RVT-2: Learning Precise Manipulation from Few Demonstrations

Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, Dieter Fox
NVIDIA

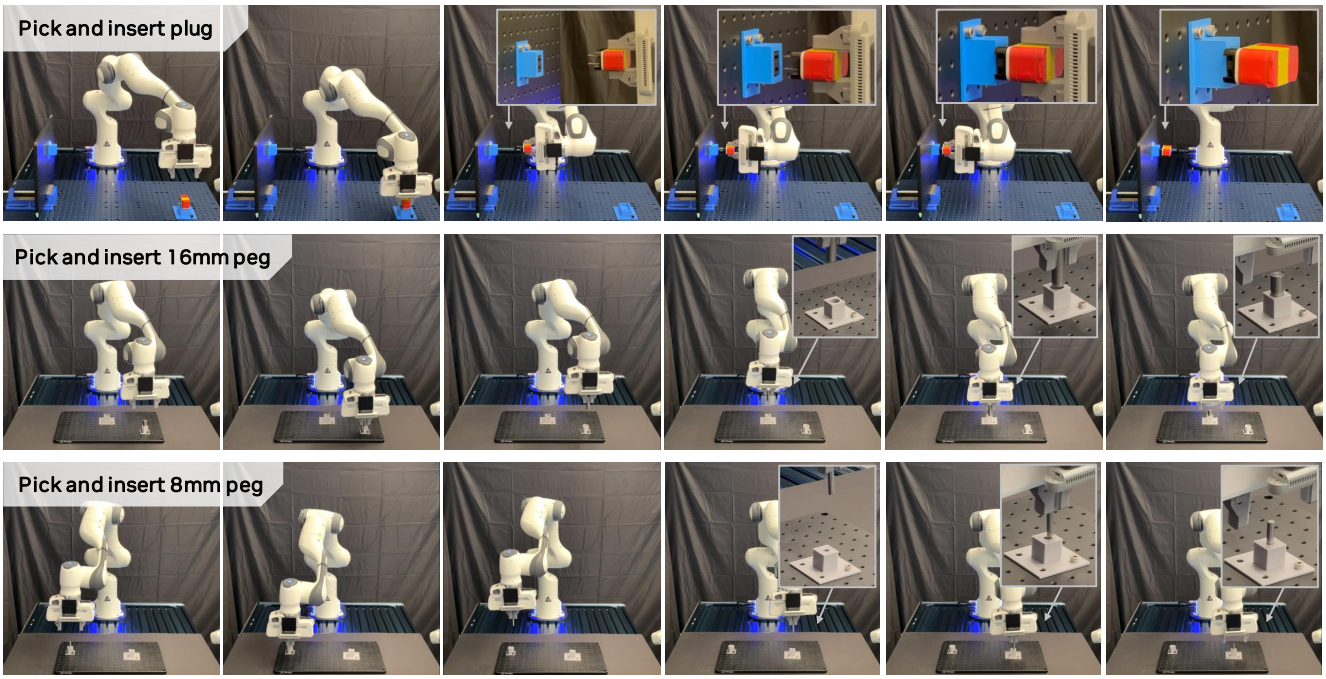

Fig. 1: **RVT-2 performing high precision tasks.** Given a language instruction, a single RVT-2 model can perform multiple 3D manipulation tasks, including ones requiring millimeter-level precision like *inserting peg in hole* and *inserting plug in socket*. RVT-2 is trained with ∼10 demonstrations per task and uses only a single third-person RGB-D camera.

***Abstract*—In this work, we study how to build a robotic system that can solve multiple 3D manipulation tasks given language instructions. To be useful in industrial and household domains, such a system should be capable of learning new tasks with few demonstrations and solving them precisely. Prior works, like PerAct [30] and RVT [12], have studied this problem, however, they often struggle with tasks requiring high precision. We study how to make them more effective, precise, and fast. Using a combination of architectural and system-level improvements, we propose RVT-2, a multitask 3D manipulation model that is 6X faster in training and 2X faster in inference than its predecessor RVT. RVT-2 achieves a new state-of-the-art on RLBench [19], improving the success rate from 65% to 82%. RVT-2 is also effective in the real world, where it can learn tasks requiring high precision, like picking up and inserting plugs, with just 10 demonstrations. Visual results, code, and trained model are provided at: https://robotic-view-transformer-2.github.io/.**

I. Introduction

One of the holy grails of robot learning is building general-purpose robotic systems that can solve multiple tasks and generalize to unseen environment configurations. To be useful, such systems should be capable of precise manipulation and only need a few demonstrations of a new task. For example, in an industrial manufacturing setting, we can expect a person to demonstrate a high-precision task like peg insertion to a robot just a few times, after which the robot should start doing that task independently. Similar examples can be found in other domains like household and retail. In this work, we study the problem of building a manipulation system that can solve various tasks precisely, given just a few demonstrations. The systems should have three key characteristics: (1) handle multiple tasks, (2) require only a few demonstrations, and (3) solve tasks with high precision.

Prior work has made significant progress towards this goal. Starting with works like Transporter Networks [38] and IFOR [11] that studied planar pick-and-place tasks, recent works have gone beyond the 2D plane and studied manipulation in 3D with a few examples [20]. Some notable methods are PerAct [30] and RVT [12]. Given a language instruction, PerAct [30] adopted a multi-task transformer model for 3D manipulation by predicting the next keyframe pose. Even though PerAct achieved impressive performance,

it uses a voxel-based representation for the scene, limiting its scalability. RVT [12] addressed the limitations of PerAct by proposing a novel multi-view representation for encoding the scene. The multi-view representation has various advantages, including faster training speed, faster inference, and better task performance. Compared to PerAct, RVT demonstrated a 36X faster training speed and improved the performance from 48% to 63% on 18 tasks in RLBench [19].

We were motivated by the question of what prevents RVT from achieving even higher performance. Upon careful analysis, we find that RVT struggles with tasks requiring high precision, like *screwing bulb* or *inserting a peg*. During our analysis, we also identified several opportunities to further improve the training and inference speed of the system. Through our architectural and system-level improvements, we were able to boost both the speed and efficacy of RVT. We thereby present RVT-2, which improves RVT on the training speed by 6X (from 2.4M samples per day to 16M samples per day), inference speed by 2X (from 11.6 fps to 20.6 fps), and task success rate by 15 points (from 62.9 to 77.6) on the RLBench benchmark, achieving state-of-the-art results.

