

PERCEIVER-ACTOR: A Multi-Task Transformer for Robotic Manipulation

Anonymous Author(s)

Affiliation

Address

email

peract.github.io

1 **Abstract:** Transformers have revolutionized vision and natural language processing
2 with their ability to scale with large datasets. But in robotic manipulation,
3 data is both limited and expensive. Can we still benefit from Transformers with
4 the right problem formulation? We investigate this question with PERACT, a
5 language-conditioned behavior-cloning agent for multi-task 6-DoF manipulation.
6 PERACT encodes language goals and RGB-D voxel observations with a Perceiver
7 Transformer [1], and outputs discretized actions by “detecting the next best voxel
8 action”. Unlike frameworks that operate on 2D images, the voxelized observation
9 and action space provides a strong structural prior for efficiently learning 6-DoF
10 policies. With this formulation, we train a single multi-task Transformer for 18
11 RLBench tasks (with 249 variations) and 7 real-world tasks (with 18 variations)
12 from just a few demonstrations per task. Our results show that PERACT signifi-
13 cantly outperforms unstructured image-to-action agents and 3D ConvNet base-
14 lines for a wide range of tabletop tasks.

15 **Keywords:** Transformers, Language Grounding, Manipulation, Behavior Cloning

16 1 Introduction

17 Transformers [2] have become prevalent in natural language processing and computer vision. By
18 framing problems as sequence modeling tasks, and training on large amounts of diverse data, Trans-
19 formers have achieved groundbreaking results in several domains [3, 4, 5, 6]. Even in domains that
20 do not conventionally involve sequence modeling [7, 8], Transformers have been adopted as a *gen-*
21 *eral* architecture [9]. But in robotic manipulation, data is both limited and expensive. Can we still
22 bring the power of Transformers to 6-DoF manipulation with the right problem formulation?

23 Language models operate on sequences of tokens [10], and vision transformers operate on sequences
24 of image patches [4]. While pixel transformers [11, 1] exist, they are not as data efficient as ap-
25 proaches that use convolutions or patches to exploit the 2D structure of images. Thus, while Trans-
26 formers may be domain agnostic, they still require the right problem formulation to be data efficient.
27 A similar efficiency issue is apparent in behavior-cloning (BC) agents that directly map 2D images
28 to 6-DoF actions. Agents like Gato [9] and BC-Z [12, 13] have shown impressive multi-task ca-
29 pabilities, but they require several weeks or even months of data collection. In contrast, recent
30 works in reinforcement-learning like C2FARM [14] construct a voxelized observation and action
31 space to efficiently learn visual representations of 3D actions with 3D ConvNets. Similarly, in this
32 work, we aim to exploit the 3D structure of *voxel patches* for efficient 6-DoF behavior-cloning with
33 Transformers, analogous to how vision transformers [4] exploit the 2D structure of image patches.

34 To this end, we present PERACT (short for PERCEIVER-ACTOR), a language-conditioned BC agent
35 that can learn to imitate a wide variety of 6-DoF manipulation tasks with just a few demonstra-
36 tions per task. PERACT encodes a sequence of RGB-D voxel patches and predicts discretized transla-
37 tions, rotations, and gripper actions that are executed with a motion-planner in an observe-act loop. PER-

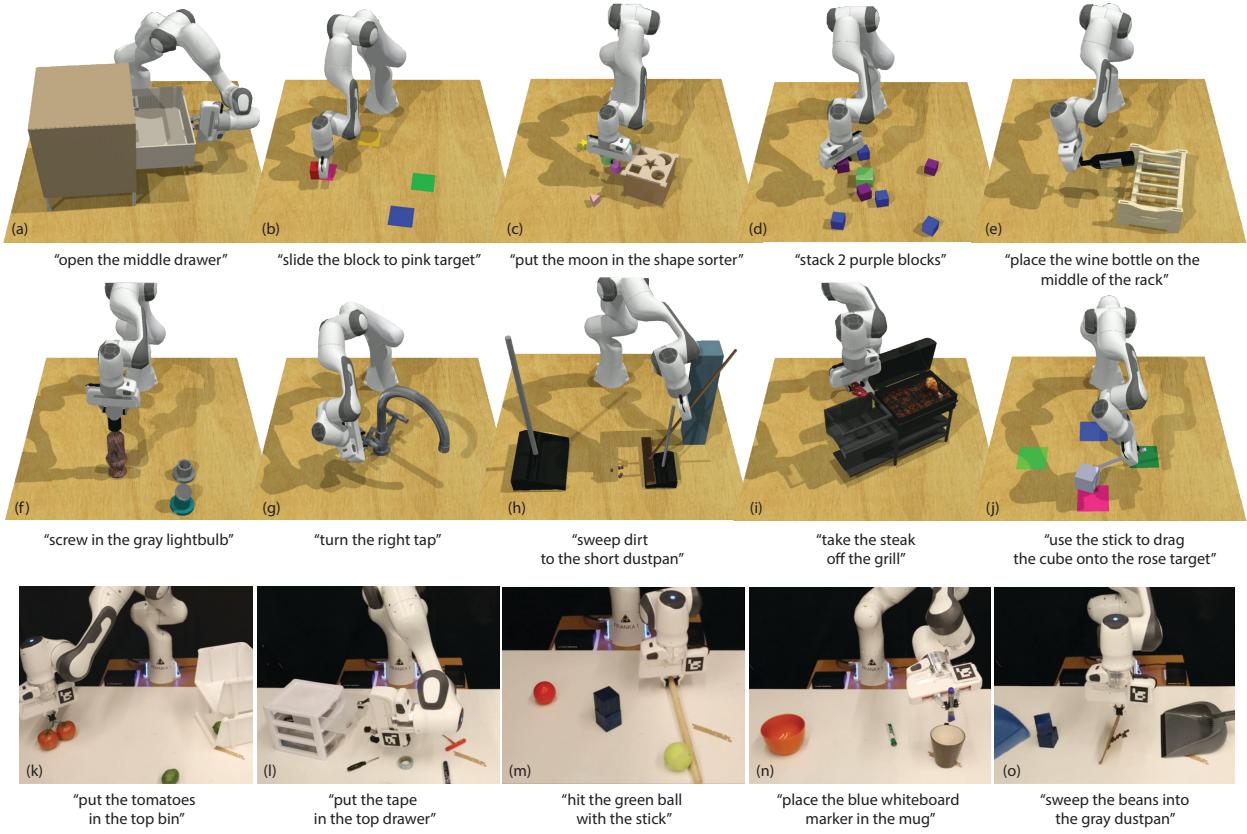


Figure 1. Language-Conditioned Manipulation Tasks: PERACT is a language-conditioned multi-task agent capable of imitating a wide range of 6-DoF manipulation tasks. We conduct experiments on 18 simulated tasks with 249 unique variations in RLBench [15] (a-j; only 10 shown), each with several pose and semantic variations. We also demonstrate our approach with a Franka Panda on 7 real-world tasks (k-o; only 5 shown) with a multi-task agent trained with just 53 demonstrations. See the supplementary video for simulated and real-world rollouts.

38 ACT is essentially a classifier trained with supervised learning to *detect actions* akin to prior work
 39 like CLIPort [16], except our observations and actions are represented with 3D voxels instead of 2D
 40 image pixels. Voxel grids are less prevalent than images in end-to-end BC approaches often due to
 41 scaling issues with high-dimensional inputs. But in PERACT, we use a Perceiver¹ Transformer [1]
 42 to encode very high-dimensional input of up to 1 million voxels with only a small set of latent vec-
 43 tors. This voxel-based formulation provides a strong structural prior with several benefits: a natural
 44 method for fusing multi-view observations, learning robust action-centric² representations [17], and
 45 enabling data augmentation in 6-DoF – all of which help learn generalizable skills by focusing on
 46 *diverse* rather than narrow multi-task data.

47 To study the effectiveness of this formulation, we conduct large-scale experiments in the RL-
 48 Bench [15] environment. We train a single multi-task agent on 18 diverse tasks (with 249 variations)
 49 that involve a range of prehensile and non-prehensile behaviors like placing wine bottles on a rack
 50 and dragging objects with a stick (see Figure 1 a-j). Each task also includes several pose and seman-
 51 tic variations with objects that differ in placement, color, shape, size, and category. Our results show
 52 that PERACT significantly outperforms image-to-action agents (by 43×) and 3D ConvNet baselines
 53 (by 3.6×), without using any explicit representations of instance segmentations, object poses, mem-
 54 ory, or symbolic states. We also validate our approach on a Franka Panda with a multi-task agent
 55 trained *from scratch* on 7 real-world tasks with a **total of just 53 demonstrations** (see Figure 1 k-o).

56 In summary, our contributions are as follows:

- 57 • **A novel problem formulation** for perceiving, acting, and specifying goals with Transformers.
- 58 • An efficient **action-centric** framework for **grounding language in 6-DoF actions**.
- 59 • **Empirical results** investigating multi-task agents on a range of simulated and real-world tasks.

60 The code and pre-trained models will be available upon publication at peract.github.io.

¹Throughout the paper we refer to PerceiverIO [1] as Perceiver for brevity.

²Action-centric refers to a perception system that learns visual representations of actions; see Appendix J.

61 **2 Related Work**

62 **Vision for Manipulation.** Traditionally, methods in robot perception have used explicit “object”
63 representations like instance segmentations, object classes, poses [18, 19, 20, 21, 22, 23]. Such
64 methods struggle with deformable and granular items like cloths and beans that are hard to represent
65 with geometric models or segmentations. In contrast, recent methods [16, 24] learn action-centric
66 representations without any “objectness” assumptions, but they are limited to top-down 2D settings
67 with simple pick-and-place primitives. In 3D, James et al. proposed C2FARM [14], an action-centric
68 reinforcement learning (RL) agent with a coarse-to-fine-grain 3D-UNet backbone. The coarse-to-
69 fine-grain scheme has a limited receptive field that cannot look at the entire scene at the finest level.
70 In contrast, PERACT learns action-centric representations with a global-receptive field through a
71 Transformer backbone. Also, PERACT does BC instead of RL, which enables us to easily train a
72 multi-task agent for several tasks by conditioning it with language goals.

73 **End-to-End Manipulation** approaches [25, 26, 27, 28] make the least assumptions about objects
74 and tasks, but are often formulated as an image-to-action prediction task. Training directly on RGB
75 images for 6-DoF tasks is often inefficient, generally requiring several demonstrations or episodes
76 just to learn basic skills like rearranging objects. In contrast, PERACT uses a voxelized observation
77 and action space, which is dramatically more efficient and robust in 6-DoF settings. While other
78 works in 6-DoF grasping [29, 30, 31, 32, 33] have used RGB-D and pointcloud input, they have not
79 been applied to sequential tasks or used with language-conditioning. Another line of work tackles
80 data inefficiency by using pre-trained image representations [16, 34, 35] to bootstrap BC. Although
81 our framework is trained from scratch, such pre-training approaches can be integrated together in
82 future works for even greater efficiency and generalization to unseen objects.

83 **Transformers for Agents and Robots.** Transformers have become the prevalent architecture in
84 several domains. Starting with NLP [2, 3, 36], recently in vision [4, 37], and even RL [8, 38, 39].
85 In robotics, Transformers have been applied to assistive teleop [40], legged locomotion [41], path-
86 planning [42, 43], imitation learning [44, 45], and grasping [46]. Transformers have also achieved
87 impressive results in multi-domain settings like in Gato [9] where a single Transformer was trained
88 for 16 domains such as captioning, language-grounding, robotic control etc. However, Gato relies on
89 extremely large datasets like 15K episodes for block stacking and 94K episodes for Meta-World [47]
90 tasks. Our voxel-based approach might complement agents like Gato, which could use our problem
91 formulation for greater efficiency and robustness in 6-DoF manipulation settings.

92 **Language Grounding for Manipulation.** Several works have proposed methods for grounding
93 language in robot actions [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58]. However, these methods use
94 disentangled pipelines for perception and action, with the language primarily being used to guide
95 perception. Recently, a number of end-to-end approaches [13, 12, 59] have been proposed for con-
96 ditioning BC agents with language instructions. These methods require thousands of human teleop-
97 erated demonstrations that are collected over several days or even months. In contrast, PERACT can
98 learn robust multi-task policy with just a few minutes of training data. For benchmarking, several
99 simulation environments exist [60, 24, 47], but we use RLBench [15] for its diversity of 6-DoF tasks
100 and ease of generating expert demonstrations with templated language goals.

