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

Anonymous Author(s)
Affiliation
Address
email

peract.github.io

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

Keywords: Transformers, Language Grounding, Manipulation, Behavior Cloning

1 Introduction

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

Language models operate on sequences of tokens [10], and vision transformers operate on sequences of image patches [4]. While pixel transformers [11, 1] exist, they are not as data efficient as approaches that use convolutions or patches to exploit the 2D structure of images. Thus, while Transformers may be domain agnostic, they still require the right problem formulation to be data efficient. A similar efficiency issue is apparent in behavior-cloning (BC) approaches that directly train image-to-action agents for 6-DoF manipulation. In Gato [9], a multi-task Transformer was trained on several domains, including MetaWorld tasks [12] like opening a drawer and pressing a button, but this required a total of 95K episodes. Relatedly, recent works in language-conditioned behavior-cloning [13, 14] for manipulation have shown impressive multi-task capabilities, but they require several days or even months of data collection. Instead of collecting thousands of demonstrations, in this work, we aim to exploit the 3D structure of voxelized observations for efficient 6-DoF behavior cloning (analogous to how vision transformers exploit the 2D structure of images with patches).

To this end, we present PERACT (short for PERCEIVER-ACTOR), a language-conditioned BC agent that can learn to imitate a wide variety of 6-DoF manipulation tasks with just a few demonstrations per task. Instead of operating on image patches, PERACT encodes a sequence of RGB-D voxel patches and predicts discretized translations, rotations, and gripper actions that are executed with

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

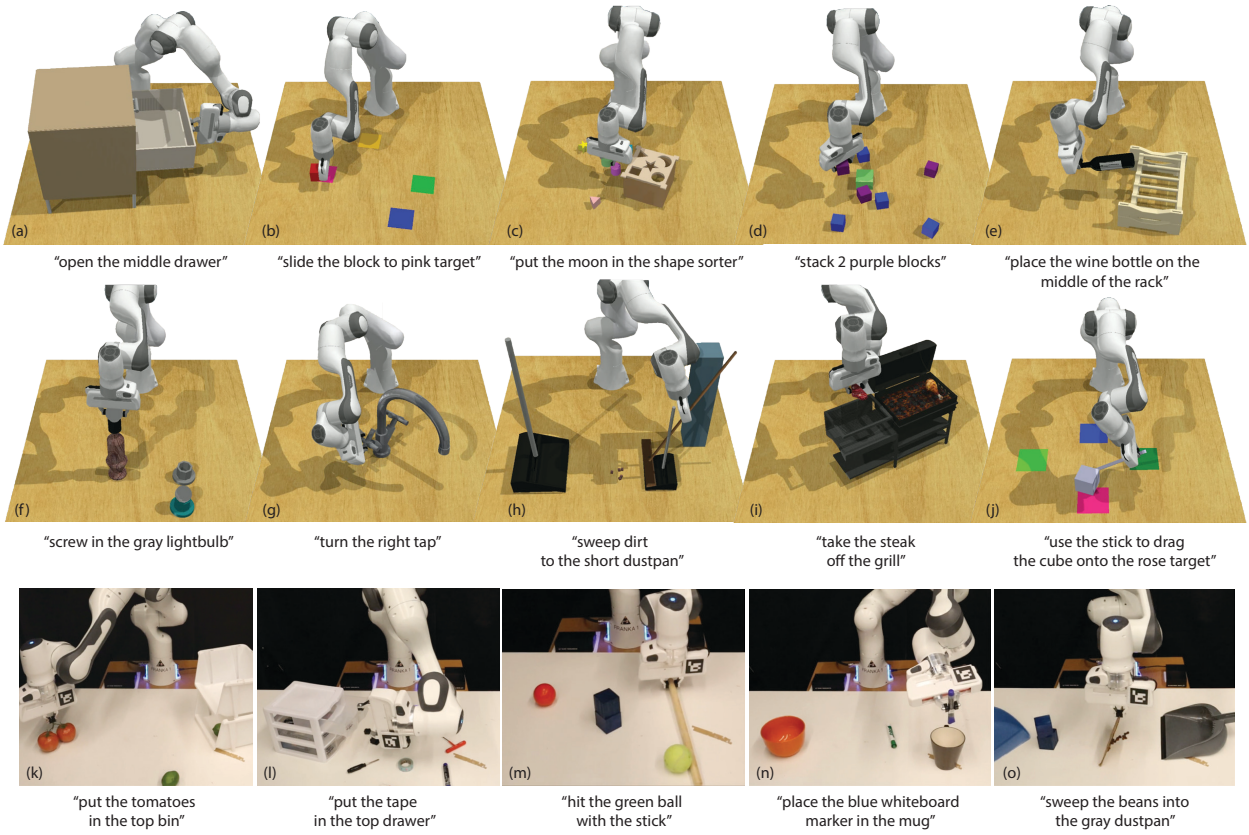

Figure 1. **Language-Conditioned Manipulation Tasks:** PERACT is a language-conditioned multi-task agent capable of imitating a wide range of 6-DoF manipulation tasks. We conduct experiments with 18 simulated tasks in RLBench [15] (a-j; only 10 shown), each with several pose and semantic variations. 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.