We also find that a single RVT-2 model is able to solve multiple tasks in the real world with as few as 10 demonstrations. Specifically, RVT-2 can perform tasks requiring millimeter-level precision, like *inserting a peg in a hole* and *inserting a plug in a socket*, while only using a single third-person camera. To the best of our knowledge, this is the first time a vision-based policy trained with a few examples has been tested to work on such high-precision tasks.

Overall, the gains in RVT-2 were achieved by a combination of architectural and system-level improvements. For the architectural improvement, we introduce three main design innovations. First, we equip RVT-2 with a multi-stage inference pipeline that allows the network to zoom into the region of interest and predict more precise end-effector poses. Further, to save GPU memory during training and to improve speed, we adopt a convex upsampling technique. Lastly, we improve end-effector rotation prediction by utilizing location-conditioned features instead of just the global features as done in RVT.

For the system-level optimizations, we create a custom virtual image renderer to replace the generic renderer used in RVT (PyTorch3D [25]). With this custom accelerated rendering library, we improve the speed and reduce the memory usage of RVT in both training and inference. We also investigate and incorporate cutting-edge practices in training transformer models, including fast, optimized optimizers and mixed-precision training. While each of these changes in itself is not novel and has appeared in some form in prior works, our contribution lies in building a precise 3D manipulation system by incorporating these changes successfully.

To summarize, we push the frontiers of 3D manipulation with few-shot demonstrations. We achieve significant improvements and demonstrate superior real-world performance. We also provide a careful analysis ablating and quantifying the improvements sourced from different factors. Our code is available at https://robotic-view-transformer-2.github.io/.

II. Related Work

Robotic Manipulation in 3D. Compared to manipulation in the 2D top-down setting [38, 11, 29], inferring robots' movements and interactions in full 3D space is much more challenging due to the higher degrees of freedom in the action space and the complexity of 3D spatial reasoning [9]. To tackle manipulation in the 3D space, recent works have leveraged various perceptual representations. Camera images have been widely used for vision-based manipulation, e.g., in models such as RT-1 [1], RT-2 [2], and ALOHA [39]. For more effective 3D spatial reasoning, depth information is commonly required, where RGB-D images are assumed as input to the manipulation policy [32]. PolarNet [3] and M2T2 [37] directly use the point cloud reconstructed from RGB-D images and process it with an encoder plus a transformer to predict actions. C2F-ARM [20], PerAct [30], and FourTran [17] voxelize the point clouds and use a 3D convolutional network as the backbone for action inference. Act3D [8] and ChainedDiffuser [35] represent the scene as a multi-scale 3D feature cloud. To boost both the time efficiency and task efficacy, RVT [12] proposes to use multi-view virtual images as the scene representation. Nonetheless, most of these prior models are only applied to real-world tasks that do not require high precision actions. We hereby aim to leverage these advances and push the boundary further on high precision manipulation problems. Inspired by prior work that uses a multi-stage "coarse-to-fine" inference strategy [20, 8], our model selects a task-critical part of the scene to "zoom into" and examine in a finer resolution through virtual images.

High Precision Manipulation. High-precision manipulation is required for tasks that have low motion error tolerance such as those in industrial settings. To learn high-precision manipulation policies, previous works have relied on various sensory modalities and data-expensive learning algorithms. As an earlier work, proprioception sensory data is used to learn a peg-in-hole policy via imitation learning [13]. By using camera images, Schoettler et al. [27] presents a residual reinforcement learning algorithm to accomplish industrial insertion tasks from visual sensory inputs. Tang et al. [33] propose to detect the peg location from the initial camera frame and apply reinforcement learning to learn a final-inch insertion policy from proprioception data. To further improve the execution accuracy, touch feedback such as force-torque sensors [22, 24] and vision-based tactile sensors [6, 36] are exploited. However, these works leverage algorithms requiring expensive training data (e.g., either reinforcement learning or imitation learning from hundreds of demonstrations) and still only learn a single model per task. In contrast, our RVT-2 is able to learn a multi-task high-precision manipulation policy from much fewer demonstrations per task. ACT [39] is another method that aims to learn precise manipulation from a few demonstrations. However, there are significant differences between ACT and RVT-2. Given language input, RVT-2 can solve different variations of a task while ACT does not take language as input and can only be trained with one variation

of a task at a time. RVT-2 makes key-point based predictions while ACT makes continuous joint state predictions. RVT-2 takes point cloud as input while ACT works with multiview images.

Virtual Views for 3D Vision. The use of virtual views provides a strategic lever for exploiting well-established image-based neural network architectures, such as convolutional neural networks and transformer models, for processing 3D scene information. Prior works have shown the benefit of virtual view rendering over sophisticated point-based methods in various vision tasks, from object recognition [31, 10, 15, 16], object detection [4], to 3D visual grounding [18]. The application of virtual views in the field of robotics has been less explored. Recently, RVT [12] leverages multi-view representation for predicting robot actions for object manipulation. Our work builds upon RVT using a series of architectural and system-level improvements to make it more performant and efficient.

III. Method

Our method, RVT-2, allows for three-dimensional and precise manipulation. A single RVT-2 model is trained to solve multiple tasks from language instructions and requires only a few demonstrations per task. RVT-2 builds upon RVT [12], a state-of-the-art model for 3D object manipulation. Similar to RVT, RVT-2 is based on the paradigm of key-frame based manipulation. But it achieves better task performance, precision, and speed through a series of improvements. We group these improvements into two categories: architectural for the ones related to changes in the neural network, and system-related for those related to software optimizations.

A. Background

Key-frame based manipulation. In key-frame based manipulation, the robot trajectory is described using a sequence of key (or bottleneck) poses. For example, a trajectory for drawer opening could be described by a sequence of key poses like *pre-grasp for drawer handle*, *grasp pose*, and *pull-pose for the drawer*. These key poses are provided in the training dataset, and the aim of a key-frame based method is to learn to predict these poses. Specifically, methods like PerAct [30] and RVT [12] take as input the language goal along with the current scene point cloud and predict the next key-frame pose. The predicted pose is then passed to a motion planner, which generates a trajectory towards it[1]. When the robot reaches the predicted pose, the method takes in a new scene point cloud and predicts the subsequent key-frame pose. This process iterates until the task is successful or a predefined number of steps is reached.

