RVT-2: Learning Precise Manipulation from Few Examples

Author Names Omitted for Anonymous Review. Paper-ID 9

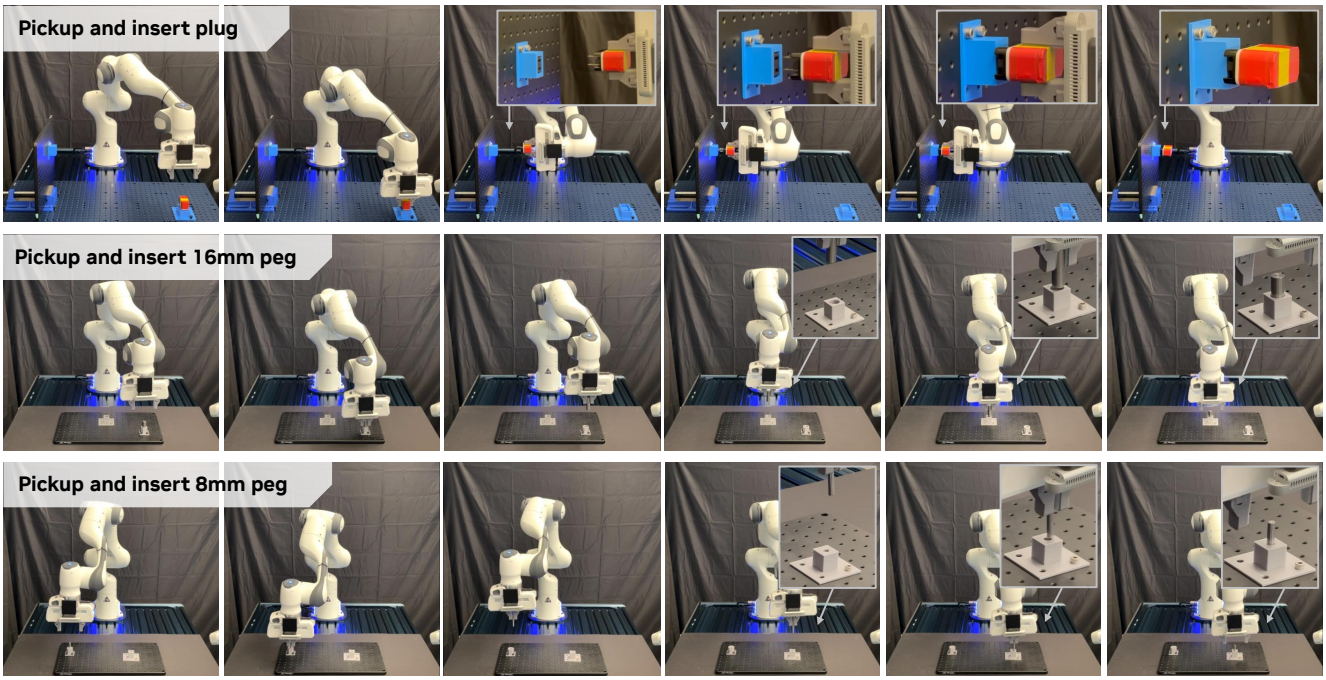

Fig. 1: High Precision Execution in the Real World

***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 domains like manufacturing and home, such a system should be capable of learning new tasks with few demonstrations and solving them with high precision. Prior works like PerAct [39] and RVT [16] have studied this problem, however, they often struggle with tasks requiring high precision. We build upon these prior works and study how to make them more efficacious, precise, and fast. Using a culmination of architectural and system-level improvement, we proposed RVT-2, a multitask 3D manipulation method 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 the RLBench benchmark, improving the success rate from 65% to 78%. 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. Video results can be found at https://rvt2.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 environments. To be affable, such systems should need very few demonstrations of a new task, and also be capable of precise manipulation. For example, in an industrial manufacturing setting, we should expect a person demonstrating a high-precision task like peg insertion to a robot just a few times, after which the robot can start doing that task independently. Similar examples can be found in various 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 [48] and IFOR [15] that studied planar pick-and-place tasks, recent works have gone beyond the 2D plane and studied manipulation in 3D with a few examples [23]. Some notable methods are PerAct [39] and RVT [16]. Given a language instruction, PerAct [39] adopted a multi-task transformer model for 3D manipulation by predicting the next keyframe pose. Even though PerAct achieved impressive performance, it used a voxel-based representation for the scene, limiting its scalability. RVT [16] 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 performance. Compared to PerAct, RVT demonstrated a 36x faster training speed and improved the performance from 48% to 63% on 18 tasks in RLBench [22].

