointNet++ networks do not share weights.

More specifically, for $\mathbf{Enc}_{\mathbf{scene}}$, we use four set abstraction layers with resolution 1024, 256, 64 and 16, with learnable Multilayer Perceptrons (MLPs) of sizes [3, 32, 32, 64], [64, 64, 64, 128], [128, 128, 128, 256], and [256, 256, 256, 512] respectively. There are four corresponding feature propagation layers with MLP sizes [131, 128, 128, 128], [320, 256, 128], [384, 256, 256], and [768, 256, 256]. Finally, we use a linear layer that produces a point-wise 128-dim feature map for all scene points. We use ReLU activation functions and Batch-Norm layers.

For $\mathbf{Enc}_{\mathbf{object}}$, we use three set abstraction layers with resolution 512, 128, and 1 (which means that we extract the global feature of the acting object O), with learnable MLPs of sizes [3, 64, 64, 128], [128, 128, 128, 256], and [256, 256, 256, 256] respectively. There are three corresponding feature propagation layers with MLP sizes [131, 128, 128, 128], [384, 256, 256], and [512, 256, 256]. Finally, we use a linear layer that produces a 128-dim feature for every point of the acting object, and use another linear layer to obtain a 128-dim global acting object feature. We use ReLU activation functions and Batch-Norm layers.

We use a PointNet [5] to implement the proposed object-kernel point convolution module $\mathbf{Conv}_{\mathbf{object}}$. The network has three linear layers [256, 128, 128, 128] that transforms each point feature. We use ReLU activation functions and Batch-Norm layers. Finally, we apply a point-wise max-pooling operation to pool over the m acting object points to obtain the aggregated 128-dim feature for every sampled scene seed point p_i.

The final affordance prediction module $\mathbf{Dec}_{\mathbf{critic}}$ is implemented with a simple MLP with two layers [384, 128, 1], with the hidden layer activated by Leaky ReLU and final layer without any activation function. We do not use Batch-Norm for either of the two layers.

E More Details on Training

We collect hundreds of thousands of interaction trials in the simulated task environments for each task. The scene and acting objects are selected randomly from the training data of the training object categories. We equally sample data from different object categories, to address the data imbalance issue. For a generated pair of scene S and acting object O, we then randomly pick an interacting position p_i from the task-specific possible region to perform a simulated interaction. The task environment provides us the final interaction outcome, either successful or failed, using the task-specific metrics described in Sec. 4.2.

Using random data sampling gives us different positive data rate for different tasks, ranging from the lowest one 7.4% for *pushing* and the highest 41.2% for *placement*. We empirically find that having enough positive data samples are essential for a successful training. Thus, for each task, we make sure to sample at least 20,000 successful interaction trials. It is also important to equally sample positive and negative data points in every batch of training.

We use batch size 32 and learning rate 0.001 (decayed by 0.9 every 5000 steps). Each training takes roughly 1-2 days until convergence on a single NVIDIA Titan-XP GPU. The GPU memory cost is about 11 GB. At the test time, the speed is fast (on average 62.5 milliseconds per data) since it only requires a single feed-forwarding inference throughout the network. Testing over a batch of 64 needs 6 GB GPU memory.

F More Results

Fig. F.2 presents more results of our affordance heatmap predictions, to augment Fig. 4 in the main paper.