To train a key-frame based behavioral cloning agent, we assume access to a dataset of samples. Each sample includes a language goal, current visual observation, and the next key-frame pose. We can extract such a dataset automatically from dense robot trajectory datasets by defining rules that specify the key-frame poses. For instance, when the state of the gripper changes between open and close, the pose is a key-frame pose. We use the same key-frame extraction scheme as PerAct and encourage readers to refer to Shridhar et al. [30] for details.

Robotic View Transformer (RVT). To predict the key-frame pose, RVT first reconstructs a point cloud of the scene using the input RGB-D images. The scene is then rendered from virtual cameras along orthogonal directions. RVT renders five virtual views, including the top, front, left, back, and right view. RVT shows that using these fixed virtual views around the robot, instead of the original input camera views, results in more effective performance.

These virtual images are then passed to a multi-view transformer model that jointly reasons over all the views. The transformer model predicts a heatmap for each of the views. The heatmap score across views is then back-projected into 3D, where each 3D point receives a score that is the average of the score received by its 2D projections. The 3D point with the largest heatmap score represents the predicted gripper location. Along with the heatmaps, RVT extracts a global feature concatenated from across the views to predict the gripper rotation and state (open or close). We encourage readers to refer to [12] for a detailed overview of RVT.

B. Architectural Changes: RVT $\rightarrow$ RVT-2

Multi-stage Design. RVT predicts the gripper pose using a fixed set of views around the robot. These fixed views might not be sufficient in tasks when the object of interest is very small, and the gripper pose needs to be very precise, like *inserting a peg in a hole*. Hence, RVT-2 adopts a multi-stage design (see Fig. 2), where, in the first or coarse stage, it predicts the area of interest using a fixed set of views. RVT-2 then zooms in the area of interest and re-renders images around it. We use a zoom-in factor of 4, meaning that zoomed-in cameras cover a region of size $1/4^{th}$ the coarse cameras. It uses these close-up images to make precise gripper pose predictions. Such adaptive rendering is possible due to the flexibility provided by the virtual rendering proposed in RVT.

Convex Upsampling. RVT is based on ViT (Vision Transformer) [7], which divides an image into $t_1 \times t_2$ patches. Each image patch is processed as a single token of dimension d. To predict a heatmap from the image tokens, RVT first arranges the image tokens at their corresponding patch location. This results in a 3D feature of shape $t_1 \times t_2 \times d$. RVT then uses transposed convolutions to upsample these features to the image resolution of $h \times w$, creating a feature of shape $h \times w \times d$. Finally, these features of shape $h \times w \times d$ are used to predict the heatmap. This sequence of operations is effective, however, it is memory intensive due to the large intermediate feature of shape $h \times w \times d$.

To address this, RVT-2 removes the feature upsampling and directly predicts heatmap of shape $h \times w$ from features at the token resolution. Specifically, it uses convex upsampling layer proposed by [34]. The convex upsampling layer uses a learned convex combination of features in the coarse grid to make predictions in the higher resolution. [34] shows how it leads

[1]Along with the pose, these methods also output whether or not the motion planner should avoid collision.

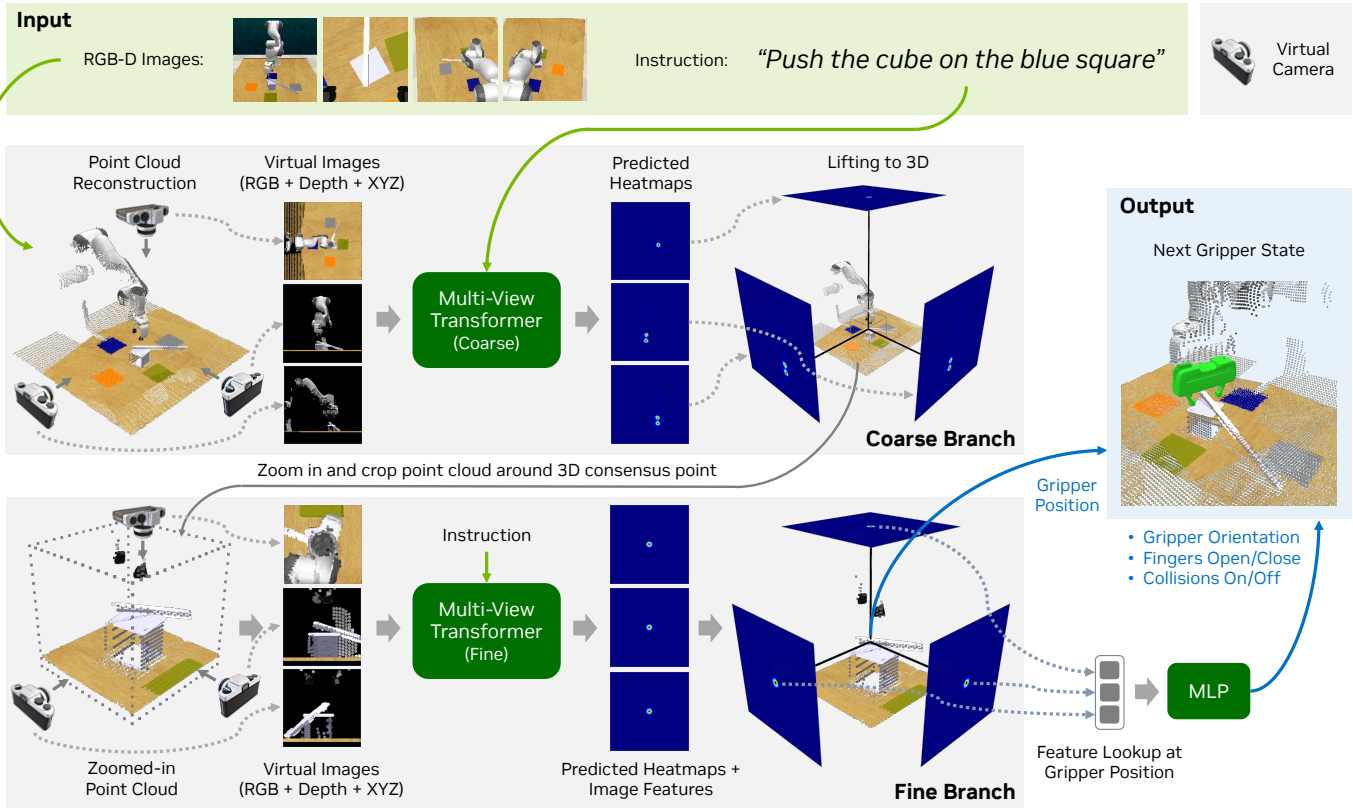

Fig. 2: **RVT-2 Architecture.** Given the current scene and a task instruction, RVT-2 predicts the next key-frame pose. It consists of two stages. The first stage uses fixed virtual views around the robot to predict the area of interest. The second stage uses zoomed-in views from the area of interest to predict the gripper pose.

to sharper predictions at higher resolution. We empirically find that convex upsampling saves memory without sacrificing performance (see Tab. III). The convex upsampling layer does not require any special implementation and can be represented using the native `fold` function in PyTorch.