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 closing a jar or inserting a peg. In these tasks, even a **[Ankit: XXmm]** error could result in failure. 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 are able to further boost both the speed and efficacy of RVT. In this work, we propose RVT-2, which improves the training speed by 6X, inference speed by 2X, and task success rate by 16 percentage points on the RLBench Benchmark, achieving the 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 $\sim$10 demonstrations. Specifically, RVT-2 is able to perform tasks requiring very high 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 improvements in RVT were achieved by a culmination of architectural improvement and system-level optimizations. For the architectural improvement, we study three main design innovations. First, we equip RVT with a multi-stage inference pipeline which allows the network to "zoom into" the area of interest and predict more precise end-effector poses. Further, to save GPU memory while training and improving speed, we adopt a convex upsampling design. Lastly, we improve end-effector rotation prediction by utilizing location-conditioned features instead of the originally proposed global features.

For the system-level optimizations, we first create a custom virtual image renderer to replace the generic renderer (viz. PyTorch3D [33]) used in RVT. With this custom accelerated rendering library we further improve the speed and reduce the memory usage for 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 impressive real-world performance. We also provide a careful analysis ablating and quantifying the improvements because of different factors. We will open source our implementation and hope it can help the community.

II. Related Work

Few-Shot Robotic 3D Manipulation. While robotic 2D manipulation confines the robot's motion in two-dimensional planar space and limits the applications to the pick-and-place tasks in 2D top-down setting [38, 15, 37], inferring robots' movements and interactions in full 3D space is a much more challenging problem due to the higher degrees of freedom for robots' action space and the requirement of 3D spatial reasoning. To tackle the robotic 3D manipulation challenge, recent works have been leveraging various perception representations. Camera images have been widely used for vision-based manipulation (examples are RT-1 [1] and ALOHA [49, 12]). For more effective 3D spatial reasoning, recent works leverage depth information and take RGB-D images as input to the policy [41] . PolarNet [4] directly uses the point cloud reconstructed from RGB-D images and processes it with PointNext and Transformer to predict actions. C2F-ARM [23], PerAct [39] and GNFactor [47] voxelize the point clouds and use a 3D convolutional network as the backbone for control inference. Act3D [13] and ChainedDiffuser [44] avoid 3D voxelization and instead represent the scene as a multi-scale 3D feature cloud. To boost both the time efficiency and task efficacy, RVT [16] proposes multi-view virtual images as the scene representation. Most of those prior works are applied on pick-and-place tasks that does not require high control accuracies. Stepping further from these advances, our work extends to the precise manipulation problems. We intend to tackle the tasks that require high motion accuracy by applying a multi-stage strategy to decide which part of the scene to zoom-into and then capture virtual images.

Transformers for Manipulation. Transformer architecture has been widely adopted in robot learning for enhancing control performance [26, 3, 7, 46]. With the flexibility to receive various observations as inputs, transformer-based models have emerged as powerful tools to extract features from multi-modality sensory inputs [8, 2, 27, 25, 30, 49, 16, 12]. Recent works also extend this multi-modal flexibility and seamlessly integrate the Transformer backbone with diffusion models to facilitate long-horizon motion planning [6, 32]. A notable trend in many prior studies is their reliance on extensive datasets, often involving hundreds of demonstrations per task, to train robust transformer architectures. In contrast, our proposed method, RVT-2, demonstrates efficacy in high-precision tasks and proficiency with as few as ten trajectories for each task in real-world scenarios.

High Precision Manipulation. High-precision manipulation is required in the tasks that have low motion error tolerance such as in industrial settings. To learn high-precision manipulation policy, previous works have been using 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 [17]. By leveraging camera images, Schoettler et al. [35] presents a residual reinforcement learning algorithm to accomplish industrial insertion tasks from vision inputs. Tang et al. [42] propose to detect the

peg location from the camera image captured in the first 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 [28, 31] and vision-based tactile sensors [10, 45] are exploited. However, the previous works leverage algorithms requiring expensive data (either reinforcement learning or imitation learning from hundreds of demonstrations) and only learn a model for single task. In contrast to prior works, our work RVT-2 is able to learn a multi-task high-precision manipulation policy from very few demos per task.