101 **3 PERCEIVER-ACTOR**

102 PERACT is a language-conditioned behavior-cloning agent for 6-DoF manipulation. The key idea
103 is to learn perceptual representations of actions conditioned on language goals. Given a voxelized
104 reconstruction of a scene, we use a Perceiver Transformer [1] to learn per-voxel features. Despite the
105 extremely large input space (100^3), Perceiver uses a small set of latent vectors to encode the input.
106 The encoded features are then used to predict the next best action in terms of discretized translation,
107 rotation, and gripper state at each timestep. PERACT relies purely on the current observation to
108 determine what to do next in sequential tasks. See Figure 2 for an overview of PERACT.

109 Section 3.1 and Section 3.2 describe our dataset setup. Section 3.3 describes our problem formula-
110 tion for PERACT, and Section 3.4 provides details on training PERACT.

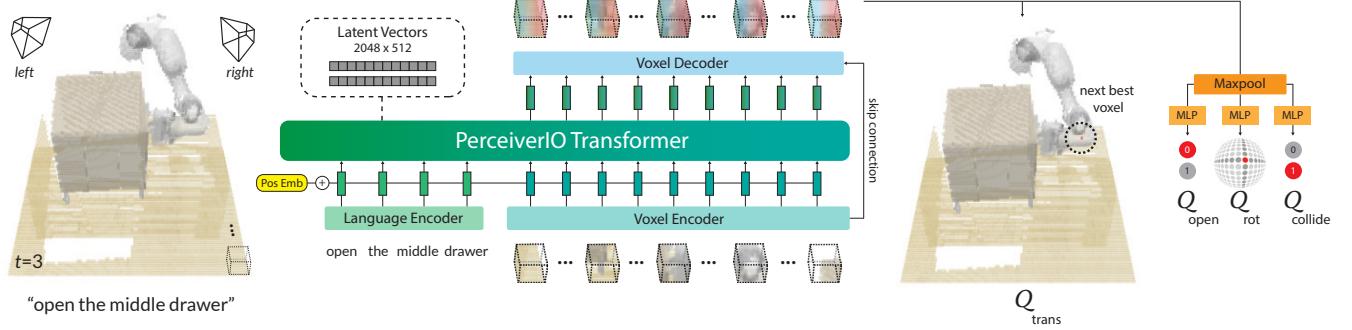


Figure 2. PERACT Overview. PERACT is a language-conditioned behavior-cloning agent trained with supervised learning to *detect actions*. PERACT takes as input a language goal and a voxel grid reconstructed from RGB-D sensors. The voxels are split into 3D patches, and the language goal is encoded with a pre-trained language model. These language and voxel features are appended together as a sequence and encoded with a Perceiver transformer [1]. Despite the extremely long input sequence, Perceiver uses a small set of latent vectors to encode the input (see Appendix Figure 6 for an illustration). These encodings are upsampled back to the original voxel dimensions with a decoder and reshaped with linear layers to predict a discretized translation, rotation, gripper open, and collision avoidance action. This action is executed with a motion-planner after which the new observation is used to predict the next discrete action in an observe-act loop until termination.

111 3.1 Demonstrations

112 We assume access to a dataset $\mathcal{D} = \{\zeta_1, \zeta_2, \dots, \zeta_n\}$ of n expert demonstrations, each paired with
 113 English language goals $\mathcal{G} = \{l_1, l_2, \dots, l_n\}$. These demonstrations are collected by an expert with
 114 the aid of a motion-planner to reach intermediate poses. Each demonstration ζ is a sequence of
 115 continuous actions $\mathcal{A} = \{a_1, a_2, \dots, a_t\}$ paired with observations $\mathcal{O} = \{\tilde{o}_1, \tilde{o}_2, \dots, \tilde{o}_t\}$. An action
 116 a consists of the 6-DoF pose, gripper open state, and whether the motion-planner used collision
 117 avoidance to reach an intermediate pose: $a = \{a_{\text{pose}}, a_{\text{open}}, a_{\text{collide}}\}$. An observation \tilde{o} consists
 118 of RGB-D images from any number of cameras. We use four cameras for simulated experiments
 119 $\tilde{o}_{\text{sim}} = \{o_{\text{front}}, o_{\text{left}}, o_{\text{right}}, o_{\text{wrist}}\}$, but just a single camera for real-world experiments $\tilde{o}_{\text{real}} = \{o_{\text{front}}\}$.

120 3.2 Keyframes and Voxelization

121 Following prior work by James et al. [14], we construct a structured observation and action space
 122 through keyframe extraction and voxelization.

123 Training our agent to directly predict continuous actions is inefficient and noisy. So instead, for
 124 each demonstration ζ , we extract a set of keyframe actions $\{k_1, k_2, \dots, k_m\} \subset \mathcal{A}$ that capture
 125 bottlenecks [61] in the action sequence with a simple heuristic: an action is a keyframe if (1) the
 126 joint-velocities are near zero and (2) the gripper open state has not changed. Each datapoint in the
 127 demonstration ζ can then be cast as a “predict the next (best) keyframe action” task [14, 62, 63]. See
 128 Appendix Figure F for an illustration of this process.

129 To learn action-centric representations [17] in 3D, we use a voxel grid [64, 65] to represent both the
 130 observation and action space. The observation voxels v are reconstructed from RGB-D observations
 131 \tilde{o} fused through triangulation $\tilde{o} \Rightarrow v$ from known camera extrinsics and intrinsics. By default, we
 132 use a voxel grid of 100^3 , which corresponds to a volume of $1.0m^3$ in metric scale. The keyframe
 133 actions k are discretized such that training our BC agent can be formulated as a “next best action”
 134 classification task [14]. Translation is simply the closest voxel to the center of the gripper fingers.
 135 Rotation is discretized into 5 degree bins for each of the three rotation axes. Gripper open state is a
 136 binary value. Collide is also a binary value that indicates if the motion-planner should avoid every-
 137 thing in the voxel grid or nothing at all; switching between these two modes of collision avoidance
 138 is crucial as tasks often involve both contact based (e.g., pulling the drawer open) and non-contact
 139 based motions (e.g., reaching the handle without colliding into anything).

140 3.3 PERACT Agent

141 PERACT is a Transformer-based [2] agent that takes in a voxel observation and language goal (v, l) ,
 142 and outputs a discretized translation, rotation, and gripper open action. This action is executed with
 143 a motion-planner, after which this process is repeated until the goal is reached.

144 The language goal l is encoded with a pre-trained language model. We use CLIP’s [66] language
 145 encoder, but any pre-trained language model would suffice [13, 59]. Our choice of CLIP opens up
 146 possibilities for future work to use pre-trained vision features that are aligned with the language for
 147 better generalization to unseen semantic categories and instances [16].

148 The voxel observation \mathbf{v} is split into 3D patches of size 5^3 (akin to vision-transformers like ViT [4]).
 149 In implementation, these patches are extracted with a 3D convolution layer with a kernel-size and
 150 stride of 5, and then flattened into a sequence of voxel encodings. The language encodings are fine-
 151 tuned with a linear layer and then appended with the voxel encodings to form the input sequence.
 152 We also add learned positional embeddings to the sequence to incorporate voxel and token positions.

153 The input sequence of language and voxel encodings is extremely long. A standard Transformer
 154 with $\mathcal{O}(n^2)$ self-attention connections and an input of $(100/5)^3 = 8000$ patches is hard to fit on the
 155 memory of a commodity GPU. Instead, we use the Perceiver [1] Transformer. Perceiver is a latent-
 156 space Transformer, where instead of attending to the entire input, it first computes cross-attention
 157 between the input and a much smaller set of latent vectors (which are randomly initialized and
 158 trained). These latents are encoded with self-attention layers, and for the final output, the latents
 159 are again cross-attended with the input to match the input-size. See Appendix Figure 6 for an
 160 illustration. By default, we use 2048 latents of dimension $512 : \mathbb{R}^{2048 \times 512}$, but in Appendix G we
 161 experiment with different latent sizes.

162 The Perceiver Transformer uses 6 self-attention layers to encode the latents and outputs a sequence
 163 of patch encodings from the output cross-attention layer. These patch encodings are upsampled
 164 with a 3D convolution layer and tri-linear upsampling to decode 64-dimensional voxel features. The
 165 decoder includes a skip-connection from the encoder (like in UNets [67]). The per-voxel features
 166 are then used to predict discretized actions [14]. For translation, the voxel features are reshaped into
 167 the original voxel grid (100^3) to form a 3D Q -function of action-values. For rotation, gripper open,
 168 and collide, the features are max-pooled and then decoded with linear layers to form their respective
 169 Q -function. The best action \mathcal{T} is chosen by simply maximizing the Q -functions:

$$\begin{aligned}\mathcal{T}_{\text{trans}} &= \underset{(x,y,z)}{\operatorname{argmax}} \mathcal{Q}_{\text{trans}}((x,y,z) | \mathbf{v}, \mathbf{l}), & \mathcal{T}_{\text{rot}} &= \underset{(\psi,\theta,\phi)}{\operatorname{argmax}} \mathcal{Q}_{\text{rot}}((\psi,\theta,\phi) | \mathbf{v}, \mathbf{l}), \\ \mathcal{T}_{\text{open}} &= \underset{\omega}{\operatorname{argmax}} \mathcal{Q}_{\text{open}}(\omega | \mathbf{v}, \mathbf{l}), & \mathcal{T}_{\text{collide}} &= \underset{\kappa}{\operatorname{argmax}} \mathcal{Q}_{\text{collide}}(\kappa | \mathbf{v}, \mathbf{l}),\end{aligned}$$

170 where (x, y, z) is the voxel location in the grid, (ψ, θ, ϕ) are discrete rotations in Euler angles, ω is
 171 the gripper open state and κ is the collide variable. See Figure 5 for examples of Q -predictions.

172 3.4 Training Details

173 PERACT is trained through supervised learning with discrete-time input-action tuples from a dataset
 174 of demonstrations. These tuples are composed of voxel observations, language goals, and keyframe
 175 actions $\{(\mathbf{v}_1, \mathbf{l}_1, \mathbf{k}_1), (\mathbf{v}_2, \mathbf{l}_2, \mathbf{k}_2), \dots\}$. During training, we randomly sample a tuple and supervise
 176 the agent to predict the keyframe action \mathbf{k} given the observation and goal (\mathbf{v}, \mathbf{l}) . For translations,
 177 the ground-truth action is represented as a one-hot voxel encoding $Y_{\text{trans}} : \mathbb{R}^{H \times W \times D}$. Rotations are
 178 also represented with a one-hot encoding per rotation axis with R rotation bins $Y_{\text{rot}} : \mathbb{R}^{(360/R) \times 3}$
 179 ($R = 5$ degrees for all experiments). Similarly, open and collide variables are binary one-hot vectors
 180 $Y_{\text{open}} : \mathbb{R}^2$, $Y_{\text{collide}} : \mathbb{R}^2$. The agent is trained with cross-entropy loss like a classifier:

$$\mathcal{L}_{\text{total}} = -\mathbb{E}_{Y_{\text{trans}}}[\log \mathcal{V}_{\text{trans}}] - \mathbb{E}_{Y_{\text{rot}}}[\log \mathcal{V}_{\text{rot}}] - \mathbb{E}_{Y_{\text{open}}}[\log \mathcal{V}_{\text{open}}] - \mathbb{E}_{Y_{\text{collide}}}[\log \mathcal{V}_{\text{collide}}],$$

181 where $\mathcal{V}_{\text{trans}} = \text{softmax}(\mathcal{Q}_{\text{trans}}((x, y, z) | \mathbf{v}, \mathbf{l}))$, $\mathcal{V}_{\text{rot}} = \text{softmax}(\mathcal{Q}_{\text{rot}}((\psi, \theta, \phi) | \mathbf{v}, \mathbf{l}))$, $\mathcal{V}_{\text{open}} =$
 182 $\text{softmax}(\mathcal{Q}_{\text{open}}(\omega | \mathbf{v}, \mathbf{l}))$, $\mathcal{V}_{\text{collide}} = \text{softmax}(\mathcal{Q}_{\text{collide}}(\kappa | \mathbf{v}, \mathbf{l}))$ respectively. For robustness, we also
 183 augment \mathbf{v} and \mathbf{k} with translation and rotation perturbations. See Appendix E for more details.

184 By default, we use a voxel grid size of 100^3 . We conducted validation tests by replaying expert
 185 demonstrations with discretized actions to ensure that 100^3 is a sufficient resolution for execution.
 186 The agent was trained with a batch-size of 16 on 8 NVIDIA V100 GPUs for 16 days (600K iterations). We use the LAMB [68] optimizer following Perceiver [1].

188 For multi-task training, we simply sample input-action tuples from all tasks in the dataset. To ensure
 189 that tasks with longer horizons are not over-represented during sampling, each batch contains a
 190 uniform distribution of tasks. That is, we first uniformly sample a set of tasks of batch-size length,
 191 then pick a random input-action tuple for each of the sampled tasks. With this strategy, longer-
 192 horizon tasks need more training steps for full coverage of input-action pairs, but all tasks are given
 193 equal weighting during gradient updates.