a motion-planner in an observe-act loop. PERACT is essentially a classifier trained with supervised learning to *detect actions* akin to prior work like CLIPort [16], except our observations and actions are represented with 3D voxels instead of 2D pixels. Voxel grids are less prevalent than images in end-to-end BC approaches often due to scaling issues with high-dimensional inputs. But in PERACT, we use a Perceiver[1] Transformer [1] to encode very high-dimensional input of up to 100^3 voxels with only a small set of latent vectors. This voxel-based formulation provides a strong structural prior with several benefits: a natural method for fusing multi-view observations, learning robust action-centric representations [17], and enabling data augmentation in 6-DoF – all of which help learn generalizable skills by focusing on *diverse* rather than narrow multi-task data.

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

In summary, our contributions are as follows:

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

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

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

2 Related Work

Vision for Manipulation. Traditionally, methods in robot perception have used explicit "object" representations like instance segmentations, object classes, poses [18, 19, 20, 21, 22, 23]. Such methods struggle with deformable and granular items like cloths and beans that are hard to represent with geometric models or segmentations. In contrast, recent methods [16, 24] learn action-centric representations without any "objectness" assumptions, but they are limited to top-down 2D settings with simple pick-and-place primitives. In 3D, James et al. proposed C2FARM [25], an action-centric reinforcement learning (RL) agent with a coarse-to-fine-grain 3D-UNet backbone. The coarse-to-fine-grain scheme has a limited receptive field that cannot look at the entire scene at the finest level. In contrast, PERACT learns action-centric representations with a global-receptive field through a Transformer backbone. Also, PERACT does BC instead of RL, which enables us to easily train a multi-task agent for several tasks by conditioning it with language goals.

End-to-End Manipulation approaches [26, 27, 28, 29] make the least assumptions about objects and tasks, but are often formulated as an image-to-action prediction task. Training directly on RGB images for 6-DoF tasks is often inefficient, generally requiring several demonstrations or episodes just to learn basic skills like rearranging objects. In contrast, PERACT uses a voxelized observation and action space, which is dramatically more efficient and robust in 6-DoF settings. While other works in 6-DoF grasping [30, 31, 32, 33, 34] have used RGB-D and pointcloud input, they have not been applied to sequential tasks or used with language-conditioning. Another line of work tackles data inefficiency by using pre-trained image representations [16, 35, 36] to bootstrap BC. Although our framework is trained from scratch, such pre-training approaches can be integrated together in future works for even greater efficiency and generalization to unseen objects.

Transformers for Agents and Robots. Transformers have become the prevalent architecture in several domains. Starting with NLP [2, 3, 37], recently in vision [4, 38], and even RL [8, 39, 40]. In robotics, Transformers have been applied to assistive teleop [41], legged locomotion [42], path-planning [43, 44], imitation learning [45, 46], and grasping [47]. Transformers have also achieved impressive results in multi-domain settings like in Gato [9] where a single Transformer was trained for 16 domains such as captioning, language-grounding, robotic control etc. However, Gato relies on extremely large datasets like 15K episodes for block stacking and 94K episodes for Meta-World [12] tasks. Our voxel-based approach might complement agents like Gato, which could use our problem formulation for greater efficiency and robustness in 6-DoF manipulation settings.

Language Grounding for Manipulation. Several works have proposed methods for grounding language in robot actions [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58]. However, these methods use disentangled pipelines for perception and action, with the language primarily being used to guide perception. Recently, a number of end-to-end approaches [13, 14, 59] have been proposed for conditioning BC agents with language instructions. These methods require 1000s of human teleoperated demonstrations that are collected over several days or even months. In contrast, PERACT can learn robust multi-task policy with just a few minutes of training data. For benchmarking, several simulation environments exist [60, 24, 12], but we use RLBench [15] for its diversity of 6-DoF tasks and ease of generating expert demonstrations with templated language goals.

3 PERCEIVER-ACTOR

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

Section 3.1 and Section 3.2 describe our dataset setup. Section 3.3 describes our problem formulation for PERACT, and Section 3.4 provides details on training PERACT.