Virtual Views for 3D. The incorporation of virtual views provides a strategic avenue for integrating well-established image-processing architectures, such as Convolutional Neural Networks and Transformer models, into processing 3D scene information. Prior works have successfully employed virtually rendered views across various vision tasks, such as object detection [5], object recognition [40, 14, 19, 20], and 3D visual grounding [21] from a point cloud of the scene, and demonstrate it outperforms sophisticated point-based methods. The application of virtual views was relatively limited in the field of robotics until recently. RVT [16] leverages multi-view representation for predicting robot actions for object manipulation. Our work is built upon RVT but extends it by introducing a novel two-layer hierarchical view representation specially designed for tasks demanding high precision.

III. Method

Our proposed method, RVT-2, allows for precise and three-dimensional manipulation. A single RVT-2 model could solve multiple tasks based on the input instructions and requires only a few demonstrations per task.

RVT-2 builds upon the state-of-the-art models for 3D object manipulation, RVT [16]. Similar to RVT, RVT-2 adopts the paradigm of key-point based manipulation. It then introduces a series of improvements, resulting in better performance, precision, and speed. We group these improvements in two categories: architectural for those related to changes in the neural network architecture and system's for those related to software optimizations.

A. Background

Key-point based manipulation. In key-point 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,' 'grasping drawer handle', and 'pull-pose for the drawer.' These key poses are provided in the training dataset, and the aim of a key-point based method is to learn to predict these poses. Specifically, methods like RVT and PerAct take as input the language goal and current scene point clouds and predict the next key-point pose. The predicted pose is then sent to a motion planner, which generates a trajectory towards it[1]. When the robot reaches the predicted pose, the method takes in the updated scene point cloud and predicts the subsequent key-point pose. This process terminates when the task is successful or a predefined number of steps is reached.

To train a key-point based behavior cloning agent, we assume access to a dataset of samples. Each sample includes a language goal, current visual observation, and the next key-point pose. One way to extract such a dataset automatically from dense robot trajectory datasets is using rules that define the key-point poses as done in Shridhar et al. [39]. We encourage readers to refer to Shridhar et al. [39] for a detailed overview of this process. In this work, we use the key-point pose extraction scheme as PerAct and RVT.

Robotic View Transformer (RVT). RVT is a state-of-the-art model for 3D object manipulation. It is a key-point-based method that takes as input the language description of the goal along with the current scene point cloud and predicts the next key-point pose.

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

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 across the views. The 3D point with the largest heatmap score represents the predicted gripper location. Along with the heatmaps, RVT concatenates global features from across the views to predict the gripper rotation and state (open or close)[2]. We encourage interested readers to refer to [16] for a detailed overview of RVT.

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

Hierarchical Design. RVT uses a fixed set of views around the robot to predict the final pose in a single shot. These views might not be sufficient for tasks like inserting a peg in the hole where the gripper pose has to be very precise. Hence RVT-2 adopts a hierarchical design, where in the coarse stage, it makes a gripper location prediction using similar views as RVT. Using the prediction in the coarse stage, RVT-2 then zooms in the area of interest and re-renders images around it. RVT-2 then uses these zoomed-in images to make fine gripper pose predictions. Note that this zoomed-in rendering is possible because of the flexibility provided by virtual rendering in RVT.

Convex Upsampling. RVT requires up-sampling of features of dimension d from the token resolution $t_1 \times t_2$ to the

[1]Along with the pose, these method also output whether the motion planner should consider or ignore collision

[2]It also predicts whether the motion planner should avoid or disregard the collision of the robot with other objects in the scene