194 **4 Results**

195 We perform experiments to answer the following questions: (1) How effective is PERACT com-
196 pared to unstructured image-to-action frameworks and standard architectures like 3D ConvNets?
197 And what are the factors that affect PERACT’s performance? (2) Is the global receptive field of
198 Transformers actually beneficial over methods with local receptive fields? (3) Can PERACT be
199 trained on real-world tasks with noisy data?

200 **4.1 Simulation Setup**

201 We conduct our primary experiments in simulation for the sake of reproducibility and benchmarking.

202 **Environment.** The simulation is set in CoppeliaSim [69] and interfaced through PyRep [70]. All
203 experiments use a Franka Panda robot with a parallel gripper. The input observations are captured
204 from four RGB-D cameras positioned at the front, left shoulder, right shoulder, and on the wrist, as
205 shown in Appendix Figure 7. All cameras are noiseless and have a resolution of 128×128 .

206 **Language-Conditioned Tasks.** We train and evaluate on 18 RLbench [15] tasks. See peract.github.io for examples and Appendix A for details on individual tasks. Each task also includes
207 several variations, ranging from 2-60 possibilities, e.g., in the `stack blocks` task, “*stack 2 red*
208 *blocks*” and “*stack 4 purple blocks*” are two variants. These variants are randomly sampled during
209 data generation, but kept consistent during evaluations for one-to-one comparisons. Some RL-
210 Bench tasks were modified to include additional variations to stress-test multi-task and language-
211 grounding capabilities. There are a total of 249 variations across 18 tasks, and the number of ex-
212 tracted keyframes range from 2-17. All keyframes from an episode have the same language goal,
213 which is constructed from templates (but human-annotated for real-world tasks). Note that in all
214 experiments, we do not test for generalization to unseen objects, i.e., our train and test objects are
215 the same. However during test time, the agent has to handle novel object poses, randomly sampled
216 goals, from randomly sampled scenes with different semantic instantiations of object color, shape,
217 size, and category. The focus here is to evaluate the effectiveness of a single multi-task agent trained
218 on all tasks and variants.

220 **Evaluation Metric.** Each multi-task agent is evaluated independently on all 18 tasks. Evaluations
221 are scored either 0 for failures or 100 for complete successes. There are no partial credits. We report
222 average success rates on 25 evaluation episodes per task ($25 \times 18 = 450$ total episodes) for agents
223 trained with $n = 10,100$ demonstrations per task. During evaluation, an agent keeps taking actions
224 until an oracle indicates task-completion or reaches a maximum of 25 steps.

225 **4.2 Simulation Results**

226 Table 1 reports success rates of multi-task agents trained on all 18 tasks. We could not investigate
227 single-task agents due to resource constraints of training 18 individual agents.

228 **Baseline Methods.** We study the effectiveness of our problem formulation by benchmarking against
229 two language-conditioned baselines: Image-BC and C2FARM-BC. Image-BC is an image-to-action
230 agent similar to BC-Z [12]. Following BC-Z, we use FiLM [71] for conditioning with CLIP [66]
231 language features, but the vision encoders take in RGB-D images instead of just RGB. We also study
232 both CNN and ViT vision encoders. C2FARM-BC is a 3D fully-convolutional network by James et
233 al. [14] that has achieved state-of-the-art results on RLbench tasks. Similar to our agent, C2FARM-
234 BC also detects actions in a voxelized space, however it uses a coarse-to-fine-grain scheme to detect
235 actions at two-levels of voxelization: 32^3 voxels with a 1^3m grid, and 32^3 voxels with a 0.15^3m
236 grid after “zooming in” from the first level. Note that at the finest level, C2FARM-BC has a higher
237 resolution (0.47cm) than PERACT (1cm). We use the same 3D ConvNet architecture as James et
238 al. [14], but instead of training it with RL, we do BC with cross-entropy loss (from Section 3.4). We
239 also condition it with CLIP [66] language features at the bottleneck like in LingUNets [72, 16].

240 **Multi-Task Performance.** Table 1 compares the performance of Image-BC and C2FARM-
241 BC against PERACT. With insufficient demonstrations, Image-BC has near zero performance on
242 most tasks. Image-BC is disadvantaged with single-view observations and has to learn hand-eye

	open drawer		slide block		sweep to dustpan		meat off grill		turn tap		put in drawer		close jar		drag stick		stack blocks	
Method	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
Image-BC (CNN)	4	4	4	0	0	0	0	0	20	8	0	8	0	0	0	0	0	0
Image-BC (ViT)	16	0	8	0	8	0	0	0	24	16	0	0	0	0	0	0	0	0
C2FARM-BC	28	20	12	16	4	0	40	20	60	68	12	4	28	24	72	24	4	0
PERACT (w/o Lang)	20	28	8	12	20	16	40	48	36	60	16	16	16	12	48	60	0	0
PERACT	68	80	32	72	72	56	68	84	72	80	16	68	32	60	36	68	12	36
	screw bulb		put in safe		place wine		put in cupboard		sort shape		push buttons		insert peg		stack cups		place cups	
	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
Image-BC (CNN)	0	0	0	4	0	0	0	0	0	0	4	0	0	0	0	0	0	0
Image-BC (ViT)	0	0	0	0	4	0	4	0	0	0	16	0	0	0	0	0	0	0
C2FARM-BC	12	8	0	12	36	8	4	0	8	8	88	72	0	4	0	0	0	0
PERACT (w/o Lang)	0	24	8	20	8	20	0	0	0	0	60	68	4	0	0	0	0	0
PERACT	28	24	16	44	20	12	0	16	16	20	56	48	4	0	0	0	0	0

Table 1. Multi-Task Test Results. Success rates (mean %) of various multi-task agents tasks trained with either 10 or 100 demonstrations per task and evaluated on 25 episodes per task. Each evaluation episode is scored either a 0 for failure or 100 for success. PERACT outperforms C2FARM-BC [14], the most competitive baseline, with an average improvement of +168% with 10 demos and +266% with 100 demos.

coordination from scratch. In contrast, PERACT’s voxel-based formulation naturally allows for integrating multi-view observations, learning 6-DoF action representations, and data-augmentation in 3D, all of which are non-trivial to achieve in image-based methods. C2FARM-BC is the most competitive baseline, but it has a limited receptive field because of the coarse-to-fine-grain scheme and partly due to the convolution-only architecture. PERACT outperforms C2FARM-BC in 25/36 evaluations in Table 1 with **an average improvement of +168% with 10 demonstrations and +266% with 100 demonstration**. For a number of tasks, C2FARM-BC actually performs worse with more demonstrations, likely due to insufficient capacity. Since additional training demonstrations include additional task variants to optimize for, they might end up hurting performance.

In general, 10 demonstrations are sufficient for PERACT to achieve > 65% success on tasks with limited variations like open drawer (3 variations). But tasks with more variations like stack blocks (60 variations) need substantially more data, sometimes to simply cover all possible concepts like “teal color block” that might have not appeared in the training data. See the simulation rollouts in the supplementary video to get a sense of the complexity of these evaluations. For three tasks: insert peg, stack cups, and place cups, all agents achieve near zero success. These are very high-precision tasks where being off by a few centimeters or degrees could lead to unrecoverable failures. But in Appendix H, we find that training single-task agents specifically for these tasks slightly alleviates this issue.

Ablations. Table 1 reports PERACT w/o Lang, an agent without any language conditioning. Without a language goal, the agent does not know the underlying task and performs at chance. We also report additional ablation results on the open drawer task in Figure 3. To summarize these results: (1) the skip connection helps train the agent slightly faster, (2) the Perceiver Transformer is crucial for achieving good performance with the global receptive field, and (3) extracting good keyframes actions is essential for supervised training as randomly chosen or fixed-interval keyframes lead to zero-performance.

Sensitivity Analysis. In Appendix G we investigate factors that affect PERACT’s performance: the number of Perceiver latents, voxelization resolution, and data augmentation. We find that more latent vectors generally improve the capacity of the agent to model more tasks, but for simple short-horizon tasks, fewer latents are sufficient. Similarly, with different voxelization resolutions, some tasks are solvable with coarse voxel grids like 32^3 , but some high-precision tasks require the full 100^3 grid. Finally, rotation perturbations in the data augmentation generally help in improving robustness essentially by exposing the agent to more rotation variations of objects.

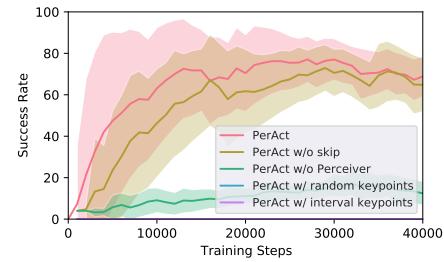


Figure 3. Ablation Experiments. Success rate of PERACT after ablating key components.

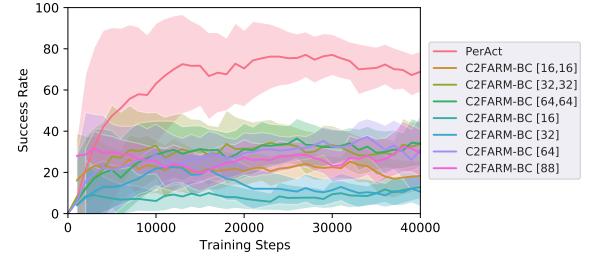


Figure 4. Global vs. Local Receptive Field Experiments. Success rates of PERACT against various C2FARM-BC [14] baselines

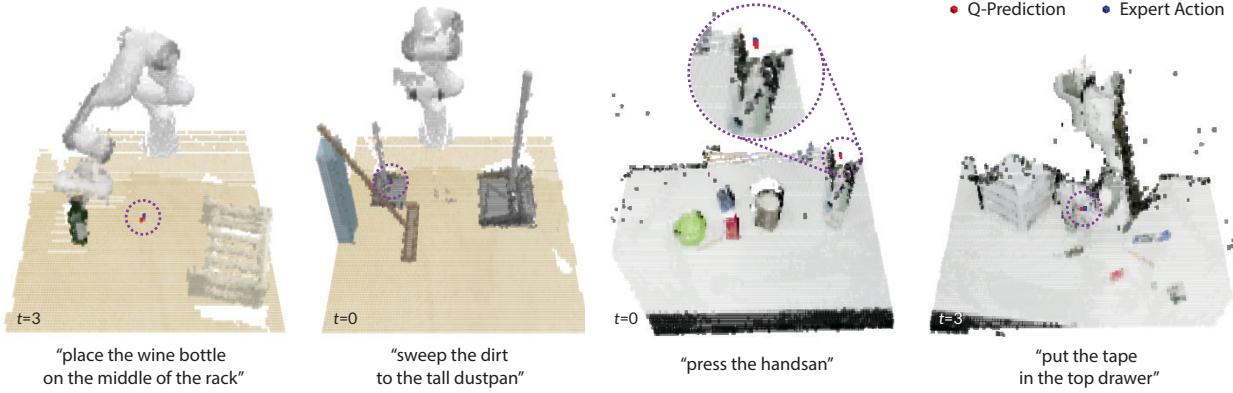


Figure 5. Q-Prediction Examples: Qualitative examples of translation Q -Predictions from PERACT along with expert actions, highlighted with dotted-circles. The left two are simulated tasks, and the right two are real-world tasks. See Appendix J for more examples.

284 4.3 Global vs. Local Receptive Fields

285 To further investigate our Transformer agent’s global receptive field, we conduct additional experiments on the `open drawer` task. The `open drawer` task has three variants: “*open the top drawer*”,
286 “*open the middle drawer*”, and “*open the bottom drawer*”, and with a limited receptive field it is
287 hard to distinguish the drawer handles, which are all visually identical. Figure 4 reports PER-
288 ACT and C2FARM-BC agents trained with 100 demonstrations. Although the `open drawer` tasks
289 can be solved with fewer demonstrations, here we want to ensure that insufficient data is not an is-
290 sue. We include several versions of C2FARM-BC with different voxelization schemes. For instance,
291 [16, 16] indicates two levels of 16^3 voxel grids at 1m^3 and 0.15m^3 , respectively. And [64] indicates
292 a single level of a 64^3 voxel grid without the coarse-to-fine-grain scheme. PERACT is the only agent
293 that achieves $> 70\%$ success, whereas all C2FARM-BC versions perform at chance with $\sim 33\%$,
294 indicating that the global receptive field of the Transformer is crucial for solving the task.
295

296 4.4 Real-Robot Results

297 We also validated our results with real-robot experiments on a
298 Franka Emika Panda. See Appendix D for setup details. Without
299 any sim-to-real transfer or pre-training, we trained a multi-task
300 PERACT agent *from scratch* on 7 tasks (with 18 unique variations)
301 from a total of just 53 demonstrations. See the supplementary video
302 for qualitative results that showcase the diversity of tasks and ro-
303 bustness to scene changes. Table 2 reports success rates from small-
304 scale evaluations. Similar to the simulation results, we find that
305 PERACT is able to achieve $> 65\%$ success on simple short-horizon
306 tasks like pressing hand-sanitizers from just a handful number of demon-
307 strations. The most common failures involved predicting incorrect gripper open actions, which often lead the agent into un-
308 seen states. This could be addressed in future works by using HG-Dagger style approaches to correct the
309 agent [12]. Other issues included the agent exploiting biases in the dataset like in prior work [16].
310 This could be addressed by scaling up expert data with more diverse tasks and task variants.