Parameter Rationalization. We find that network parameters in RVT, like the virtual image size (220) and patch size (11) may not be optimal for GPU as they are not divisible by powers of 2 like 16[2]. RVT-2 rationalizes these parameters to make the neural network more GPU-friendly, improving its speed. RVT-2 adopts parameters similar to ViT [7], i.e. image size of 224 and patch size of 14. Apart from being more GPU-friendly, these parameters reduce the total number of tokens inside the multi-view transformer which is equal to $(image\_size/patch\_size)^2$, boosting the speed further. These choices make RVT-2 faster during training and testing without affecting performance (see Tab. III).

Location Conditioned Rotation. RVT and PerAct use global visual features, like max-pooling over the entire image, to make predictions for end-effector rotation. This can be problematic when there are multiple valid end-effector locations, and the end-effector rotation depends on the location. For example, consider the task of stacking blocks where the scene has two similar blocks but in different orientations. Here, picking either of the two blocks is a valid step. However, since the blocks have different orientations, the end-effector rotation would depend on the chosen end-effector location. Since RVT only uses global visual features to predict rotation, it cannot handle such cases. To address this, RVT-2 uses local features pooled from the feature map at the end-effector location for rotation prediction. This allows RVT-2 to make location-dependent rotation prediction.

Fewer Virtual Views. RVT renders the scene point cloud with five virtual cameras placed in orthogonal locations i.e. back, front, top, left, and right. This choice was based on their observation that fewer camera views reduced performance. However, in our multi-stage RVT-2 model, we find that using only three views, i.e., front, top, and right, suffices and does not sacrifice performance. This is likely because RVT-2 uses zoomed-in views for the final prediction. Fewer virtual views reduce the number of images to be rendered by the renderer and the number of tokens to be processed by the multi-view transformer. Thus, this improves training and inference speed.

C. System-Related Changes: RVT → RVT-2

Point-Renderer. RVT uses PyTorch3D [25] to render virtual RGB-D images. PyTorch3D is an appealing choice because

[2]Exact power of 2 depends on the data-type and NVIDIA GPU architecture. More details can be found at https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html.

of its easy-to-use interface. However, it is a fully-featured differentiable renderer that incurs significant time and memory overhead for point-cloud rendering. To avoid this, we implement a custom projection-based point-cloud renderer in CUDA. Our renderer performs 3 steps to render a point cloud with N points to an RGB image and depth image of size (h, w):

a) Projection: For each 3D point of index $n \in \{0, 1 \dots N\}$ and RGB value f_n, it computes the depth d_n and image pixel coordinate (x_n, y_n) using camera intrinsics and extrinsics. From the 2D pixel coordinate (x_n, y_n), it computes the linear pixel index $i_n = x_n \cdot w + y_n$. The projection operation is easily accelerated using GPU matrix multiplications.

b) Z-ordering: For each pixel of linear-index j in the image, it finds the point index with smallest depth d_n among the set of points that project to the pixel $\{n \mid i_n = j\}$. It assigns that point's RGB value f_n to pixel j of the RGB image and depth d_n to pixel j of the depth image.

To accelerate Z-ordering, we pack each point's depth and index into a single 64-bit integer, such that the most significant 32 bits encode depth, while the least significant bits encode the point index. Then, Z-ordering can be implemented with two CUDA kernels. First, a parallel loop over point cloud points, tries to store each packed depth-index into a depth-index image at the pixel j using the *atomicMin* operation. Only the depth-index stored by the minimum-depth point at each pixel survives. The second kernel, in a loop over pixels, creates depth and feature images by unpacking the depth-index, and looking up the point feature. This trick was proposed by Schütz et al. [28] for rendering color point-clouds by packing the 32-bit color. We extend this to images with arbitrary number of channels, by packing the point index instead.

c) Screen-space splatting: The first two steps are sufficient to produce rendered images. However, the points are treated as infinitesimal light sources, which creates noise in areas where the screen-space point cloud resolution is not higher than the image resolution. A common way to counter this is 3D splatting, whereby each point is modelled by some geometry of a finite size. We represent each point as a disc of radius r facing the camera. This splatting can be computed in screen space after projection and z-ordering, thereby reducing the computation required in the projection and z-ordering. For each pixel j in the image, search in a neighbourhood for another pixel k of lowest depth. If the pixel k has depth $d_k < d_j$, and is closer than $r \cdot focal\_length/d_k$, replace the feature and depth of pixel j with that of pixel k.

Improved training pipeline. We optimize RVT's training pipeline by adopting the latest developments in training transformers. We analyze various techniques and adopt the ones that improve speed without affecting performance. Specifically, we use mixed precision training, 8-bit LAMB optimizer [5], and fast GPU implementation of the attention layer based on xFormers [23].

IV. Experiments

We evaluate RVT-2 by conducting comprehensive experiments in both the simulation and the real-world.

A. Simulation

Dataset and Setup. We conduct the experiments on a standard multi-task manipulation benchmark developed in RLBench [19] and adopted by previous works [30, 8, 12]. The benchmark contains 18 tasks, including non-prehensile tasks like *push buttons*, common pick-and-place tasks like *place wine*, and peg-in-hole tasks that require high precision like *insert peg*. Each task is specified by a language description and consists of 2 to 60 variations such as handling objects in different colors or locations. A Franka Panda robot with a parallel jaw gripper is commanded to complete the tasks. The task and the robot are simulated via CoppelaSim [26]. The input RGB-D images are of resolution 128×128 and are captured by four noiseless cameras mounted at the front, left shoulder, right shoulder, and wrist of the robot. We train and test RVT-2 with the same dataset as PerAct and RVT, with 100 demonstrations per task for training and 25 unseen demonstrations for testing[3]

Training and Evaluation Details. We train RVT-2 with similar computing resources as RVT and PerAct. Specifically, we use a node with 8 NVIDIA V100 16 GB GPUs. Like RVT and PerAct, we use translation augmentation of 12.5 cm along the x, y, and z axis, as well as rotation augmentation of 45° along the z axis. We train RVT-2 for $\sim$80K steps with a cosine learning rate decay schedule and an initial warmup of 2000 steps. The batch size is 192 (24×8) and the learning rate is 2.4×10^3. We use the final model for evaluation. Since RLBench uses a sampling-based motion planner, we evaluate each model four times on each task and report the mean and variance. We measure the inference speed of RVT-2, RVT, and PerAct on an NVIDIA RTX 3090 GPU.