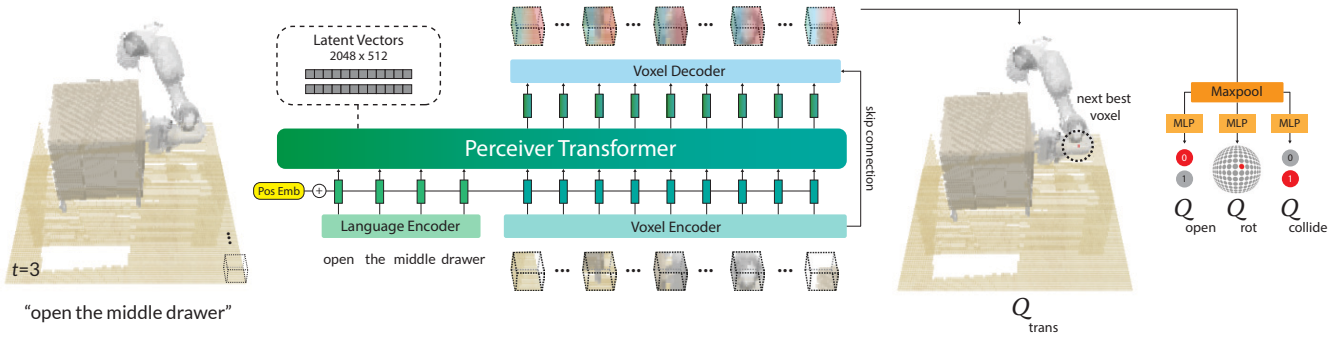

Figure 2. **PERACT Overview.** PERACT is a language-conditioned behavior-cloning agent trained with supervised learning to *detect actions*. PERACT takes as input a language goal and a voxel grid reconstructed from RGB-D sensors. The voxels are split into 3D patches, and the language goal is encoded 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 discretized translation, rotation, gripper open, and collision avoidance actions. 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.

3.1 Demonstrations

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

3.2 Keyframes and Voxelization

Following prior work by James et al. [25], we construct a structured observation and action space through keyframe extraction and voxelization.

Training our agent to directly predict continuous actions is inefficient and noisy. So instead, for each demonstration ζ, we extract a set of keyframe actions $\{\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_m\} \subset \mathcal{A}$ that capture bottlenecks [61] in the action sequence with a simple heuristic: an action is a keyframe if (1) the joint-velocities are near zero and (2) the gripper open state has not changed. Each datapoint in the demonstration ζ can then be cast as a "predict the next (best) keyframe action" task. See Appendix Figure F for an illustration of this process.

To learn action-centric representations [17] in 3D, we use a voxel grid [62, 63] to represent both the observation and action space. The observation voxels $\mathbf{v}$ are reconstructed from RGB-D observations $\tilde{o}$ fused through triangulation $\tilde{o} \Rightarrow \mathbf{v}$ from known camera extrinsics and intrinsics. By default, we use a voxel grid of 100^3, which corresponds to a volume of 1.0m^3 in metric scale. The keyframe actions $\mathbf{k}$ are discretized such that training our BC agent can be formulated as a "next best action" classification task. Translation is simply the closest voxel to the center of the gripper fingers. Rotation is discretized into 5 degree bins for each of the three rotation axes. Gripper open state is a binary value. Collide is also a binary value that indicates if the motion-planner should avoid everything in the voxel grid or nothing at all; switching between these two modes of collision avoidance is crucial as tasks often involve both contact based (e.g., pulling the drawer open) and non-contact based motions (e.g., reaching the handle without colliding into anything).

3.3 PERACT Agent

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

The language goal $\mathbf{l}$ is encoded with a pre-trained language model. We use CLIP's [64] language encoder, but any pre-trained language model would suffice [13, 59]. Our choice of CLIP opens up possibilities for future work to use pre-trained vision features that are aligned with the language for better generalization to unseen semantic categories and instances [16].

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

The input sequence of language and voxel encodings is extremely long. A standard Transformer with $\mathcal{O}(n^2)$ self-attention connections and an input of $(100/5)^3 = 8000$ patches is hard to fit on the memory of a commodity GPU. Instead, we use the Perceiver [1] Transformer. Perceiver is a latent-space Transformer, where instead of attending to the entire input, it first computes cross-attention between the input and a much smaller set of latent vectors (which are randomly initialized and trained). These latents are encoded with self-attention layers, and for the final output, the latents are again cross-attended with the input to match the input-size. See Appendix Figure 6 for an illustration. By default, we use 2048 latents of dimension 512 : $\mathbb{R}^{2048\times512}$, but in Appendix G we experiment with different latent sizes.