311 5 Conclusion and Limitations

312 We presented PERACT, a Transformer-based multi-task agent for 6-DoF manipulation. Our ex-
313 periments with both simulated and real-world tasks indicate that the right problem formulation, i.e., de-
314 tecting voxel actions, makes a substantial difference in terms of data efficiency and robustness.
315 While PERACT is quite capable, extending it to dexterous continuous control remains a challenge.
316 PERACT is at the mercy of a sampling-based motion-planner to execute discretized actions, and is
317 not easily extendable to N-DoF actuators like multi-fingered hands. See Appendix L for an extended
318 discussion on PERACT’s limitations. But overall, we are excited about scaling up robot learning with
319 Transformers by focusing on *diverse* rather than narrow multi-task data for robotic manipulation.

Task	# Train	# Test	Succ. %
Press Handsan	5	10	90
Put Marker	8	10	70
Place Food	8	10	60
Put in Drawer	8	10	40
Hit Ball	8	10	60
Stack Blocks	10	10	40
Sweep Beans	8	5	20

Table 2. Success rates (mean %) of a multi-task model trained and evaluated 7 real-world tasks (see Figure 1).

320 **References**

- 321 [1] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran,
322 A. Brock, E. Shelhamer, et al. Perceiver io: A general architecture for structured inputs &
323 outputs. *arXiv preprint arXiv:2107.14795*, 2021.
- 324 [2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin.
325 Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- 327 [3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan,
328 P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Neural Information
329 Processing Systems (NeurIPS)*, 2020.
- 330 [4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. De-
331 hghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transfor-
332 mers for image recognition at scale. In *International Conference on Learning Representations
333 (ICLR)*, 2020.
- 334 [5] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool,
335 R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with
336 alphafold. *Nature*, 2021.
- 337 [6] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik,
338 A. Huang, P. Georgiev, R. Powell, et al. Alphastar: Mastering the real-time strategy game
339 starcraft ii. *DeepMind blog*, 2, 2019.
- 340 [7] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. Hinton. Pix2seq: A language modeling framework
341 for object detection. *arXiv preprint arXiv:2109.10852*, 2021.
- 342 [8] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and
343 I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Neural
344 Information Processing Systems (NeurIPS)*, 2021.
- 345 [9] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez,
346 Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint
347 arXiv:2205.06175*, 2022.
- 348 [10] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional
349 transformers for language understanding. In *Conference of the North American Chapter of the
350 Association for Computational Linguistics (NAACL)*, 2018.
- 351 [11] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira. Perceiver: General
352 perception with iterative attention. In *International Conference on Machine Learning (ICML)*,
353 2021.
- 354 [12] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z:
355 Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning
356 (CoRL)*, 2021.
- 357 [13] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan,
358 K. Hausman, A. Herzog, et al. Do as i can, not as i say: Grounding language in robotic
359 affordances. *arXiv preprint arXiv:2204.01691*, 2022.
- 360 [14] S. James, K. Wada, T. Laidlow, and A. J. Davison. Coarse-to-fine q-attention: Efficient learning
361 for visual robotic manipulation via discretisation. In *Computer Vision and Pattern Recognition
362 (CVPR)*, 2022.
- 363 [15] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark &
364 learning environment. *IEEE Robotics and Automation Letters (RA-L)*, 2020.

- 365 [16] M. Shridhar, L. Manuelli, and D. Fox. Cliprot: What and where pathways for robotic manipulation. In *In Conference on Robot Learning (CoRL)*, 2021.
- 366
- 367 [17] J. J. Gibson. *The ecological approach to visual perception: classic edition*. Psychology Press, 368 2014.
- 369 [18] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Conference on Computer Vision* 370 and *Pattern Recognition (CVPR)*, 2017.
- 371 [19] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 372 6d object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018.
- 373 [20] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and 374 K. Daniilidis. Single image 3d object detection and pose estimation for grasping. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.
- 375
- 376 [21] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao. Multi-view self- 377 supervised deep learning for 6d pose estimation in the amazon picking challenge. In *2017 IEEE international conference on robotics and automation (ICRA)*, 2017.
- 378
- 379 [22] X. Deng, Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox. Self-supervised 6d object 380 pose estimation for robot manipulation. In *2020 IEEE International Conference on Robotics* 381 and *Automation (ICRA)*, 2020.
- 382
- 383 [23] C. Xie, Y. Xiang, A. Mousavian, and D. Fox. The best of both modes: Separately leveraging 384 rgb and depth for unseen object instance segmentation. In *Conference on Robot Learning* 385 (*CoRL*), 2020.
- 386
- 387 [24] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, 388 D. Duong, V. Sindhwani, and J. Lee. Transporter networks: Rearranging the visual world for 389 robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.
- 390
- 391 [25] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, 392 M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision- 393 based robotic manipulation. *Conference on Robot Learning (CoRL)*, 2018.
- 394
- 395 [26] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel. Learning to Manipulate Deformable 396 Objects without Demonstrations. In *Robotics: Science and Systems (RSS)*, 2020.
- 397
- 398 [27] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. 399 *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.
- 400
- 401 [28] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.
- 402
- 403 [29] S. Song, A. Zeng, J. Lee, and T. Funkhouser. Grasping in the wild: Learning 6dof closed-loop 404 grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters (RA-L)*, 2020.
- 405
- 406 [30] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox. 6-dof grasping for target- 407 driven object manipulation in clutter. In *International Conference on Robotics and Automation* 408 (*ICRA*), 2020.
- 409
- 410 [31] A. Mousavian, C. Eppner, and D. Fox. 6-dof grapsnet: Variational grasp generation for object 411 manipulation. In *International Conference on Computer Vision (ICCV)*, 2019.
- 412
- 413 [32] Z. Xu, H. Zhanpeng, and S. Song. Umpnet: Universal manipulation policy network for articu- 414 lated objects. *Robotics and Automation Letters (RA-L)*, 2022.
- 415
- 416 [33] A. Simeonov, Y. Du, A. Tagliasacchi, J. B. Tenenbaum, A. Rodriguez, P. Agrawal, and V. Sitz- 417 mann. Neural descriptor fields: Se (3)-equivariant object representations for manipulation. 418 *arXiv preprint arXiv:2112.05124*, 2021.
- 419

- 409 [34] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation
410 for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.
- 411 [35] W. Yuan, C. Paxton, K. Desingh, and D. Fox. Sornet: Spatial object-centric representations for
412 sequential manipulation. In *In Conference on Robot Learning (CoRL)*. PMLR, 2021.
- 413 [36] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer,
414 and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint
415 arXiv:1907.11692*, 2019.
- 416 [37] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer:
417 Hierarchical vision transformer using shifted windows. In *In International Conference on
418 Computer Vision (ICCV)*, 2021.
- 419 [38] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling
420 problem. *Neural information processing systems (NeurIPS)*, 2021.
- 421 [39] K.-H. Lee, O. Nachum, M. Yang, L. Lee, D. Freeman, W. Xu, S. Guadarrama, I. Fischer,
422 E. Jang, H. Michalewski, et al. Multi-game decision transformers. *arXiv preprint
423 arXiv:2205.15241*, 2022.
- 424 [40] H. M. Clever, A. Handa, H. Mazhar, K. Parker, O. Shapira, Q. Wan, Y. Narang, I. Akinola,
425 M. Cakmak, and D. Fox. Assistive tele-op: Leveraging transformers to collect robotic task
426 demonstrations. *arXiv preprint arXiv:2112.05129*, 2021.
- 427 [41] R. Yang, M. Zhang, N. Hansen, H. Xu, and X. Wang. Learning vision-guided quadrupedal lo-
428 comotion end-to-end with cross-modal transformers. *arXiv preprint arXiv:2107.03996*, 2021.
- 429 [42] D. S. Chaplot, D. Pathak, and J. Malik. Differentiable spatial planning using transformers. In
430 *International Conference on Machine Learning (ICML)*, 2021.
- 431 [43] J. J. Johnson, L. Li, A. H. Qureshi, and M. C. Yip. Motion planning transformers: One model
432 to plan them all. *arXiv preprint arXiv:2106.02791*, 2021.
- 433 [44] S. Dasari and A. Gupta. Transformers for one-shot visual imitation. *arXiv preprint
434 arXiv:2011.05970*, 2020.
- 435 [45] H. Kim, Y. Ohmura, and Y. Kuniyoshi. Transformer-based deep imitation learning for dual-arm
436 robot manipulation. In *International Conference on Intelligent Robots and Systems (IROS)*.
437 IEEE, 2021.
- 438 [46] Y. Han, R. Batra, N. Boyd, T. Zhao, Y. She, S. Hutchinson, and Y. Zhao. Learning generalizable
439 vision-tactile robotic grasping strategy for deformable objects via transformer. *arXiv preprint
440 arXiv:2112.06374*, 2021.
- 441 [47] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A
442 benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on
443 Robot Learning (CoRL)*, 2020.
- 444 [48] M. Shridhar and D. Hsu. Interactive visual grounding of referring expressions for human-robot
445 interaction. In *Robotics: Science and Systems (RSS)*, 2018.
- 446 [49] C. Matuszek, L. Bo, L. Zettlemoyer, and D. Fox. Learning from unscripted deictic gesture
447 and language for human-robot interactions. In *AAAI Conference on Artificial Intelligence*,
448 volume 28, 2014.
- 449 [50] M. Bollini, S. Tellex, T. Thompson, N. Roy, and D. Rus. Interpreting and executing recipes
450 with a cooking robot. In *Experimental Robotics*, pages 481–495. Springer, 2013.

- 451 [51] D. K. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of
452 natural language to manipulation instructions. *The International Journal of Robotics Research*
453 (*IJRR*), 2016.
- 454 [52] Y. Bisk, D. Yuret, and D. Marcu. Natural language communication with robots. In *North*
455 *American Chapter of the Association for Computational Linguistics (NAACL)*, 2016.
- 456 [53] J. Thomason, S. Zhang, R. J. Mooney, and P. Stone. Learning to interpret natural language
457 commands through human-robot dialog. In *Twenty-Fourth International Joint Conference on*
458 *Artificial Intelligence (IJCAI)*, 2015.
- 459 [54] J. Hatori, Y. Kikuchi, S. Kobayashi, K. Takahashi, Y. Tsuboi, Y. Unno, W. Ko, and J. Tan.
460 Interactively picking real-world objects with unconstrained spoken language instructions. In
461 *International Conference on Robotics and Automation (ICRA)*, 2018.
- 462 [55] Y. Chen, R. Xu, Y. Lin, and P. A. Vela. A Joint Network for Grasp Detection Conditioned on
463 Natural Language Commands. *arXiv:2104.00492 [cs]*, Apr. 2021.
- 464 [56] V. Blukis, R. A. Knepper, and Y. Artzi. Few-shot object grounding for mapping natural lan-
465 guage instructions to robot control. In *Conference on Robot Learning (CoRL)*, 2020.
- 466 [57] C. Paxton, Y. Bisk, J. Thomason, A. Byravan, and D. Fox. Prospection: Interpretable plans
467 from language by predicting the future. In *International Conference on Robotics and Automa-*
468 *tion (ICRA)*, 2019.
- 469 [58] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy. Under-
470 standing natural language commands for robotic navigation and mobile manipulation. In *AAAI*
471 *Conference on Artificial Intelligence (AAAI)*, 2011.
- 472 [59] C. Lynch and P. Sermanet. Grounding language in play. *arXiv preprint arXiv:2005.07648*,
473 2020.
- 474 [60] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. Calvin: A benchmark for language-
475 conditioned policy learning for long-horizon robot manipulation tasks. *arXiv preprint*
476 *arXiv:2112.03227*, 2021.
- 477 [61] E. Johns. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration.
478 In *International Conference on Robotics and Automation (ICRA)*, 2021.
- 479 [62] S. James and A. J. Davison. Q-attention: Enabling efficient learning for vision-based robotic
480 manipulation. *IEEE Robotics and Automation Letters (RA-L)*, 7(2):1612–1619, 2022.
- 481 [63] S. Liu, S. James, A. J. Davison, and E. Johns. Auto-lambda: Disentangling dynamic task
482 relationships. *Transactions on Machine Learning Research*, 2022.
- 483 [64] H. Moravec. Robot spatial perceptionby stereoscopic vision and 3d evidence grids. *Perception*,
484 1996.
- 485 [65] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Com-*
486 *puter*, 22(6):85–90, 1989.
- 487 [66] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell,
488 P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning Transferable Visual Models From
489 Natural Language Supervision. *arXiv:2103.00020*, 2021.
- 490 [67] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical im-
491 age segmentation. In *International Conference on Medical image computing and computer-*
492 *assisted intervention*, pages 234–241. Springer, 2015.