Baselines. We compare RVT-2 with various baselines. These include simple image-to-action behavioral cloning baselines, *Image-BC (CNN)* [21, 30] and *Image-BC (ViT)* [21, 30], that use a CNN and ViT backbone respectively. We also compare with models that have been specifically designed for 3D object manipulation including *C2F-ARM-BC* [19], *PerAct* [30] and *HiveFormer* [14]; as well as more recently proposed methods like *RVT* [12], *PolarNet* [3] and *Act3D* [8]. All baselines and RVT-2 are trained and tested with input images of 128×128, while Act3D uses images of size 256×256.

Training time vs. Performance. In Fig. 3, we compare the training time and success rate on RLBench for RVT-2, RVT, and PerAct. We find that RVT-2 significantly outperforms both while requiring much less compute to train. Because of the efficiency gains, with the same compute, RVT-2 trains 6X faster[4] than RVT, while improving performance by 19% in absolute or 29% in relative terms. While comparing with PerAct, RVT-2 improves the relative performance by 65%.

[3]for the close jar task, we use the success criteria fixed by Tsung-Wei Ke here: https://github.com/buttomnutstoast/RLBench/commit/587a6a0e6dc8cd36612a208724eb275fe8cb4470. The fix is used in Act3D. This fix did not affect the performance of the released RVT [12].

[4]RVT can fit a batch size of 24 and train for 100K steps, equivalent to training over 2.4M samples in 24 hours. RVT-2 fits a batch size of 192 and trains for 83.3k steps, equivalent to 16M samples in 20 hours.

Models	Avg. Success ↑	Avg. Rank ↓	Train time (in days) ↓	Inf. Speed (in fps) ↑	Close Jar	Drag Stick	Insert Peg	Meat off Grill	Open Drawer	Place Cups	Place Wine
Image-BC (CNN) [21, 30]	1.3	7.4	-	-	0	0	0	0	4	0	0
Image-BC (ViT) [21, 30]	1.3	7.7	-	-	0	0	0	0	0	0	0
C2F-ARM-BC [20, 30]	20.1	5.8	-	-	24	24	4	20	20	0	8
HiveFormer [14]	45.3	5.2	-	-	52.0	76.0	0.0	**100.0**	52.0	0.0	80
PolarNet [3]	46.4	4.8	-	-	36.0	92.0	4.0	**100.0**	84.0	0.0	40
PerAct [30]	49.4	4.4	16.0	4.9	55.2 ± 4.7	89.6 ± 4.1	5.6 ± 4.1	70.4 ± 2.0	88.0 ± 5.7	2.4 ± 3.2	44.8 ± 7.8
Act3D [8]	65.0	2.8	5.0		92.0	92.0	27.0	94.0	**93.0**	3.0	80
RVT [12]	62.9	2.8	1.0	11.6	52.0 ± 2.5	**99.2** ± 1.6	11.2 ± 3.0	88.0 ± 2.5	71.2 ± 6.9	4.0 ± 2.5	91.0 ± 5.2
RVT-2 (ours)	**81.4**	**1.5**	**0.83**	**20.6**	**100.0** ± 0.0	99.0 ± 1.7	**40.0** ± 0.0	99.0 ± 1.7	74.0 ± 11.8	**38.0** ± 4.5	**95.0** ± 3.3

Models	Push Buttons	Put in Cupboard	Put in Drawer	Put in Safe	Screw Bulb	Slide Block	Sort Shape	Stack Blocks	Stack Cups	Sweep to Dustpan	Turn Tap
Image-BC (CNN) [21, 30]	0	0	8	4	0	0	0	0	0	0	8
Image-BC (ViT) [21, 30]	0	0	0	0	0	0	0	0	0	0	16
C2F-ARM-BC [20, 30]	72	0	4	12	8	16	8	0	0	0	68
HiveFormer [14]	84	32.0	68.0	76.0	8.0	64.0	8.0	8.0	0.0	28.0	80
PolarNet [3]	96	12.0	32.0	84.0	44.0	56.0	12.0	4.0	8.0	52.0	80
PerAct [30]	92.8 ± 3.0	28.0 ± 4.4	51.2 ± 4.7	84.0 ± 3.6	17.6 ± 2.0	74.0 ± 13.0	16.8 ± 4.7	26.4 ± 3.2	2.4 ± 2.0	52.0 ± 0.0	88.0 ± 4.4
Act3D [8]	99	51.0	90.0	95.0	47.0	**93.0**	8.0	12.0	9.0	92.0	94
RVT [12]	**100.0** ± 0.0	49.6 ± 3.2	88.0 ± 5.7	91.2 ± 3.0	48.0 ± 5.7	81.6 ± 5.4	**36.0** ± 2.5	28.8 ± 3.9	26.4 ± 8.2	72.0 ± 0.0	93.6 ± 4.1
RVT-2 (ours)	**100.0** ± 0.0	**66.0** ± 4.5	**96.0** ± 0.0	**96.0** ± 2.8	**88.0** ± 4.9	92.0 ± 2.8	35.0 ± 7.1	**80.0** ± 2.8	**69.0** ± 5.9	**100.0** ± 0.0	**99.0** ± 1.7

TABLE I: **Multi-Task Performance on RLBench.** We report the success rate for 18 RLBench [19] tasks and the average success rate across all the tasks. The success condition is as defined in RLBench. RVT-2 outperforms all methods while having higher training and inference speed. Performance of HiveFormer and PolarNet are reported by [3]; RVT and PerAct are reported by [12]; and Act3D is reported by [8]. All are trained with 100 demonstrations and a single model is evaluated on all the tasks. All methods use input images of resolution 128×128, except Act3D, which uses 256×256.