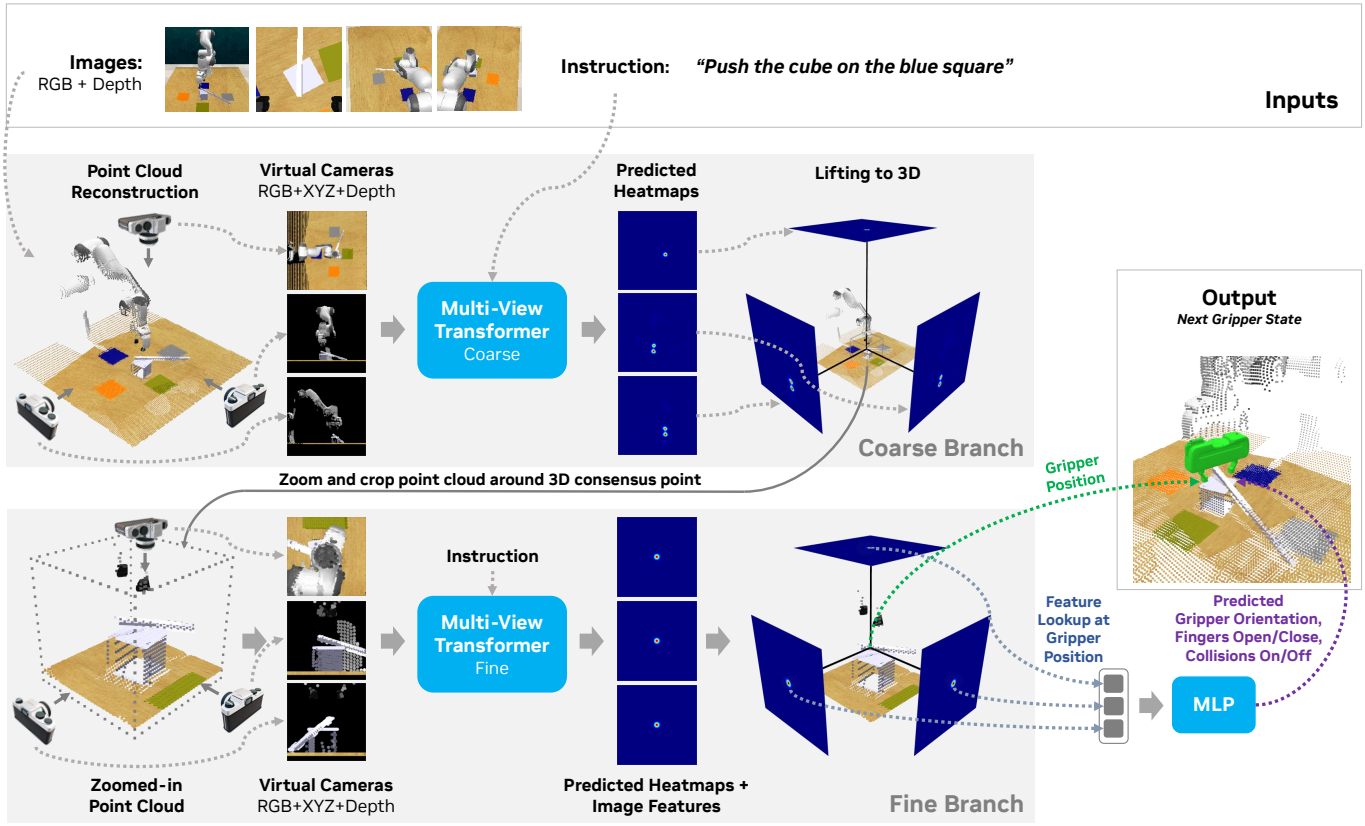

Fig. 3: RVT-2 Architecture

image resolution of $h \times w$. These features are then used to predict single-channeled heat maps for each image. For this, it uses transposed convolutions. However, this operation is GPU memory intensive as it creates a dense feature of size $h \times w \times d$. To address this, RVT-2 removes feature up-sampling and directly predicts heatmaps of shape $h \times w$ from features at the token resolution. Specifically, it uses convex upsampling layer proposed by [43]. The convex upsampling layer uses a learned convex combination of features in the coarse grid to make predictions in the higher resolution. [43] show how it leads to sharper predictions at higher resolution. We find that convex upsampling saves memory without sacrificing performance. It can be implemented using native 'fold' function in PyTorch.

Parameter Rationalization. We find that network parameters in RVT, like image size of 220 and patch size of 11, may not be optimal for GPU as they are not divisible by powers of 2 like 16[3]. RVT-2 rationalizes these network parameters to make them more GPU-friendly, improving their speed. RVT-2 adopts parameters similar to ViT [11], 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 $(\frac{image\ size}{patch\ size})^2$, improving speed further. These choices make RVT-2 faster during training and testing without affecting performance.