- 493 [68] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer,
 494 and C.-J. Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv*
 495 *preprint arXiv:1904.00962*, 2019.
- 496 [69] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation
 497 framework. In *International Conference on Intelligent Robots and Systems (IROS)*, 2013.
- 498 [70] S. James, M. Freese, and A. J. Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv*
 499 *preprint arXiv:1906.11176*, 2019.
- 500 [71] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a
 501 general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 2018.
- 502 [72] D. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkhin, and Y. Artzi. Mapping instructions
 503 to actions in 3d environments with visual goal prediction. In *2019 Conference on Empirical*
 504 *Methods in Natural Language Processing (EMNLP)*, 2018.
- 505 [73] Z. Mandi, P. Abbeel, and S. James. On the effectiveness of fine-tuning versus meta-
 506 reinforcement learning. *arXiv preprint arXiv:2206.03271*, 2022.
- 507 [74] S. Sodhani, A. Zhang, and J. Pineau. Multi-task reinforcement learning with context-based
 508 representations. In M. Meila and T. Zhang, editors, *International Conference on Machine*
 509 *Learning (ICML)*, 2021.
- 510 [75] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht. ALFWorld:
 511 Aligning Text and Embodied Environments for Interactive Learning. In *International Confer-*
 512 *ence on Learning Representations (ICLR)*, 2021.
- 513 [76] A. Zeng, A. Wong, S. Welker, K. Choromanski, F. Tombari, A. Purohit, M. Ryoo, V. Sind-
 514 hwani, J. Lee, V. Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reason-
 515 ing with language. *arXiv preprint arXiv:2204.00598*, 2022.
- 516 [77] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Ex-
 517 tracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.
- 518 [78] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a mul-
 519 tiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 2022.
- 520 [79] Sara Fridovich-Keil and Alex Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels:
 521 Radiance fields without neural networks. In *CVPR*, 2022.
- 522 [80] V. Blukis, C. Paxton, D. Fox, A. Garg, and Y. Artzi. A persistent spatial semantic representation
 523 for high-level natural language instruction execution. In *Conference on Robot Learning*, pages
 524 706–717. PMLR, 2022.
- 525 [81] R. Corona, S. Zhu, D. Klein, and T. Darrell. Voxel-informed language grounding. In *Associa-*
 526 *tion for Computational Linguistics (ACL)*, 2022.
- 527 [82] S. Lal, M. Prabhudesai, I. Mediratta, A. W. Harley, and K. Fragkiadaki. Coconets: Contin-
 528 uous contrastive 3d scene representations. In *Proceedings of the IEEE/CVF Conference on*
 529 *Computer Vision and Pattern Recognition*, 2021.
- 530 [83] H.-Y. F. Tung, Z. Xian, M. Prabhudesai, S. Lal, and K. Fragkiadaki. 3d-oes: Viewpoint-
 531 invariant object-factorized environment simulators. *arXiv preprint arXiv:2011.06464*, 2020.
- 532 [84] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale
 533 using voxel hashing. *ACM Transactions on Graphics (ToG)*, 2013.
- 534 [85] R. Ranftl, A. Bochkovskiy, and V. Koltun. Vision transformers for dense prediction. In *Inter-*
 535 *national Conference on Computer Vision (ICCV)*, pages 12179–12188, 2021.

- 536 [86] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint*
537 *arXiv:1412.6980*, 2014.
- 538 [87] S. James and P. Abbeel. Coarse-to-fine q-attention with learned path ranking. *arXiv preprint*
539 *arXiv:2204.01571*, 2022.
- 540 [88] A. Kamath, M. Singh, Y. LeCun, I. Misra, G. Synnaeve, and N. Carion. Mdetr-modulated
541 detection for end-to-end multi-modal understanding. *arXiv preprint arXiv:2104.12763*, 2021.
- 542 [89] A. Birhane, V. U. Prabhu, and E. Kahembwe. Multimodal datasets: misogyny, pornography,
543 and malignant stereotypes. *arXiv preprint arXiv:2110.01963*, 2021.
- 544 [90] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic
545 parrots: Can language models be too big? In *2021 ACM Conference on Fairness, Accountabil-*
546 *ity, and Transparency*, pages 610–623, 2021.
- 547 [91] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. A tutorial on energy-based
548 learning. *Predicting structured data*, 1(0), 2006.
- 549 [92] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mor-
550 datch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning (CoRL)*,
551 2022.

Task	Variation Type	# of Variations	Avg. Keyframes	Language Template
open drawer	placement	3	3.0	“open the ___ drawer”
slide block	color	4	4.7	“slide the block to the ___ target”
sweep to dustpan	size	2	4.6	“sweep dirt to the ___ dustpan”
meat off grill	category	2	5.0	“take the ___ off the grill”
turn tap	placement	2	2.0	“turn the ___ tap”
put in drawer	placement	3	12.0	“put the item in the ___ drawer”
close jar	color	20	6.0	“close the ___ jar”
drag stick	color	20	6.0	“use the stick to drag the cube onto the ___ target”
stack blocks	color, count	60	14.6	“stack ___ blocks”
screw bulb	color	20	7.0	“screw in the ___ light bulb”
put in safe	placement	3	5.0	“put the money away in the safe on the ___ shelf”
place wine	placement	3	5.0	“stack the wine bottle to the ___ of the rack”
put in cupboard	category	9	5.0	“put the ___ in the cupboard”
sort shape	shape	5	5.0	“put the ___ in the shape sorter”
push buttons	color	50	3.8	“push the ___ button, [then the ___ button]”
insert peg	color	20	5.0	“put the ring on the ___ spoke”
stack cups	color	20	10.0	“stack the other cups on top of the ___ cup”
place cups	count	3	11.5	“place ___ cups on the cup holder”

Table 3. Language-Conditioned Tasks in RLBench [15].

552 A Task Details

553 **Setup.** Our simulated experiments are set in RLBench [15]. We select 18 out of 100 tasks that
 554 involve at least two or more variations to evaluate the multi-task capabilities of agents. While PER-
 555 ACT could be easily applied to more RLBench tasks, in our experiments, we were specifically
 556 interested grounding diverse language instructions, rather than learning one-off policies for single-
 557 variation tasks like “[always] take off the saucepan lid”. Some tasks were modified to include
 558 additional variations. See Table 3 for an overview. We report average keyframes extracted from the
 559 method described in Section 3.2.

560 **Variations.** Task variations include randomly sampled colors, sizes, shapes, counts, placements, and
 561 categories of objects. The set of colors include 20 instances: `colors = {red, maroon, lime, green,`
 562 `blue, navy, yellow, cyan, magenta, silver, gray, orange, olive, purple, teal, azure,`
 563 `violet, rose, black, white}`. The set of sizes include 2 instances: `sizes = {short, tall}`. The set of
 564 shapes include 5 instances: `shapes = {cube, cylinder, triangle, star, moon}`. The set of
 565 counts include 3 instances: `counts = {1, 2, 3}`. The placements and object categories are specific
 566 to each task. For instance, `open drawer` has 3 placement locations: `top, middle, and bottom`,
 567 and `put in cupboard` includes 9 YCB objects. In addition to these semantic variations, objects
 568 are placed on the tabletop at random poses. Some large objects like drawers have constrained pose
 569 variations [15] to ensure that manipulating them is kinematically feasible with the Franka arm.

570 In the following sections, we describe each of 18 tasks in detail. We highlight tasks that were
 571 modified from the original RLBench [15] codebase³ and describe what exactly was modified.

572 A.1 Open Drawer

573 **Filename:** `open_drawer.py`

574 **Task:** Open one of the three drawers: `top, middle, or bottom`.

575 **Modified:** No.

576 **Objects:** 1 drawer.

577 **Success Metric:** The prismatic joint of the specified drawer is fully extended.

578 A.2 Slide Block

579 **Filename:** `slide_block_to_color_target.py`

³<https://github.com/stepjam/RLBench>

580 **Task:** Slide the block on to one of the colored square targets. The target colors are limited to `red`,
581 `blue`, `pink`, and `yellow`.

582 **Modified:** Yes. The original `slide_block_to_target.py` task contained only one target. Three
583 other targets were added to make a total of 4 variations.

584 **Objects:** 1 block and 4 colored target squares.

585 **Success Metric:** Some part of the block is inside the specified target area.

586 **A.3 Sweep to Dustpan**

587 **Filename:** `sweep_to_dustpan_of_size.py`

588 **Task:** Sweep the dirt particles to either the short or tall dustpan.

589 **Modified:** Yes. The original `sweep_to_dustpan.py` task contained only one dustpan. One other
590 dustpan was added to make a total of 2 variations.

591 **Objects:** 5 dirt particles and 2 dustpans.

592 **Success Metric:** All 5 dirt particles are inside the specified dustpan.

593 **A.4 Meat Off Grill**

594 **Filename:** `meat_off_grill.py`

595 **Task:** Take either the chicken or steak off the grill and put it on the side.

596 **Modified:** No.

597 **Objects:** 1 piece of chicken, 1 piece of steak, and 1 grill.

598 **Success Metric:** The specified meat is on the side, away from the grill.

599 **A.5 Turn Tap**

600 **Filename:** `turn_tap.py`

601 **Task:** Turn either the left or right handle of the tap. Left and right are defined with respect to the
602 faucet orientation.

603 **Modified:** No.

604 **Objects:** 1 faucet with 2 handles.

605 **Success Metric:** The revolute joint of the specified handle is at least 90° off from the starting
606 position.

607 **A.6 Put in Drawer**

608 **Filename:** `put_item_in_drawer.py`

609 **Task:** Put the block in one of the three drawers: `top`, `middle`, or `bottom`.

610 **Modified:** No.

611 **Objects:** 1 block and 1 drawer.

612 **Success Metric:** The block is inside the specified drawer.

613 **A.7 Close Jar**

614 **Filename:** `close_jar.py`

615 **Task:** Put the lid on the jar with the specified color and screw the lid in. The jar colors are sampled
616 from the full set of 20 color instances.

617 **Modified:** No.

618 **Objects:** 1 block and 2 colored jars.

619 **Success Metric:** The lid is on top of the specified jar and the Franka gripper is not grasping anything.

620 A.8 Drag Stick

621 **Filename:** reach_and_drag.py

622 **Task:** Grab the stick and use it to drag the cube on to the specified colored target square. The target
623 colors are sampled from the full set of 20 color instances.

624 **Modified:** Yes. The original reach_and_drag.py task contained only one target. Three other
625 targets were added with randomized colors.

626 **Objects:** 1 block, 1 stick, and 4 colored target squares.

627 **Success Metric:** Some part of the block is inside the specified target area.

628 A.9 Stack Blocks

629 **Filename:** stack_blocks.py

630 **Task:** Stack N blocks of the specified color on the green platform. There are always 4 blocks of the
631 specified color, and 4 distractor blocks of another color. The block colors are sampled from the full
632 set of 20 color instances.

633 **Modified:** No.

634 **Objects:** 8 color blocks (4 are distractors), and 1 green platform.

635 **Success Metric:** N blocks are inside the area of the green platform.

636 A.10 Screw Bulb

637 **Filename:** light_bulb.in.py

638 **Task:** Pick up the light bulb from the specified holder, and screw it into the lamp stand. The colors
639 of holder are sampled from the full set of 20 color instances. There are always two holders in the
640 scene – one specified and one distractor holder.

641 **Modified:** No.

642 **Objects:** 2 light bulbs, 2 holders, and 1 lamp stand.

643 **Success Metric:** The bulb from the specified holder is inside the lamp stand dock.

644 A.11 Put in Safe

645 **Filename:** put_money_in_safe.py

646 **Task:** Pick up the stack of money and put it inside the safe on the specified shelf. The shelf has
647 three placement locations: top, middle, bottom.

648 **Modified:** No.