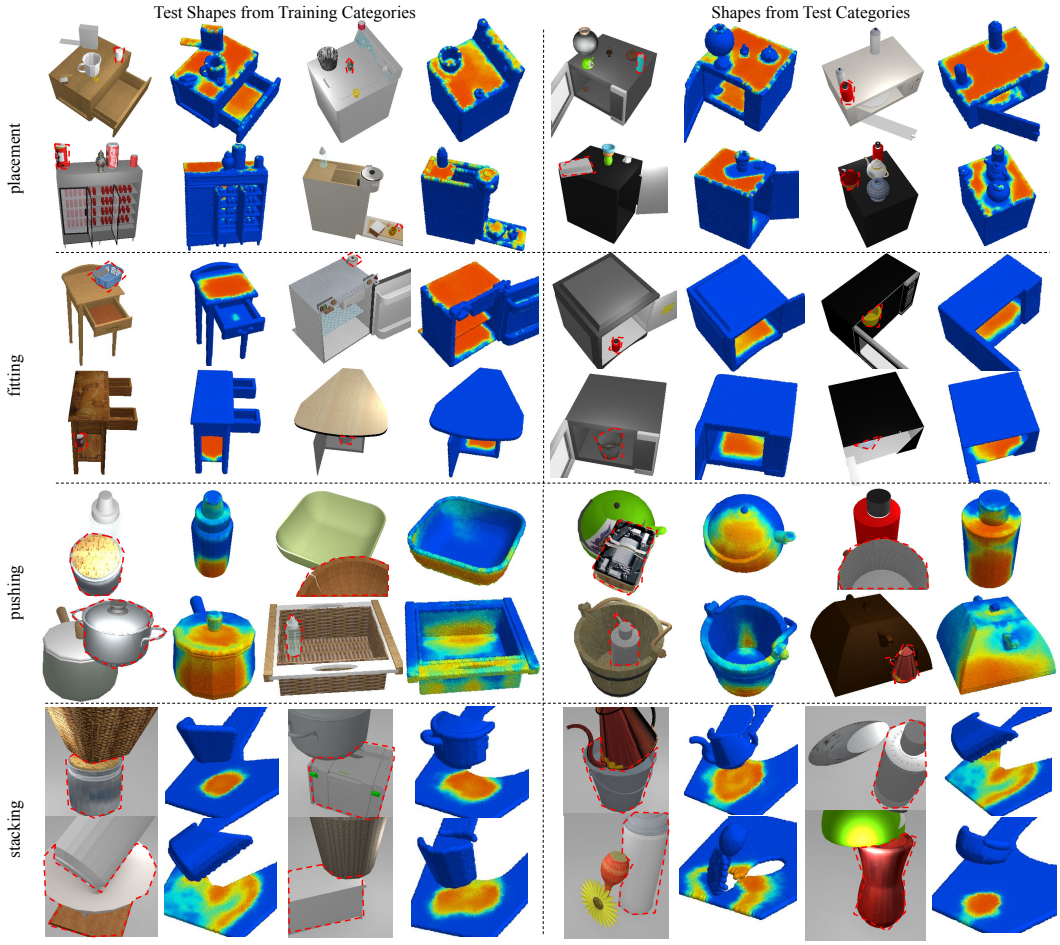

Figure F.2: **More Results.** We present more results of point-wise affordance heatmap predictions, to augment Fig. 4 in the main paper. For each of the tasks, we show examples of our network predictions. Left two examples show test shapes from the training categories, while two right ones are shapes from the test categories. In each pair of result figures, we draw the scene geometry together with the acting object (marked with red dashed boundary) on the left, and show our predicted per-point affordance heatmaps on the right.

G More Results on Real-world Data

Fig. G.3 presents more results testing if our learned model can generalize to real-world data, to augment Fig. 5 in the main paper. We use the Replica dataset [6], the RBO dataset [7], and Google Scanned Objects [8, 9, 10, 11] as the input 3D scans.

For qualitative results over the real-world scans shown in Fig. 5 in the main paper, we use an iPad Pro to collect the real-world scans by ourselves. We first mount the camera on a fixed plane and rotate the iPad to make the camera view 20° top down to the ground. We use the front structured light camera on iPad to capture a cluttered table top and a microwave oven.

Our method is designed to learn visual priors for object-object interaction affordance. Future works may further finetune our priors predictions by training on some real-world tasks to obtain more accurate posterior results.

H Failure Cases: Discussion and Visualization

Fig. H.4 summarizes common failure cases of our method. See the caption for detailed explanations and discussions.

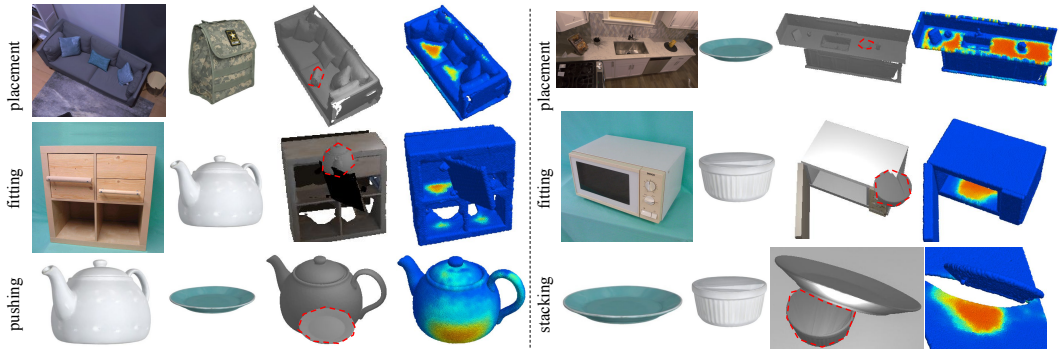

Figure G.3: **More Results on Real-world Data.** From left to right: we respectively show the scene geometry (a partial 3D scan from the camera viewpoint), the acting object (a complete 3D point cloud), the interaction snapshot (to illustrate the poses and sizes of the objects), and our point-wise affordance predictions. For the input 3D scene geometry scans, we use the Replica dataset [6] (the *placement* examples), the RBO dataset [7] (the *fitting* examples), and Google Scanned Objects [9, 11] (the scene geometry for the other two tasks). We use shapes from Google Scanned Objects [8, 9, 10, 11] for the acting objects in the four tasks.

We hope that future works may study improving the performance regarding these matters.

References

[1] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[2] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019.

[3] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020.

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[6] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[7] R. Martín-Martín, C. Eppner, and O. Brock. The rbo dataset of articulated objects and interactions, 2018.