Location Conditioned Rotation Prediction. RVT and PerAct use only global visual features, like max-pool over the entire image, to make predictions for end-effector rotation. This can be an issue 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. In this case, picking either of the two blocks is a valid step. However, since the blocks have different orientations, the end-effector rotation should 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 enables 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, including back, front, top, left, and right. [16] find that for RVT, using fewer camera views reduced performance. However, we find that in our hierarchical RVT-2 model, it is sufficient to use only three views, i.e., front, top, and right, without sacrificing performance. This is likely because we RVT-2 used zoomed-in views to make the final prediction. Using fewer camera views improves performance as it reduces the number of images that the render needs to render and the number of tokens that the

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

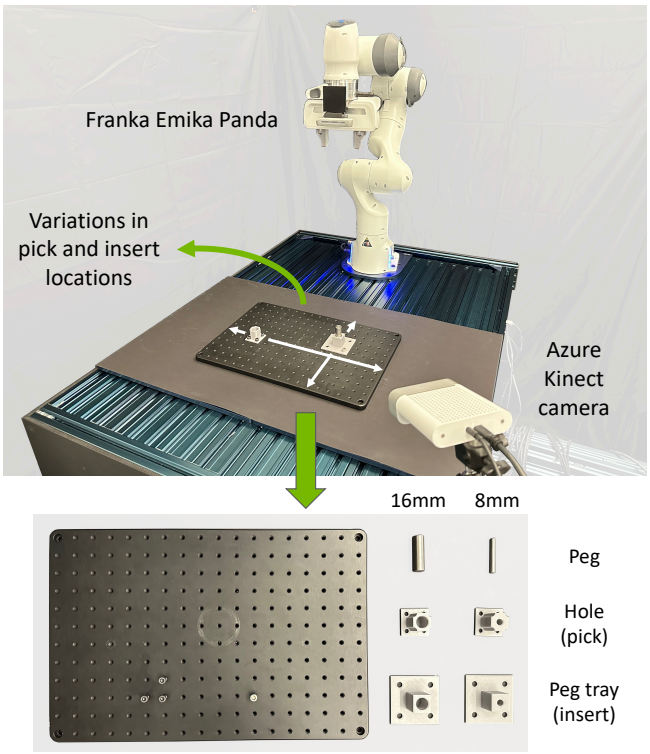

Fig. 4:

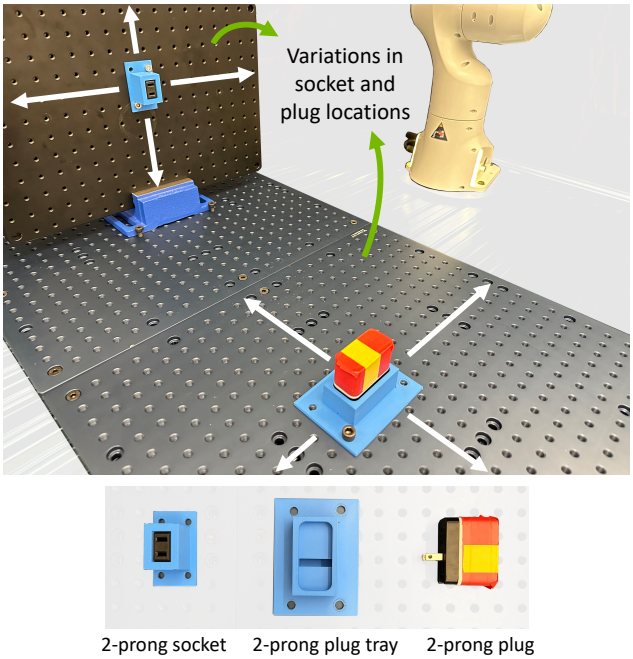

Fig. 5:

multi-view transformer needs to process.

C. System's Changes: RVT → RVT-2

Point-Renderer. RVT uses PyTorch3D [33] to render images at virtual views. PyTorch3D is an appealing choice because of its easy-to-use interface. However, it is a fully-featured differentiable renderer that incurs significant time and memory overhead unjustifiable for point-cloud rendering. To avoid this overhead, 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 a feature image and depth image of size (h, w):

1. Projection. For each 3D point of index $n \in 0, 1, \ldots, N-1$, compute the depth d_n and image pixel coordinate (u_n, v_n) using camera intrinsics and extrinsics. From the pixel coordinate, resolve the linear pixel index $i_n = u_n \cdot w + v_n$. The projection operation is easily accelerated using GPU matrix multiplications.