649 **Objects:** 1 stack of money, and 1 safe.

650 **Success Metric:** The stack of money is on the specified shelf inside the safe.

651 **A.12 Place Wine**

652 **Filename:** place_wine_at_rack_location.py

653 **Task:** Grab the wine bottle and put it on the wooden rack at one of the three specified locations:
654 `left`, `middle`, `right`. The locations are defined with respect to the orientation of the wooden rack.

655 **Modified:** Yes. The original `stack_wine.py` task had only one placement location. Two other
656 locations were added to make a total of 3 variations.

657 **Objects:** 1 wine bottle, and 1 wooden rack.

658 **Success Metric:** The wine bottle is at the specified placement location on the wooden rack.

659 **A.13 Put in Cupboard**

660 **Filename:** put_groceries_in_cupboard.py

661 **Task:** Grab the specified object and put it in the cupboard above. The scene always contains 9 YCB
662 objects that are randomly placed on the tabletop.

663 **Modified:** No.

664 **Objects:** 9 YCB objects, and 1 cupboard (that hovers in the air like magic).

665 **Success Metric:** The specified object is inside the cupboard.

666 **A.14 Sort Shape**

667 **Filename:** place_shape_in_shape_sorter.py

668 **Task:** Pick up the specified shape and place it inside the correct hole in the sorter. There are always
669 4 distractor shapes, and 1 correct shape in the scene.

670 **Modified:** Yes. The sizes of the shapes and sorter were enlarged so that they are distinguishable in
671 the RGB-D input.

672 **Objects:** 5 shapes, and 1 sorter.

673 **Success Metric:** The specified shape is inside the sorter.

674 **A.15 Push Buttons**

675 **Filename:** push_buttons.py

676 **Task:** Push the colored buttons in the specified sequence. The button colors are sampled from the
677 full set of 20 color instances. There are always three buttons in scene.

678 **Modified:** No.

679 **Objects:** 3 buttons.

680 **Success Metric:** All the specified buttons were pressed.

681 **A.16 Insert Peg**

682 **Filename:** insert_onto_square_peg.py

683 **Task:** Pick up the square and put it on the specified color spoke. The spoke colors are sampled from
684 the full set of 20 color instances.

685 **Modified:** No.

686 **Objects:** 1 square, and 1 spoke platform with three color spokes.

687 **Success Metric:** The square is on the specified spoke.

688 **A.17 Stack Cups**

689 **Filename:** stack_cups.py

690 **Task:** Stack all cups on top of the specified color cup. The cup colors are sampled from the full set
691 of 20 color instances. The scene always contains three cups.

692 **Modified:** No.

693 **Objects:** 3 tall cups.

694 **Success Metric:** All other cups are inside the specified cup.

695 **A.18 Place Cups**

696 **Filename:** place_cups.py

697 **Task:** Place N cups on the cup holder. This is a very high precision task where the handle of the
698 cup has to be exactly aligned with the spoke of the cup holder for the placement to succeed.

699 **Modified:** No.

700 **Objects:** 3 cups with handles, and 1 cup holder with three spokes.

701 **Success Metric:** N cups are on the cup holder, each on a separate spoke.

702 **B PERACT Details**

703 In this section, we provide implementation details for PERACT.

704 **Input Observation.** Following James et al. [14], our input voxel observation is a 100^3 voxel grid
705 with 10 channels: $\mathbb{R}^{100 \times 100 \times 100 \times 10}$. The grid is constructed by fusing calibrated pointclouds with
706 PyTorch’s `scatter_` function⁴. The 10 channels are composed of: 3 RGB, 3 point, 1 occupancy,
707 and 3 position index values. The RGB values are normalized to a zero-mean distribution. The point
708 values are Cartesian coordinates in the robot’s coordinate frame. The occupancy value indicates if
709 a voxel is occupied or empty. The position index values represent the 3D location of the voxel with
710 respect to the 100^3 grid. In addition to the voxel observation, the input also includes proprioception
711 data with 4 scalar values: gripper open, left finger joint position, right finger joint position, and
712 timestep (of the action sequence).

713 **Input Language.** The language goals are encoded with CLIP’s language encoder [66]. We use
714 CLIP’s tokenizer to preprocess the sentence, which always results in an input sequence of 77 tokens
715 (with zero-padding). These tokens are encoded with the language encoder to produce a sequence of
716 dimensions $\mathbb{R}^{77 \times 512}$.

717 **Preprocessing.** The voxel grid is encoded with a 3D convolution layer with a 1×1 kernel to
718 upsample the channel dimension from 10 to 64. Similarly, the proprioception data is encoded with
719 a linear layer to upsample the input dimension from 4 to 64. The encoded voxel grid is split into
720 5^3 patches through a 3D convolution layer with a kernel-size and stride of 5, which results in a
721 patch tensor of dimensions $\mathbb{R}^{20 \times 20 \times 20 \times 64}$. The proprioception features are tiled in 3D to match the
722 dimensions of the patch tensor, and concatenated along the channel to form a tensor of dimensions
723 $\mathbb{R}^{20 \times 20 \times 20 \times 128}$. This tensor is flattened into a sequence of dimensions $\mathbb{R}^{8000 \times 128}$. The language
724 features are downsampled with a linear layer from 512 to 128 dimensions, and then appended to the
725 tensor to form the final input sequence to the Perceiver Transformer, which of dimensions $\mathbb{R}^{8077 \times 128}$.
726 We also add learned positional embeddings to the input sequence. These embeddings are represented
727 with trainable `nn.Parameter(s)` in PyTorch.

728 **Perceiver Transformer** is a latent-space Transformer that uses a small set of latent vectors
729 to encode extremely long input sequences. See Figure 6 for an illustration of this process.

⁴https://pytorch.org/docs/stable/generated/torch.Tensor.scatter_.html

730 Perceiver first computes cross-
 731 attention between the input sequence
 732 and the set of latent vectors of
 733 dimensions $\mathbb{R}^{2048 \times 512}$. These latents
 734 are randomly initialized and trained
 735 end-to-end. The latents are encoded
 736 with 6 self-attention layers, and
 737 then cross-attended with the input
 738 to output a sequence that matches
 739 the input-dimensions. This output
 740 is upsampled with a 3D convolution
 741 layer and tri-linear upsampling to
 742 form a voxel feature grid with 64 channels:
 $\mathbb{R}^{100 \times 100 \times 100 \times 64}$. This feature grid is concatenated
 743 with the initial 64-dimensional feature grid from the processing stage as a skip connection to the
 744 encoding layers. Finally, a 3D convolution layer with a 1×1 kernel downsamples the channels from
 745 128 back to 64 dimensions. Our implementation of Perceiver is based on an existing open-source
 746 repository⁵.

747 **Decoding.** For translation, the voxel feature grid is decoded with a 3D convolution layer with a 1×1
 748 kernel to downsample the channel dimension from 64 to 1. This tensor is the translation Q -function
 749 of dimensions $\mathbb{R}^{100 \times 100 \times 100 \times 1}$. For rotation, gripper open, and collision avoidance actions, the
 750 voxel feature grid is max-pooled along the 3D dimensions to form a vector of dimensions $\mathbb{R}^{1 \times 64}$.
 751 This vector is decoded with three independent linear layers to form the respective Q -functions for
 752 rotation, gripper open, and collision avoidance. The rotation linear layer outputs logits of dimensions
 753 \mathbb{R}^{216} (72 bins of 5 degree increments for each of the three axes). The gripper open and collide linear
 754 layers output logits of dimensions \mathbb{R}^2 .

755 Our codebase is built on the ARM repository⁶ by James et al. [14]. We will release our PyTorch
 756 implementation upon publication.

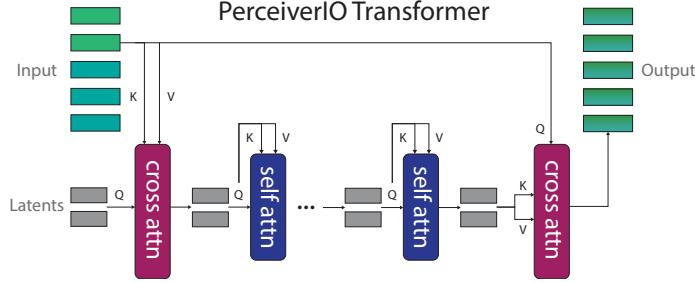


Figure 6. **Perceiver Transformer Architecture.** Perceiver is a latent-space transformer. Q, K, V represent queries, keys, and values, respectively. We use 6 self-attention layers in our implementation.

757 **C Evaluation Workflow**

758 C.1 Simulation

759 Simulated experiments in Section 4.2 follow a four-phase workflow: (1) generate a dataset with
 760 train, validation, and test sets, each containing 100, 25, and 25 demonstrations, respectively. (2)
 761 Train an agent on the train set and save checkpoints at intervals of 10K iterations. (3) Evaluate all
 762 saved checkpoints on the validation set, and mark the best performing checkpoint. (4) Evaluate the
 763 best performing checkpoint on the test set. While this workflow follows a standard train-val-test
 764 paradigm from supervised learning, it is not the most feasible workflow for real-robot settings. With
 765 real-robots, collecting a validation set and evaluating all checkpoints could be very expensive.

766 C.2 Real-Robot

767 For real-robot experiments in Section 4.4, we simply pick the last checkpoint from training. We
 768 check if the agent has been sufficiently trained by visualizing Q -predictions on training examples
 769 with swapped or modified language goals. While evaluating a trained agent, the agent keeps acting
 770 until a human user stops the execution. We also visualize the Q -predictions live to ensure that the
 771 agent’s upcoming action is safe to execute.

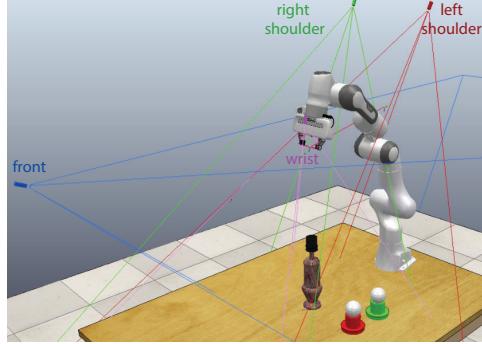
⁵<https://github.com/lucidrains/perceiver-pytorch>

⁶<https://github.com/stepjam/ARM>

772 **D Robot Setup**

773 **D.1 Simulation**

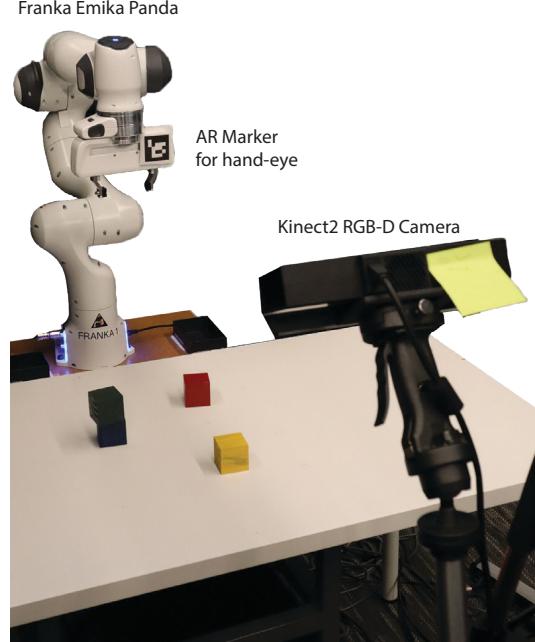
774 All simulated experiments use the four camera
 775 setup illustrated in Figure 7. The front, left
 776 shoulder, and right shoulder cameras, are
 777 static, but the wrist camera moves with the end-
 778 effector. We did not modify the default cam-
 779 era poses from RL Bench [15]. These poses
 780 maximize coverage of the tabletop, while min-
 781 imizing occlusions caused by the moving arm.
 782 The wrist camera in particular is able to pro-
 783 vide high-resolution observations of small ob-
 784 jects like handles.



785 **D.2 Real-Robot**
 786 **Hardware Setup.** The real-robot experiments use a Franka Panda manipulator with a parallel-
 787 gripper. For perception, we use a Kinect-2 RGB-D camera mounted on a tripod, at an angle, pointing
 788 towards the tabletop. See Figure D for reference. We tried setting-up multiple Kinetics for multi-
 789 view observations, but we could not fix the interference issue caused by multiple Time-of-Flight
 790 sensors. The Kinect-2 provides RGB-D images of resolution 512×424 at 30Hz. The extrinsics
 791 between the camera and robot base-frame are calibrated with the `easy_handeye` package⁷. We use
 792 an ARUCO⁸ AR marker mounted on the gripper to aid the calibration process.