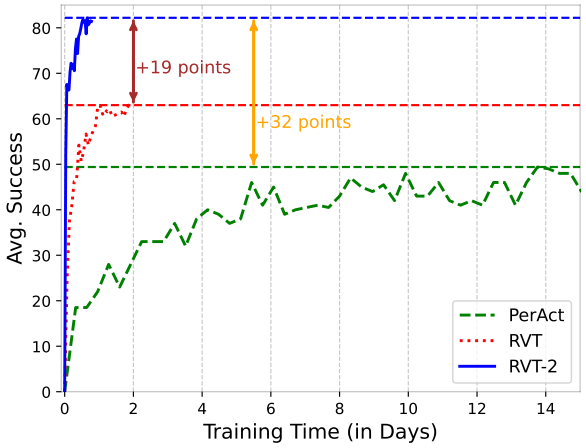

Fig. 3: **Training time vs Success rate on RLBench.** All models are trained on 8 NVIDIA V100 GPUs. RVT-2 trains significantly faster and achieves higher performance than prior state-of-the-art RVT and PerAct.

We find that within 2 hours of training, RVT-2 outperforms RVT trained for 24 hours and PerAct trained for 16 days. These efficiency gains could allow for further scaling up RVT-2 in the future. In inference speed, RVT-2 exhibits around 2X improvement compared to RVT. With an inference speed of 20 fps, RVT-2 opens up new possibilities for real-time reactive control.

Mulit-Task Performance. Table I summarizes the comparison of RVT-2 with prior methods on the RLBench tasks. Among all the methods, RVT-2 achieves the highest average success rate of 81.4%. RVT-2 outperforms the prior best-performing model Act3D by 16.4% absolute or 25% relative improvement while requiring 6X less compute to train (5 days vs. within 20 hours). From Fig. 3, we see that RVT-2 achieves higher performance than Act3D with just 4 hours of training. Overall, RVT-2 achieves the best results in 13 out of 18 tasks and an average rank of 1.5.

Out of the 18 tasks, the task where RVT-2 does not achieve close to the best results is *open drawer*. Upon further investigation, we find that on *open drawer*, RVT-2 achieves higher success rates like 86% on earlier checkpoints than the final one. The lower performance on the final checkpoint could be an artifact of over-fitting or multi-task training where performance on some tasks degrades while improving on others.

High-Precision Tasks. We find that RVT-2 outperforms other methods on high-precision tasks like *insert peg*, *stack cups* and *screw bulb*. In *insert peg*, the robot must pick up a square peg on the tabletop and insert it onto a specific cuboid stick. This task can effectively examine the precision of the learned model since the clearance between the stick and the peg is very tight, and the robot has to align the square peg perfectly with the cuboid stick; otherwise, any tiny error will result in a failure insertion. In *stack cups*, a minor error in the pick and place locations of the cup results in failure as seen in the low success rate of prior methods. Similarly, in *screw bulb*, the bulb's base must be well aligned with the socket for successful screwing. Our experiments show that RVT-2 achieves a significantly higher success rate on these tasks, achieving 88% versus 48% for the previous best on *scew bulb*; 69% versus 26.4% on *stack cups* and 40% versus 27% on *insert peg*.

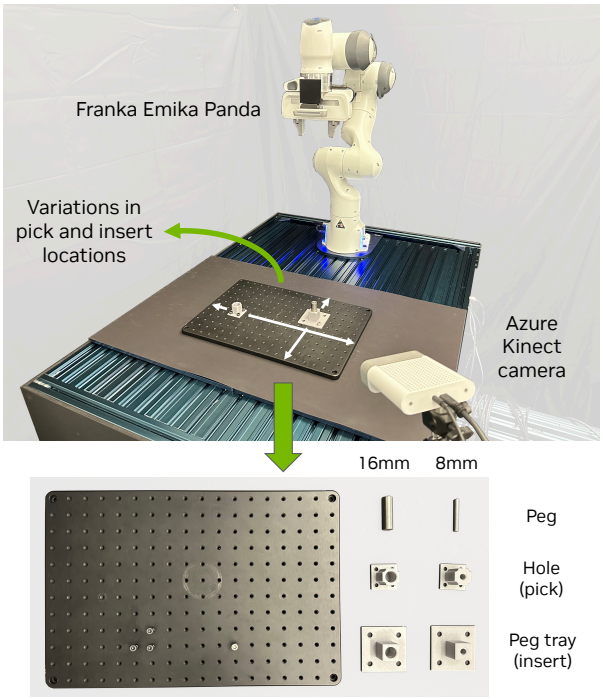

Fig. 4: Our real-world setup (top) and the peg picking and insertion task from [33].

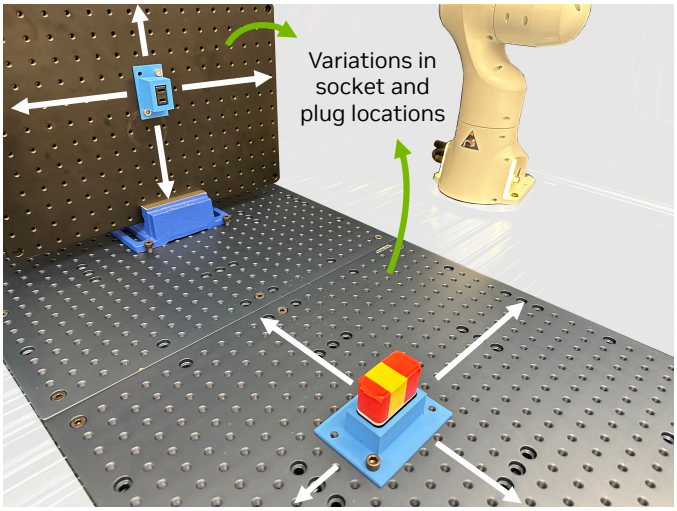

Fig. 5: The 2-prong plug picking and insertion task from [33]; and the considered variations in object locations.