2. Z-ordering. For each pixel of index j in the image, find the point index with smallest depth d_n among the set of points that project to the pixel $\{n \mid i_n = j\}$. Store the point feature vector f_n in the feature image and depth d_n in the depth image at pixel j.

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. The first, in a parallel loop over point cloud points, tries to store that points 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. [36] 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.

3. 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-rendering, 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 $r \cdot \texttt{focal\_length}/d_k$, replace the feature and depth of pixel j with that of pixel k.

We will release our point renderer as a standalone python library alongside the RVT-2 code release.

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 [9], and memory-aware attention based on xFormers [29].

IV. Experiments

We conducted comprehensive experiments in both simulation and real-world settings to demonstrate the efficacy of our approach. The experiments are designed to answer the following pivotal questions: (1) How does RVT-2's training efficiency compare to state-of-the-art 3D manipulation methods in terms of training time and memory cost? (2) Can RVT-2 successfully execute high-precision tasks with only a minimal set of demonstrations? (3) To what extent do the architectural and system-level modifications introduced in RVT-2 enhance its performance over its predecessor, RVT?

A. Simulation Results

Experiment Setup. We first evaluate the multi-task performance of our approach in simulation. For a fair comparison, we conduct the experiments on a standard multi-task manipulation benchmark developed in RLBench [22] that has been widely adopted by previous works [16, 39, 13]. The benchmark task set contains 18 tasks including nonprehensile 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 has 2-60 variations such as manipulating 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 [34]. The input RGB-D images are in a resolution of 128×128 and are captured by four noiseless cameras mounted at the front, left shoulder, right shoulder, and wrist. We train our RVT-2 model with the same dataset as previous work [16, 39, 13], where 100 training demonstrations are provided for each task and 25 unseen demonstrations are used for evaluation. We compare RVT-2 against 6 baselines: (1) **Image-BC (CNN)** [24] (2) **Image-BC (ViT)** (3) **C2F-ARM-BC** [22] (4) **PerAct** [39] (5) **Act3D** [13] (6) **RVT** [16]

Training Details. **[Jie: @ankit: add training details]**

Experiment Results. Table I summarizes the comparisons of RVT-2 against baselines on RLBench tasks.

a) Multi-Task Performance: Among all the baselines and state-of-the-art approaches, RVT-2 achieves the highest success rates on 12 out of 18 tasks. Comparing the average success rate across all tasks, RVT-2 also achieves a 12.6% success rate boost over the best-performing baseline Act3D [13] while only requiring $4\times$ shorter training time. The average rank of 1.8 is also a strong evidence of the superiority of RVT-2's performance over baselines.

[Jie: @ankit: maybe add some words for close jar or open drawer which RVT-2 is much worse than Act3D?]

b) High-Precision Task Performance: One of the key features of RVT-2 is its special two-stage architectural design for high-precision tasks such as the "Insert Peg" task. In this task, the robot is asked to 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 robot has to align the square peg perfectly with the cuboid stick otherwise any tiny error will result in a failure insertion. Our experiments show that RVT-2 achieves a significantly higher success rate (40%) on "Insert Peg" task than all other baselines (Act3D is the best baseline on this task with 27% success rate).

c) Training and Inference Efficiency: RVT-2 is also featured by its efficient training time and outstanding inference time that is enabled by the improved architectural design and the replacement of the point render. From the table, while RVT has significantly improved the training efficiency compared to other baselines, RVT-2 achieves an even higher success rate given the same amount of model training time as RVT. Those meticulous design choices also result in a $2\times$ speed up of RVT-2's inference time to RVT. Such high inference speed also opens up new possibilities for utilizing RVT-2 as a real-time reactive control framework in the future.

B. Real World Results

Experiment Setup. **[Jie: @ankit @yu-wei: could you write something about the real experiment setup?]**

Experiment Results. Table II shows the experiment results in the real world. The results show that RVT-2 can learn a performing multi-task policy in real with only a handful set of demonstrations per task. Of the five tasks originally introduced by RVT[16], RVT-2 attains higher success rates in two tasks and achieves a tie in the remaining two. More importantly, on all three newly introduced tasks that require high precisions, RVT-2 constantly beats RVT and achieves 53.3% average success rates with as few as 10 demonstrations per task.