793 **Data Collection.** We collect demonstrations
 794 with an HTC Vive controller. The controller is a
 795 6-DoF tracker that provides accurate poses with
 796 respect to a static base-station. These poses are
 797 displayed as a marker on RViz⁹ along with the
 798 real-time RGB-D pointcloud from the Kinect-
 799 2. A user specifies target poses by using the
 800 marker and pointcloud as reference. These tar-
 801 get poses are executed with a motion-planner.
 802 We use Franka ROS and MoveIt¹⁰, which by
 803 default uses an RRT-Connect planner.

804 **Training and Execution.** We train a PER-
 805 ACT agent from scratch with 53 demon-
 806 strations. The training samples are augmented with
 807 $\pm 0.125\text{m}$ translation perturbations and $\pm 45^\circ$
 808 yaw rotation perturbations. We train on 8
 809 NVIDIA P100 GPUs for 2 days. During eval-
 810 uation, we simply chose the last checkpoint from
 811 training (since we did not collect a validation
 812 set for optimization). Inference is done on a
 813 single Titan X GPU.



814 **Figure 8. Real-Robot Setup with Kinect-2 and Franka Panda.**

⁷https://github.com/IFL-CAMP/easy_handeye

⁸https://github.com/pal-robotics/aruco_ros

⁹<http://wiki.ros.org/rviz>

¹⁰http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/

814 E Data Augmentation

815 PERACT’s voxel-based formulation naturally allows for data augmentation with SE(3) transformations.
 816 During training, samples of voxelized observations \mathbf{v} and their corresponding keyframe actions \mathbf{k} are perturbed with random translations and rotations. Translation perturbations have a range
 817 of $[\pm 0.125\text{m}, \pm 0.125\text{m}, \pm 0.125\text{m}]$. Rotation perturbations are limited to the yaw axis and have a
 818 range of $[0^\circ, 0^\circ, \pm 45^\circ]$. The 45° limit ensures that the perturbed rotations do not go beyond what is
 819 kinematically reachable for the Franka arm. We did experiment with pitch and roll perturbations, but
 820 they substantially lengthened the training time. Any perturbation that pushed the discretized action
 821 outside the observation voxel grid was discarded. See the bottom row of Figure 10 for examples of
 822 data augmentation.
 823

824 F Demo Augmentation

825 Following James et al. [15], we cast every datapoint in a demonstration as a “predict the next (best) keyframe action” task. See Figure 9 for an illustration of this process.
 826 In this illustration, k_1 and k_2 are two keyframes that were
 827 extracted from the method described in Section 3.2. The
 828 orange circles indicate datapoints whose RGB-D obser-
 829 vations are paired with the next keyframe action.
 830

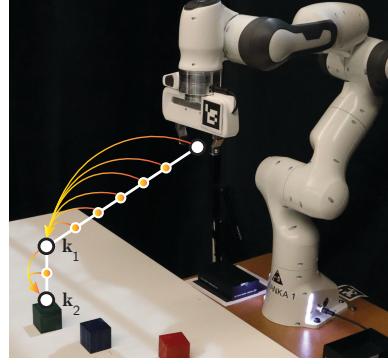


Figure 9. Keyframes and Demo Augmentation.

832 G Sensitivity Analysis

833 In Table 4, we investigate three factors that affect PERACT’s performance: rotation data augmentation,
 834 number of Perceiver latents, and voxelization resolution. All multi-task agents were trained
 835 with 100 demonstrations per task and evaluated on 25 episodes per task. To briefly summarize these
 836 results: (1) 45° yaw perturbations improve performance on tasks with lots of rotation variations like
 837 `stack blocks`, but also worsen performance on tasks with constrained rotations like `place wine`.
 838 (2) PERACT with just 512 latents is competitive with (and sometimes even better than) the default
 839 agent with 2048 latents, which showcases the compression capability of the Perceiver architecture.
 840 (3) Coarse grids like 32^3 are sufficient for some tasks, but high-precision tasks like `sort shape`
 841 need higher resolution voxelization.

Table 4. Sensitivity Analysis. Success rates (mean %) of various PERACT agents trained with 100 demonstrations per task. We investigate three factors that affect PERACT’s performance: rotation augmentation, number of Perceiver latents, and voxel resolution.

	open drawer	slide block	sweep to dustpan	meat off grill	turn tap	put in drawer	close jar	drag stick	stack blocks
PERACT	80	72	56	84	80	68	60	68	36
PERACT w/o Rot Aug	92	72	56	92	96	60	56	100	8
PERACT 4096 latents	84	88	44	68	84	48	48	84	12
PERACT 1024 latents	84	48	52	84	84	52	32	92	12
PERACT 512 latents	92	84	48	100	92	32	32	100	20
PERACT 64^3 voxels	88	72	80	60	84	36	40	84	32
PERACT 32^3 voxels	28	44	100	60	72	24	0	24	0
	screw bulb	put in safe	place wine	put in cupboard	sort shape	push buttons	insert peg	stack cups	place cups
PERACT	24	44	12	16	20	48	0	0	0
PERACT w/o Rot Aug	20	32	48	8	8	56	8	4	0
PERACT 4096 latents	32	44	52	8	12	72	4	4	0
PERACT 1024 latents	24	32	36	8	20	40	8	4	0
PERACT 512 latents	48	40	36	24	16	32	12	0	4
PERACT 64^3 voxels	24	48	44	12	4	32	0	4	0
PERACT 32^3 voxels	12	20	52	0	0	60	0	0	0

842 **H High-Precision Tasks**

843 In Table 1, PERACT achieves zero performance on three high-
 844 precision tasks: place cups, stack cups, and insert peg. To
 845 investigate if multi-task optimization is itself one of the factors af-
 846 fecting performance, we train 3 separate single-task agents for each
 847 task. We find that single-task agents are able to achieve non-zero
 848 performance, indicating that better multi-task optimization methods
 849 might improve performance on certain tasks.

	Multi	Single
place cups	0	24
stack cups	0	32
insert peg	0	16

Table 5. Success rates (mean %) of multi-task and single-task PERACT agents trained with 100 demos and evaluated on 25 instances.

850 **I Additional Related Work**

851 In this section, we briefly discuss additional related work that were not mentioned in Section 2.

852 **Concurrent Work.** Recently, Mandi et al. [73] found that pre-training and fine-tuning on new tasks
 853 is competitive, or even better, than meta-learning approaches for RLBench tasks in multi-task (but
 854 single-variation) settings. This pre-training and fine-tuning paradigm might be directly applicable
 855 to PERACT, where a pre-trained PERACT agent could be quickly adapted to new tasks without the
 856 explicit use of meta-learning algorithms.

857 **Multi-Task Learning.** In the context of RLBench, Auto- λ [63] presents a multi-task optimization
 858 framework that goes beyond uniform task weighting from Section 3.4. The method dynamically
 859 tunes task weights based on the validation loss. Future works with PERACT could replace uniform
 860 task weighting with Auto- λ for better multi-task performance. In the context of Meta-World [47],
 861 Sodhani et al. [74] found that language-conditioning leads to substantial performance gains in multi-
 862 task RL on 50 task variations.

863 **Language-based Planning.** In this paper, we only investigated single-goal settings where the lan-
 864 guage instruction does not change throughout the episode. However, language-conditioning natu-
 865 ral allows for composing several instructions in a sequential manner [59]. As such, several prior
 866 works [75, 13, 76, 77] have used language as medium for planning high-level actions, which can
 867 then be executed with pre-trained low-level skills. Future works could incorporate language-based
 868 planning for grounding abstract goals like “make dinner”.

869 **Voxel Representations.** Voxel-based representations have been used in several domains that specif-
 870 ically benefit from 3D understanding. In Neural Radiance Fields (NeRFs), voxel-based feature grids
 871 have dramatically reduced training and rendering times [78, 79]. In vision-language grounding,
 872 voxel maps have been used to build persistent scene representations [80, 81]. Similarly, other works
 873 in robotics have used voxelized representations to embed viewpoint-invariance for driving [82] and
 874 manipulation [83]. Furthermore, the use of latent vectors in PerceiverIO [1] is broadly related to
 875 voxel hashing [84] from computer graphics. Instead of using a location-based hashing function to
 876 map voxels to fixed size memory, PerceiverIO uses cross attention to map the input to fixed size la-
 877 tent vectors, which are trained end-to-end. Another major difference is the treatment of unoccupied
 878 space. In graphics, unoccupied space does not affect rendering, but in PERACT, unoccupied space is
 879 where a lot of “action detections” happen. Thus the relationship between unoccupied and occupied
 880 space, i.e., scene, objects, robot, is crucial for learning action representations.

881 J Additional Q-Prediction Examples

882 Figure 10 showcases additional Q -prediction examples from trained PERACT agents. Traditional
 883 object-centric representations like poses and instance-segmentations struggle to represent piles of
 884 beans or tomato vines with high-precision. Whereas action-centric agents like PERACT focus on
 885 learning perceptual representations of actions, which elevates the need for practitioners to define
 886 *what should be an object*.

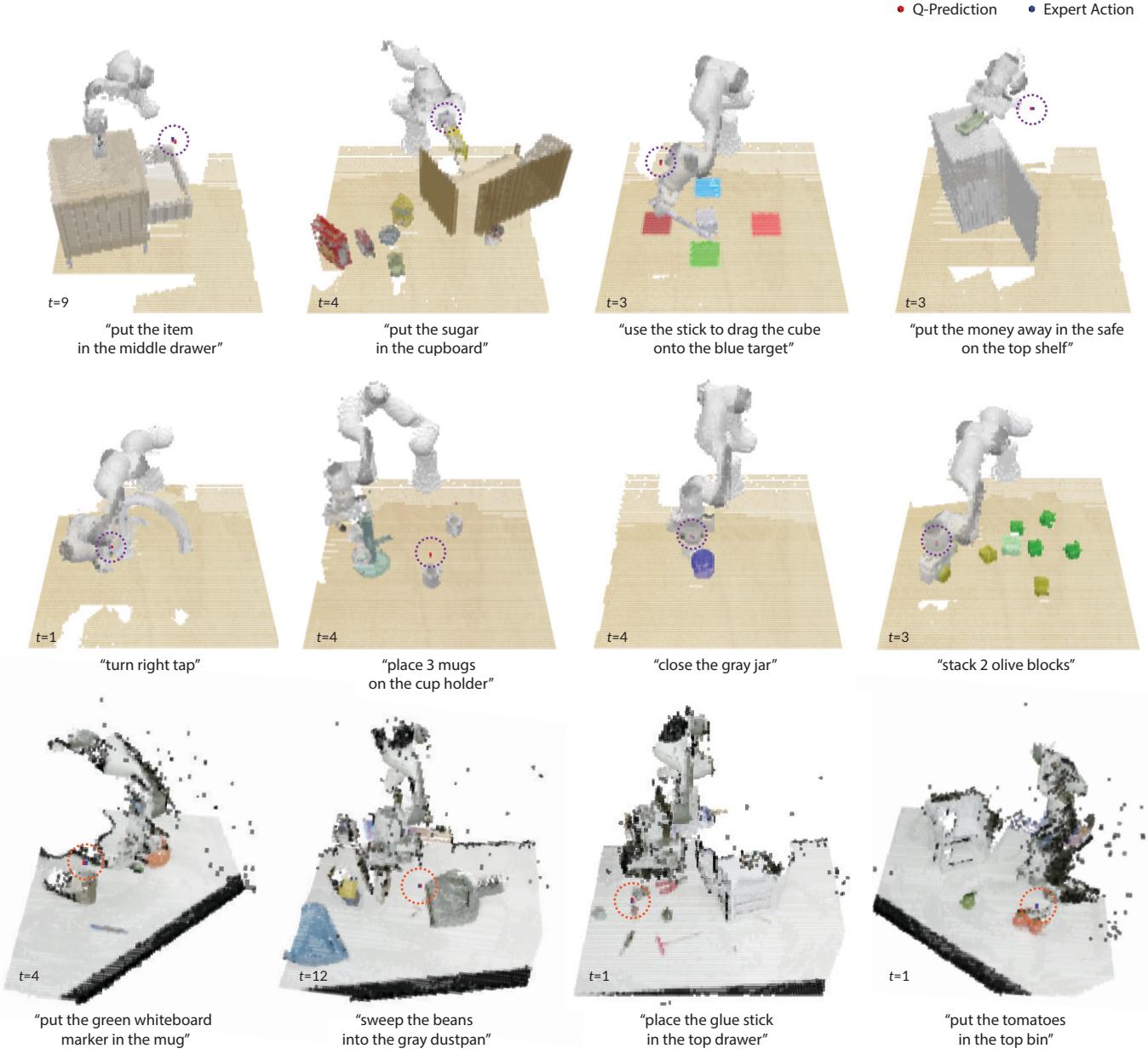


Figure 10. **Additional Q-Prediction Examples.** Q -Prediction examples from PERACT. The top two rows are from simulated tasks without any data augmentation perturbations, and the bottom row is from real-world tasks with translation and yaw-rotation perturbations.