B. Real World

Dataset and Setup. We compare RVT-2 with RVT on a real-world manipulation setup similar to that used in RVT (Fig. 4 top). The setup consists of a statically mounted Franka Emika Panda arm and a static third-person view Azure Kinect RGB-D camera. Instead of following the camera position in RVT, we move the camera closer to the robot's workspace to ensure the point cloud's quality for high-precision manipulation. We use this camera position for all tasks. Besides the same five tasks used in RVT (*stack blocks*, *press sanitizer*, *put marker in mug/bowl*, *put object in drawer*, *put object in shelf*), we additionally evaluate on three high-precision tasks from IndustRealKit [33]: *pick and insert 16mm peg*, *pick and insert 8mm peg*, *pick and insert plug*. The two peg tasks consist of picking up the peg from a hole and inserting it into another hole (Fig. 4). Both the pick and place locations are randomized over the work surface during evaluation. The plug task consists of picking up a 2-prong plug from a tray and inserting it into a vertically mounted socket (Fig. 5). This task further goes beyond 2D pick-and-place and requires precise manipulation in the 3D space. Similarly, the location of the plug tray and socket are randomized over their work surface during evaluation. For tasks in RVT [12], we collect the same number of demonstrations (~ 10) as reported by them. For new high-precision tasks, we collect 10 demonstrations per task. The dataset statistics are provided in Tab. II.

Task	# of vari.	# of train	# of test	*Models* RVT	RVT-2 (ours)
Stack blocks	3	15	10	**80%**	**80%**
Press sanitizer	1	7	10	**90%**	80%
Put marker in mug/bowl	4	12	10	20%	**50%**
Put object in drawer	3	12	10	30%	**50%**
Put object in shelf	2	8	10	**100%**	**100%**
All tasks in RVT [12]	13	54	50	64%	**72%**
Pick and insert 16mm peg	1	10	10	50%	**60%**
Pick and insert 8mm peg	1	10	10	40%	**50%**
Pick and insert plug	1	10	10	10%	**50%**
All high precision tasks	3	30	30	33.3%	**53.3%**
All tasks	16	84	80	52.5%	**65%**

TABLE II: **Results in the real world.** Both RVT-2 and RVT use a single model for all 8 tasks with 16 variations. RVT-2 outperforms RVT on the tasks from [12] and the new high-precision tasks.

Training and Evaluation Details. We train both RVT and RVT-2 on the same dataset for fairness. We train a single RVT and RVT-2 model for all eight tasks. Both models are trained for 10 epochs using a cosine learning rate schedule and the same data augmentation as in our simulation experiments. For RVT-2 we use the same batch size and learning rate as in the simulation experiments. For RVT, we cannot fit a larger batch size as RVT-2 in memory, so we use the official training parameters of batch size 24 and learning rate 2.4×10^3 [12]. We use the final model for evaluation.

Experiment Results. Table II shows the results in the real world. We find that RVT-2 can perform multiple tasks with only a handful of demonstrations (~ 10) per task. Of the five tasks from RVT [12], RVT-2 outperforms RVT by 8 absolute points and 12.5 in relative terms. On all three new tasks that

Row ID	Multi-Stage	Parameter Rational.	Loc. Cond. Rot	Point Render	Convex Upsamp.	Mixed Prec.	8-bit Opt. + Fast Attn.	# of Views	Training Time (in hours)	Training Time % of base	Avg. Succ.	Avg. Succ. diff. wrt. base
1	✓	✓	✓	✓	✓	✓	✓	3	19.5	100%	81.4	0
2	✗	✓	✓	✓	✓	✓	✓	3	13.0	67%	63.9	- 17.5
3	✓	✗	✓	✓	✓	✓	✓	3	26.7	137%	77.2	- 4.2
4	✓	✓	✗	✓	✓	✓	✓	3	19.3	99%	78.9	-2.5
5	✓	✓	✓	✗	✓	✓	✓	3	71.1	366%	79.3	-2.1
6	✓	✓	✓	✓	✗	✗	✓	3	79.2	406%	82.0	+0.6
7	✓	✓	✓	✓	✓	✗	✓	3	58.5	300%	81.2	-0.2
8	✓	✓	✓	✓	✓	✗	✗	3	62.4	320%	81.3	-0.1
9	✓	✓	✓	✓	✓	✓	✓	5	40.3	207%	79.7	-1.7

TABLE III: **Ablations on RLBench.** We quantify the impact of all the architectural and system-level improvements in RVT-2. All these contribute to increasing the training speed, inference speed and performance of RVT-2.

require high precision, RVT-2 constantly outperforms RVT and achieves 53.3% average success rate versus 33.3% for RVT. Although RVT-2 achieves encouraging results on the high-precision tasks with just a single camera, a common reason for failure was small errors during insertion. We believe augmenting RVT-2 with a reactive policy to make fine adjustments in the final stages of insertion could be an exciting future direction. We encourage readers to view video results provided on the project website for success and failure examples.

C. Ablations

We conduct an extensive ablation study in simulation to analyze the effect of each component of RVT-2. Results are shown in Table III.

a) Multi-Stage Design: Comparing row 1 and 2, we can see that including multi-stage design introduces a slowdown in the training time due to the extra stage of rendering and inference. However, it brings a 17.5% success rate improvement since the zoom-in view provides more task-relevant details about the region of interest.

b) Parameter Rationalisation: Comparing row 1 and 3, we see employing GPU-friendly network parameters accelerates the training process without compromising the performance of the network.

c) Location Conditioned Rotation: From row 1 and 4, we see that predicting the rotation using the local features improves performance by 4.2%.

d) Fewer Virtual Views: We vary the number of views in row 1 and 9. It reveals that reducing the number of camera views from 5 to 3 not only maintains task performance but also leads to a twofold increase in training speed. Unlike RVT, the multi-stage network in RVT-2 performs well even with fewer virtual views per stage.

e) Convex Upsampling: On comparing row 6 and 7, we find that removing convex upsampling increases the training time by 20.7 hours. We observe that removing convex upsampling but keeping the mixed precision leads to undefined gradients during training. Hence, to ablate convex upsampling, we compare row 6 and 7, both without mixed precision.

f) Point-Renderer: By replacing PyTorch3D with our customized Point-Rederer, the forward pass is significantly sped up, resulting in 3.6X faster training as seen in row 5.

g) Improved Training Pipeline: Comparing rows 1 and 7, we see that removing automatic mixed precision training leads to a 300% increase in training time. Further removing the 8-bit LAMB optimization and fast attention (row 8) further increases the training time by 20%. Although the improvements due to 8-bit LAMB optimizer and fast attention are not large, they are helpful for the most optimized pipeline and could potentially be beneficial for scaling up RVT-2.