C. Ablations

We further conduct an extensive ablation study in simulation to analyze the effect of each component of RVT-2 and the results are shown in Table III. Specifically, we ablate the architectural changes including the introduction of the hierarchical architecture ("Hier. Design"), using GPU-friendly network parameters ("Parameter Rationalization"), adopting local feature pooling for rotation prediction ("Loc. Cond. Rot"), leveraging convex upsampling for saving memory ("Convex Upsampl."), and the number of virtual views to be rendered ("No. of Views"). We also ablate the systems' changes including the replacement of the point render ("Point Render"), **[Jie: "AMP"]**, **[Jie: "8-bit Opt. + Fast Atten."]** **[Jie: complete after results are ready]**

a) Hier. Design: Comparing Row 1 and Row 2 in Table III, we can see that including hierarchical design introduces a slowdown in the training time due to the extra stage of rendering and inference, however it brings a 14.2% success rate improvement since the zoom-in view provides more task-relevant details about the workspace region of interest.

b) Parameter Rational.: Comparing Row 1 and Row 3, by using more GPU-friendly network parameters, the training process is significantly sped up while the performance of the policy is not sacrificed any.

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) [24, 39]	1.3	7.4	-	-	0	0	0	0	4	0	0
Image-BC (ViT) [24, 39]	1.3	7.7	-	-	0	0	0	0	0	0	0
C2F-ARM-BC [23, 39]	20.1	5.8	-	-	24	24	4	20	20	0	8
HiveFormer [18]	45.3	5.2	-	-	52.0	76.0	0.0	**100.0**	52.0	0.0	80
PolarNet [4]	46.4	4.8	-	-	36.0	92.0	4.0	**100.0**	84.0	0.0	40
PerAct [39]	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 [13]	65.0	2.8	5.0		**92.0**	92.0	27.0	94.0	**93.0**	3.0	80
RVT [16]	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)	**77.6**	**1.8**	**1.0**	**20.6**	32.0 ± 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) [24, 39]	0	0	8	4	0	0	0	0	0	0	8
Image-BC (ViT) [24, 39]	0	0	0	0	0	0	0	0	0	0	16
C2F-ARM-BC [23, 39]	72	0	4	12	8	16	8	0	0	0	68
HiveFormer [18]	84	32.0	68.0	76.0	8.0	64.0	8.0	8.0	0.0	28.0	80
PolarNet [4]	96	12.0	32.0	84.0	44.0	56.0	12.0	4.0	8.0	52.0	80
PerAct [39]	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 [13]	99	51.0	90.0	95.0	47.0	**93.0**	8.0	12.0	9.0	92.0	94
RVT [16]	**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.** Performance of HiveFormer and PolarNet as reported by [4], Act3D as reported by [13], RVT and PerAct as reported by [16].

Task	# of variations	# of train	# of test	*Models* RVT [16]	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 [16]	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 of the real-world experiments. A single RVT-2 model can perform well on most tasks with only a few demonstrations.

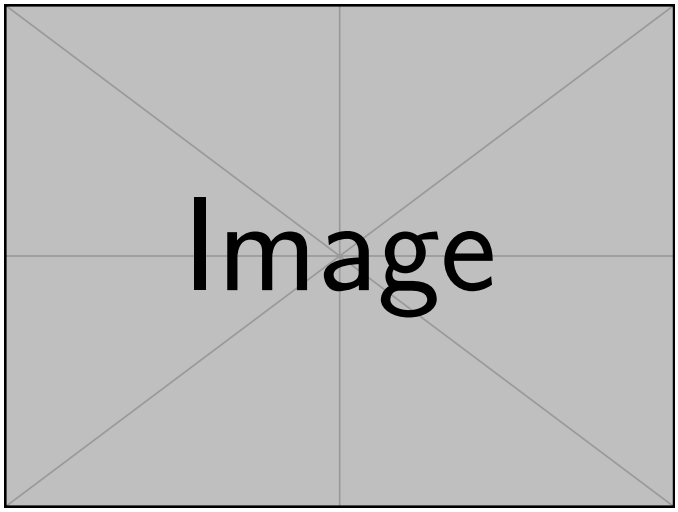

Fig. 6: Acc vs time plot comparing RVT-2, RVT and Peract

V. Conclusions and Limitations

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] Théo Cachet, Julien Perez, and Seungsu Kim. Transformer-based meta-imitation learning for robotic manipulation. In *NeurIPS Workshop on Robot Learning*. 2020.