[8] G. Research. Threshold ramekin white porcelain. https://fuel.ignitionrobotics.org/1.0/GoogleResearch/models/Threshold_Ramekin_White_Porcelain, 2020.

[9] G. Research. Room essentials salad plate turquoise. https://fuel.ignitionrobotics.org/1.0/GoogleResearch/models/Room_Essentials_Salad_Plate_Turquoise, 2020.

[10] G. Research. Us army stash lunch bag. https://fuel.ignitionrobotics.org/1.0/GoogleResearch/models/US_Army_Stash_Lunch_Bag, 2020.

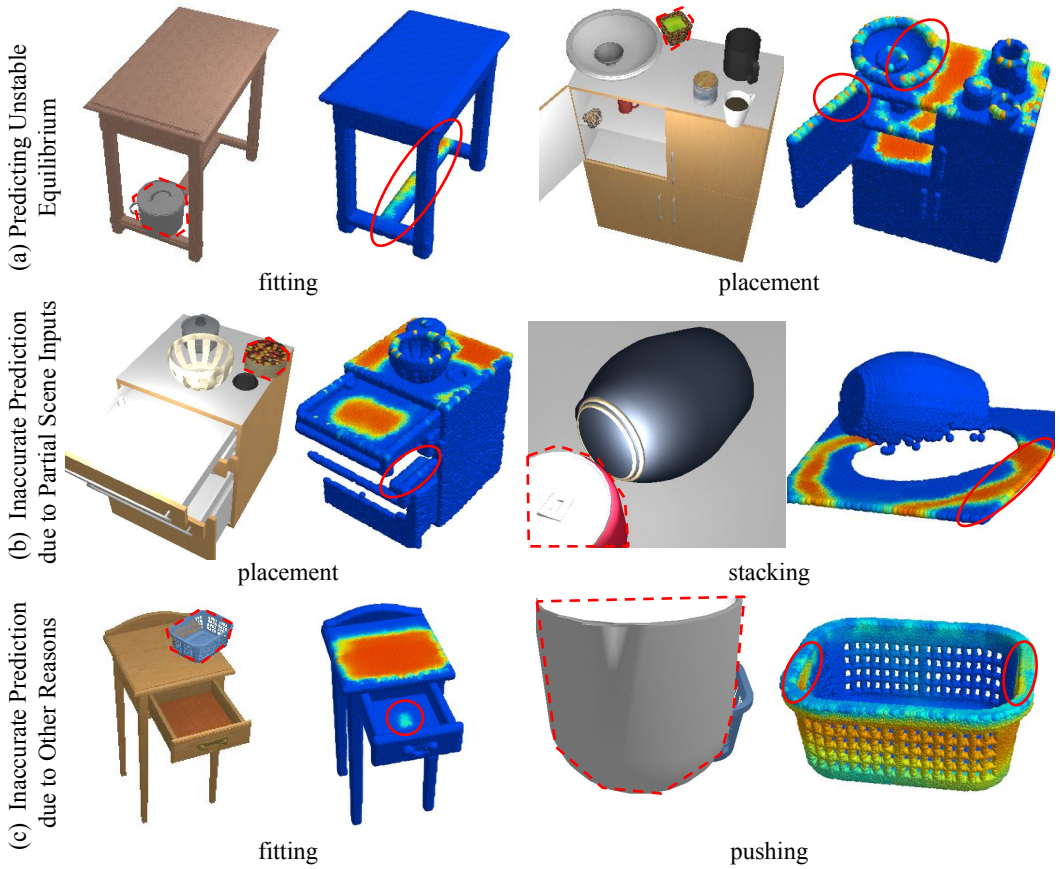

Figure H.4: **Failure Cases.** We visualize common failure cases of our method, by presenting two examples for each of the following categories: predicting unstable equilibrium (a), inaccurate prediction due to partial scene inputs (b), and inaccurate prediction due to other reasons (c). For each pair of results, we show the acting and scene objects on the left, with the acting object marked with red dashed boundary, and our network prediction on the right. On the right image, we explicitly mark out the areas for problematic predictions using red circles. In many real-world scenarios, especially for the *placement* and *fitting* tasks, one might want to find positions for object to be put stably. However, for the examples as shown in (a), we observe some failure cases that unstable equilibrium positions are also predicted, though usually with smaller likelihood scores. For (b), since our network takes as input partial 3D scanned point clouds for the scene geometry, we observe some artifacts in the predictions due to the incompleteness of the input scene point clouds. For instances, one may also put the object in the second drawer, besides the first drawer, in the *placement* example; and in the *stacking* example, one cannot support the jar from the right side. Finally, there are other inaccurate predictions in the examples as shown in (c) due to various reasons: in the *fitting* case, there is an erroneous prediction that the middle of the drawer can fit the basket; and for the *pushing* case, the two side areas should not be pushable. We hope future works can improve upon addressing these issues.

[11] G. Research. Threshold porcelain teapot whit. https://fuel.ignitionrobotics.org/1.0/GoogleResearch/models/Threshold_Porcelain_Teapot_White, 2020.