887 **K Things that did not work**

888 In this section, we describe things we tried, but did not work or caused issues in practice.

889 **Real-world multi-camera setup.** We tried setting up multiple Kinect-2s for real-world multi-view
890 observations, but we could not solve interference issues with multiple Time-of-Flight sensors. Par-
891 ticularly, the depth frames became very noisy and had lots of holes. Future works could try turning
892 the cameras on-and-off in a rapid sequence, or use better Time-of-Flight cameras with minimal
893 interference.

894 **Fourier features for positional embeddings.** Instead of the learned positional embeddings, we
895 also experimented with concatenating Fourier features to the input sequence like in some Perceiver
896 models [1]. The Fourier features led to substantially worse performance.

897 **Pre-trained vision features.** Following CLIPort [16], we tried using pre-trained vision features
898 from CLIP [66], instead of raw RGB values, to bootstrap learning and also to improve generalization
899 to unseen objects. We ran CLIP’s ResNet50 on each of the 4 RGB frames, and upsampled features
900 with shared decoder layers in a UNet fashion. But we found this to be extremely slow, especially
901 since the ResNet50 and decoder layers need to be run on 4 independent RGB frames. With this
902 additional overhead, training multi-task agents would have taken substantially longer than 16 days.
903 Future works could experiment with methods for pre-training the decoder layers on auxiliary tasks,
904 and pre-extracting features for faster training.

905 **Upsampling at multiple self-attention layers.** Inspired by Dense Prediction Transformers
906 (DPT) [85], we tried upsampling features at multiple self-attention layers in the Perceiver Trans-
907 former. But this did not work at all; perhaps the latent-space self-attention layers of Perceiver are
908 substantially different to the full-input self-attention layers of ViT [4] and DPT [85].

909 **Extreme rotation augmentation.** In addition to yaw rotation perturbations, we also tried perturbing
910 the pitch and roll. While PERACT was still able to learn policies, it took substantially longer to train.
911 It is also unclear if the default latent size of $\mathbb{R}^{2048 \times 512}$ is appropriate for learning 6-DoF polices with
912 such extreme rotation perturbations.

913 **Using Adam instead of LAMB.** We tried training PERACT with the Adam [86] optimizer instead
914 of LAMB [68], but this led to worse performance in both simulated and real-world experiments.

915 **L Limitations and Risks**

916 While PERACT is quite capable, it is not without limitations. In the following sections, we discuss
917 some of these limitations and potential risks for real-world deployment.

918 **Sampling-Based Motion Planner.** PERACT relies on a sampling-based motion planner to execute
919 discretized actions. This puts PERACT at the mercy of randomized planner to reach poses. While
920 this issue did not cause any major problems with the tasks in our experiments, a lot of other tasks
921 are sensitive to the paths taken to reach poses. For instance, pouring water into a cup would require
922 a smooth path for tilting the water container appropriately. This could be addressed in future works
923 by using a combination of learned and sampled motion paths [87].

924 **Dynamic Manipulation.** Another issue with discrete-time discretized actions is that they are not
925 easily applicable to dynamic tasks that require real-time closed-loop maneuvering. This could be
926 addressed with a separate visuo-servoing mechanism that can reach target poses with closed-loop
927 control. Alternatively, instead of predicting just one action, PERACT could be extended to predict
928 a sequence of discretized actions. Here, the Transformer-based architecture could be particularly
929 advantageous.

930 **Dexterous Manipulation.** Using discretized actions with N-DoF robots like multi-fingered hands is
931 also non-trivial. Specifically for multi-fingered hands, PERACT could be modified to predict finger-
932 tip poses that can be reached with an IK (Inverse Kinematics) solver. But it is unclear how feasible
933 or robust such an approach would be with under-actuated systems like multi-fingered hands.

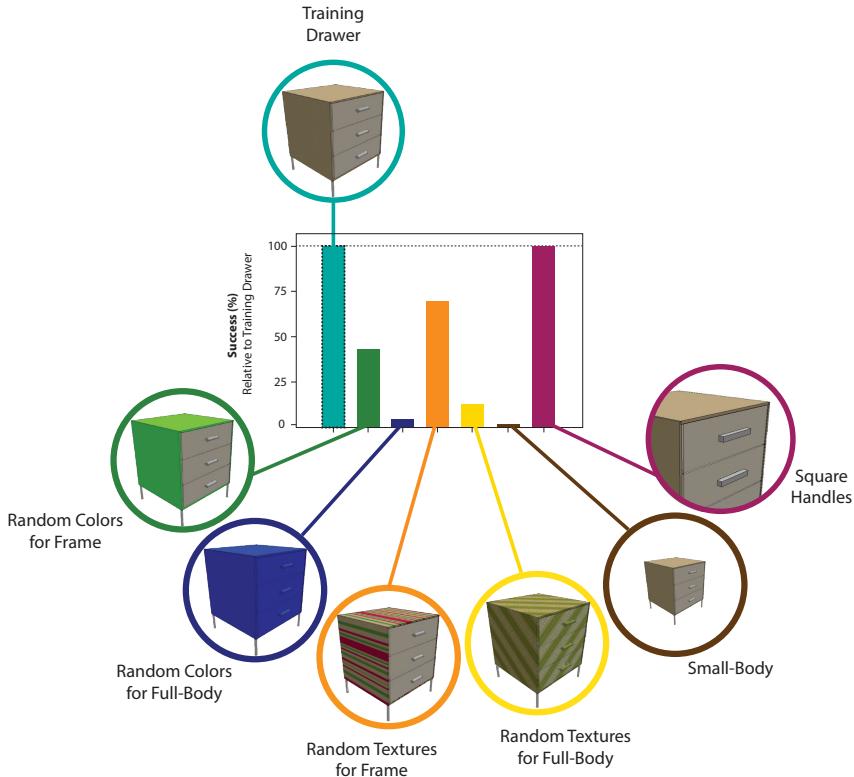


Figure 11. Perturbation Tests. Results from a multi-task PERACT agent trained on a single drawer and evaluated on several instances perturbed drawers. Each perturbation consists of 25 evaluation instances, and reported successes are relative to the training drawer.

934 **Generalization to Novel Instances and Objects.** In Figure 11, we report results from small-scale
 935 perturbation experiments on the open drawer task. We observe that changing the shape of the
 936 handles does not affect performance. However, handles with randomized textures and colors confuse
 937 the agent since it has only seen one type of drawer color and texture during training. Going beyond
 938 this one-shot setting, and training on several instances of drawers might improve generalization
 939 performance. Although we did not explicitly study generalization to unseen objects, it might be
 940 feasible to train PERACT’s action-detector on a broad range of objects and evaluate its ability to
 941 handle novel objects, akin to how language-conditioned instance-segmentors and object-detectors
 942 are used [88]. Alternatively, pre-trained vision features from multi-modal encoders like CLIP [66]
 943 or R3M [34] could be used to bootstrap learning.

944 **Scope of Language Grounding.** Like with prior work [16], PERACT’s understanding of verb-noun
 945 phrases is closely grounded in demonstrations and tasks. For example, “cleaning” in “*clean the*
 946 *beans on the table with a dustpan*” is specifically associated with the action sequence of pushing
 947 beans on to a dustpan, and not “cleaning” in general, which could be applied to other tasks like
 948 cleaning the table with a cloth.

949 **Predicting Task Completion.** For both real-world and simulated evaluations, an oracle indicates
 950 whether the desired goal has been reached. This oracle could be replaced with a success classifier
 951 that can be pre-trained to predict task completion from RGB-D observations.

952 **Data Augmentation with Kinematic Feasibility.** The data augmentation method described in Sec-
 953 tion E does not consider the kinematic feasibility of reaching perturbed actions with the Franka arm.
 954 Future works could pre-compute unreachable poses in the discretized action space, and discard any
 955 augmentation perturbations that push actions into unreachable zones.

956 **Balanced Datasets.** Since PERACT is trained with just a few demonstrations, it occasionally tends
 957 to exploit biases in the training data. For instance, PERACT might have a tendency to always “*place*
 958 *blue blocks on yellow blocks*” if such an example is over-represented in the training data. Such
 959 issues could be potentially fixed by scaling datasets to include more diverse examples of objects and
 960 attributes. Additionally, data visualization methods could be used to identify and fix these biases.

961 **Multi-Task Optimization.** The uniform task sampling strategy presented in Section 3.4 might
 962 sometimes hurt performance. Since all tasks are weighted equally, optimizing for certain tasks
 963 with common elements (e.g., moving blocks), might adversarial affect the performance on other
 964 dissimilar tasks (e.g., turning taps). Future works, could use dynamic task-weighting methods like
 965 Auto- λ [63] for better multi-task optimization.

966 **Deployment Risks.** PERACT is an end-to-end framework for 6-DoF manipulation. Unlike some
 967 methods in Task-and-Motion-Planning (TAMP) that can sometimes provide theoretical guarantees
 968 on task completion, PERACT is a purely reactive system, whose performance can only be evaluated
 969 through empirical means. Also, unlike prior works [16], we do not use internet pre-trained vision
 970 encoders that sometimes contain harmful biases [89, 90]. Even so, it might be prudent to thoroughly
 971 study and mitigate any biases before deployment. As such, for real-world applications, keeping
 972 humans in the loop both during training and testing, might help.

973 M Emergent Properties

974 In this section, we present some preliminary findings on the emergent properties of PERACT.

975 M.1 Object Tracking

976 Although PERACT was not explicitly trained for 6-DoF
 977 object-tracking, our action detection framework can be
 978 used to localize objects in cluttered scenes. In this [video](#),
 979 we show an agent that was trained with one hand sani-
 980 tizer instance on just 5 “press the handsan” demos, and
 981 then evaluated on tracking an unseen sanitizer instance.
 982 PERACT does not need to build a complete representation
 983 of hand sanitizers, and only has to learn *where to press*
 984 them. Our implementation runs at an inference speed of
 985 2.23 FPS (or 0.45 seconds per frame), allowing for near
 986 real-time closed-loop behaviors.

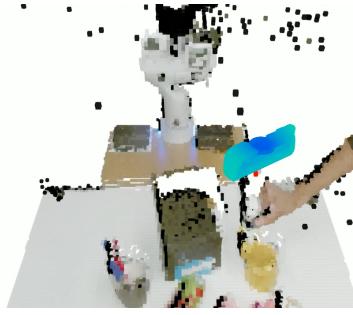


Figure 12. Object Tracker. Tracking an unseen hand sanitizer instance.

987 M.2 Multi-Modal Action Predictions

988 PERACT’s problem formulation naturally allows for
 989 modeling multi-modal action distributions, i.e., scenar-
 990 ios where multiple actions are valid given a specific
 991 goal. Figure 13 presents some selected examples of
 992 multi-modal action predictions from PERACT. Since
 993 there are several “yellow blocks” and “cups” to choose
 994 from, the Q -predictions have several modes. In prac-
 995 tice, we observe that the agent has a tendency to pre-
 996 fer certain object instances over others (like the front
 997 mug in Figure 13) due to preference biases in the train-
 998 ing dataset. We also note that the cross-entropy based
 999 training method from Section 3.4 is broadly related to
 1000 Energy-Based Models (EBMs) [91, 92]. In some sense,
 1001 the cross-entropy loss is *pulling up* ground-truth 6-DoF
 1002 actions, while *pushing-down* every other action in the
 1003 discretized action space. At test time, we simply max-
 1004 imize the learned Q -predictions, instead of minimizing an
 1005 energy function with optimization. Future works could look
 1006 into EBM training and inference methods for better gen-
 1007 eralization and performance.

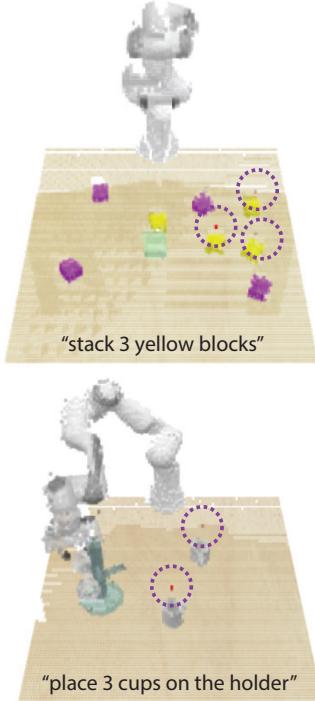


Figure 13. Examples of Multi-Modal Predictions.