[3] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. Differentiable spatial planning using transformers. In *ICML*, 2021.

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

[5] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia.

Row ID	Hier. Design	Parameter Rational.	Loc. Cond. Rot	Point Render	Convex Upsampl.	AMP	8-bit Opt. + Fast Atten.	No. of Views	Training Time (in hours)	Training Time % rel. base	Average Success	Avg. Succ diff. wrt. base
1	✓	✓	✓	✓	✓	✓	✓	3	19.5	0%	77.6	0
2	✗	✓	✓	✓	✓	✓	✓	3	13.0	-33%	63.4	- 14.2
3	✓	✗	✓	✓	✓	✓	✓	3	26.7	+37%	74.1	- 3.5
4	✓	✓	✗	✓	✓	✓	✓	3	19.3	-1%	74.7	-2.9
5	✓	✓	✓	✗	✓	✓	✓	3	82.1	+321%	73.4	-4.2
6	✓	✓	✓	✓	✗	✗	✓	3	62.4	+220%	77.2	-0.4
7	✓	✓	✓	✓	✓	✗	✓	3	58.5	+200%	76.9	-0.7
8	✓	✓	✓	✓	✓	✗	✗	3	79.2	+306%	77.8	+0.2
9	✓	✓	✓	✓	✓	✓	✓	5	40.3	+107%	75.6	-2.0

TABLE III: Ablations on RLBench. A larger res., adding view correspondence, adding depth channel, separating initial attention layers, orthographic projection, using rotation aug., and re-rendered views around the cube improve the performance.

Multi-view 3D object detection network for autonomous driving. In *CVPR*, 2017.

[6] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *RSS*, 2023.

[7] Henry M. Clever, Ankur Handa, Hammad Mazhar, Kevin Parker, Omer Shapira, Qian Wan, Yashraj Narang, Iretiayo Akinola, Maya Cakmak, and Dieter Fox. Assistive tele-op: Leveraging transformers to collect robotic task demonstrations. In *NeurIPS Workshop on Robot Learning*. 2021.

[8] Sudeep Dasari and Abhinav Gupta. Transformers for one-shot visual imitation. In *CoRL*, 2020.

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

[10] 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.

[11] 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.

[12] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile ALOHA: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

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

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

[22] 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.

[23] 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.

[24] 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.

[25] Rishabh Jangir, Nicklas Hansen, Sambaran Ghosal, Mohit Jain, and Xiaolong Wang. Look closer: Bridging egocentric and third-person views with transformers for robotic manipulation. *RA-L*, 7(2):3046–3053, 2022.

[26] Jacob J. Johnson, Uday S. Kalra, Ankit Bhatia, Linjun Li, Ahmed H. Qureshi, and Michael C. Yip. Motion Planning Transformers: A motion planning framework for mobile robots. *arXiv preprint arXiv:2106.02791*, 2021.

[27] Heecheol Kim, Yoshiyuki Ohmura, and Yasuo Kuniyoshi. Transformer-based deep imitation learning for dual-arm robot manipulation. In *IROS*, 2021.

[28] 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.

[29] 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.

[30] Weiyu Liu, Chris Paxton, Tucker Hermans, and Dieter Fox. StructFormer: Learning spatial structure for language-guided semantic rearrangement of novel objects. In *ICRA*, 2022.

[31] 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.

[32] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. https://octo-models.github.io, 2023.

[33] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D deep learning with PyTorch3D. *arXiv preprint arXiv:2007.08501*, 2020.

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

[35] 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.

[36] 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.

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

[38] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. CLIPort: What and where pathways for robotic manipulation. In *CoRL*, 2021.

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

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

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

[42] 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.

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

[44] 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.

[45] 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.

[46] Ruihan Yang, Minghao Zhang, Nicklas Hansen, Huazhe Xu, and Xiaolong Wang. Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers. In *ICLR*, 2022.

[47] Yanjie Ze, Ge Yan, Yueh-Hua Wu, Annabella Macaluso, Yuying Ge, Jianglong Ye, Nicklas Hansen, Li Erran Li, and Xiaolong Wang. GNFactor: Multi-task real robot learning with generalizable neural feature fields. In *CoRL*, 2023.

[48] 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.

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