V. Conclusions and Limitations

In this work, we proposed RVT-2, a fast and precise model for 3D object manipulation. It is built on the prior state-of-the-art RVT. Using a combination of architectural and system-level improvements, we significantly improved the speed, precision, and task performance. Although none of the techniques we used is novel in itself, our contribution lies in combining them effectively to advance the state-of-the-art in few-shot 3D manipulation. We found that RVT-2 significantly outperforms prior methods on RLBench while requiring much less compute. In the real world, we found that RVT-2 can solve high-precision tasks that involve inserting pegs and plugs using a single third-person camera and with just 10 demonstrations.

We identify various limitations of RVT-2 which could be avenues of future work. RVT-2, like RVT and PerAct, works with object instances that it was trained on. Extending this to unseen object instances would be an exciting direction. Although on high precision tasks, RVT-2 achieves surprising success with just a single RGB-D sensor, it sometimes fails due to minor insertion position errors. Augmenting RVT-2 to use force information to adjust fine-grained motions could be very interesting. As seen with the *open drawer* task for RVT-2, multi-task optimization could worsen performance on some tasks as training progresses. Developing a strategy to prevent this would be very useful. Lastly, although RVT-2 improves the overall performance on multi-task 3D manipulation by 17 points, the task is still far from being solved with RVT-2 achieving a success rate of 82% in simulation and 72% in the real world.

References

[1] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas

Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[2] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *CoRL*, 2023.

[3] Shizhe Chen, Ricardo Garcia-Pinel, Cordelia Schmid, and Ivan Laptev. PolarNet: 3D point clouds for language-guided robotic manipulation. In *CoRL*, 2023.

[4] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D object detection network for autonomous driving. In *CVPR*, 2017.

[5] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. In *ICLR*, 2022.

[6] Siyuan Dong, Devesh K. Jha, Diego Romeres, Sangwoon Kim, Daniel Nikovski, and Alberto Rodriguez. Tactile-RL for insertion: Generalization to objects of unknown geometry. In *ICRA*, 2021.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[8] Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. Act3D: 3D feature field transformers for multi-task robotic manipulation. In *CoRL*, 2023.

[9] Ankit Goyal and Jia Deng. Packit: A virtual environment for geometric planning. In *International Conference on Machine Learning*, pages 3700–3710. PMLR, 2020.

[10] Ankit Goyal, Hei Law, Bowei Liu, Alejandro Newell, and Jia Deng. Revisiting point cloud shape classification with a simple and effective baseline. In *ICML*, 2021.

[11] Ankit Goyal, Arsalan Mousavian, Chris Paxton, Yu-Wei Chao, Brian Okorn, Jia Deng, and Dieter Fox. IFOR: Iterative flow minimization for robotic object rearrangement. In *CVPR*, 2022.

[12] Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. RVT: Robotic view transformer for 3D object manipulation. In *CoRL*, 2023.

[13] Sagar Gubbi, Shishir Kolathaya, and Bharadwaj Amrutur. Imitation learning for high precision peg-in-hole tasks. In *ICCAR*, 2020.

[14] Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In *CoRL*, 2022.

[15] Abdullah Hamdi, Silvio Giancola, and Bernard Ghanem. MVTN: Multi-view transformation network for 3D shape recognition. In *ICCV*, 2021.

[16] Abdullah Hamdi, Silvio Giancola, and Bernard Ghanem. Voint Cloud: Multi-view point cloud representation for 3D understanding. In *ICLR*, 2023.

[17] Haojie Huang, Owen Howell, Xupeng Zhu, Dian Wang, Robin Walters, and Robert Platt. Fourier Transporter: Bi-equivariant robotic manipulation in 3D. In *ICLR*, 2024.

[18] Shijia Huang, Yilun Chen, Jiaya Jia, and Liwei Wang. Multi-view transformer for 3D visual grounding. In *CVPR*, 2022.

[19] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The robot learning benchmark & learning environment. *RA-L*, 5(2):3019–3026, 2020.

[20] Stephen James, Kentaro Wada, Tristan Laidlow, and Andrew J. Davison. Coarse-to-fine Q-attention: Efficient learning for visual robotic manipulation via discretisation. In *CVPR*, 2022.

[21] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. BC-Z: Zero-shot task generalization with robotic imitation learning. In *CoRL*, 2021.

[22] Michelle A. Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *ICRA*, 2019.

[23] Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, and Daniel Haziza. xFormers: A modular and hackable Transformer modelling library. https://github.com/facebookresearch/xformers, 2022.

[24] Yifang Liu, Diego Romeres, Devesh K. Jha, and Daniel Nikovski. Understanding multi-modal perception using behavioral cloning for peg-in-a-hole insertion tasks. *arXiv preprint arXiv:2007.11646*, 2020.

[25] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Tay-

lor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D deep learning with PyTorch3D. *arXiv preprint arXiv:2007.08501*, 2020.

[26] Eric Rohmer, Surya P. N. Singh, and Marc Freese. V-REP: A versatile and scalable robot simulation framework. In *IROS*, 2013.

[27] Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. In *IROS*, 2020.

[28] Markus Schütz, Bernhard Kerbl, and Michael Wimmer. Rendering point clouds with compute shaders and vertex order optimization. *Computer Graphics Forum*, 40(4): 115–126, 2021.

[29] Lucy Xiaoyang Shi, Archit Sharma, Tony Z. Zhao, and Chelsea Finn. Waypoint-based imitation learning for robotic manipulation. In *CoRL*, 2023.

[30] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-Actor: A multi-task transformer for robotic manipulation. In *CoRL*, 2022.

[31] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *ICCV*, 2015.

[32] Priya Sundaresan, Suneel Belkhale, Dorsa Sadigh, and Jeannette Bohg. KITE: Keypoint-conditioned policies for semantic manipulation. *arXiv preprint arXiv:2306.16605*, 2023.

[33] Bingjie Tang, Michael A. Lin, Iretiayo A. Akinola, Ankur Handa, Gaurav S. Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. IndustReal: Transferring Contact-Rich Assembly Tasks from Simulation to Reality. In *RSS*, 2023.

[34] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *ECCV*, 2020.

[35] Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. ChainedDiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In *CoRL*, 2023.

[36] Jie Xu, Sangwoon Kim, Tao Chen, Alberto Rodriguez Garcia, Pulkit Agrawal, Wojciech Matusik, and Shinjiro Sueda. Efficient tactile simulation with differentiability for robotic manipulation. In *CoRL*, 2022.

[37] Wentao Yuan, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. M2T2: Multi-task masked transformer for object-centric pick and place. In *CoRL*, 2023.

[38] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. In *CoRL*, 2020.

[39] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *RSS*, 2023.