The Perceiver Transformer uses 6 self-attention layers to encode the latents and outputs a sequence of patch encodings. These patch encodings are upsampled with a 3D convolution layer and tri-linear upsampling. The decoder includes a skip-connection from the encoder (like in UNets [65]). The per-voxel encodings are then used to predict discretized actions. For translation, the voxel encodings are reshaped into the original voxel grid (100^3) to form a 3D $\mathcal{Q}$-function of action-values. For rotation, gripper open, and collide, the encodings are max-pooled and then decoded with linear layers to form their respective $\mathcal{Q}$-function. The best action $\mathcal{T}$ is chosen by simply maximizing the $\mathcal{Q}$-functions:

$$\mathcal{T}_{\text{trans}} = \underset{(x,y,z)}{\text{argmax}}\ \mathcal{Q}_{\text{trans}}((x,y,z) \mid \mathbf{v}, \mathbf{l}), \qquad \mathcal{T}_{\text{rot}} = \underset{(\psi,\theta,\phi)}{\text{argmax}}\ \mathcal{Q}_{\text{rot}}((\psi,\theta,\phi) \mid \mathbf{v}, \mathbf{l}),$$
$$\mathcal{T}_{\text{open}} = \underset{\omega}{\text{argmax}}\ \mathcal{Q}_{\text{open}}(\omega \mid \mathbf{v}, \mathbf{l}), \qquad \mathcal{T}_{\text{collide}} = \underset{\kappa}{\text{argmax}}\ \mathcal{Q}_{\text{collide}}(\kappa \mid \mathbf{v}, \mathbf{l}),$$

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

3.4 Training Details

PERACT is trained through supervised learning with discrete-time input-action tuples from a dataset of demonstrations. These tuples are composed of voxel observations, language goals, and keyframe actions $\{(\mathbf{v}_1, \mathbf{l}_1, \mathbf{k}_1), (\mathbf{v}_2, \mathbf{l}_2, \mathbf{k}_2), \ldots\}$. During training, we randomly sample a tuple and supervise the agent to predict the keyframe action $\mathbf{k}$ given the observation and goal $(\mathbf{v}, \mathbf{l})$. For translations, the ground-truth action is represented as a one-hot voxel encoding $Y_{\text{trans}} : \mathbb{R}^{H\times W\times D}$. Rotations are also represented with a one-hot encoding per rotation axis with R rotation bins $Y_{\text{rot}} : \mathbb{R}^{(360/R)\times3}$ ($R = 5$ degrees for all experiments). Similarly, open and collide variables are binary one-hot vectors $Y_{\text{open}} : \mathbb{R}^2$, $Y_{\text{collide}} : \mathbb{R}^2$. The agent is trained with cross-entropy loss like a classifier:

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

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

By default, we use a voxel grid size of 100^3. We conducted validation tests by replaying expert demonstrations with discretized actions to ensure that 100^3 is a sufficient resolution for execution. The agent was trained with a batch-size of 16 on 8 NVIDIA V100 GPUs for 16 days (600K iterations). We use the LAMB [66] optimizer following Perceiver [1].

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

4 Results

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

4.1 Simulation Setup

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

Environment. The simulation is set in CoppelaSim [67] and interfaced through PyRep [68]. All experiments use a Franka Panda robot with a parallel gripper. The input observations are captured from four RGB-D cameras positioned at the front, left shoulder, right shoulder, and on the wrist, as shown in Appendix Figure 7. All cameras are noiseless and have a resolution of 128×128.

Language-Conditioned Tasks. We train and evaluate on 18 RLBench [15] tasks. See Figure 1 for examples and Appendix A for details on individual tasks. Each task also includes several variations, ranging from 2-60 possibilities, e.g., in the `stack blocks` task, *"stack 2 red blocks"* and *"stack 4 purple blocks"* are two variants. These variants are randomly sampled during data generation, but kept consistent during evaluations for one-to-one comparisons. Some RLBench tasks were modified to include additional variations to stress-test multi-task and language-grounding capabilities. There are a total of 249 variations across 18 tasks, and the number of extracted keyframes range from 2-17 frames. Language goals are constructed from templates (but human-annotated for real-world tasks). Note that in all experiments, we do not test for generalization to unseen objects, i.e., our train and test objects are the same. However the poses, goals, and semantic instantiations (like color, shape, size, category) change across episodes. The focus here is to evaluate the effectiveness of a single multi-task agent trained on all tasks and variants.

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

4.2 Simulation Results

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

Baseline Methods. We study the effectiveness of our problem formulation by benchmarking against two baselines: Image-BC and C2FARM-BC . Image-BC is an image-to-action agent similar to BC-Z [14]. Following BC-Z we use FiLM [69] for language conditioning, but the CNN vision encoders take in RGB-D images instead of just RGB. C2FARM-BC is a 3D fully-convolutional network by James et al. [25] that has achieved state-of-the-art results on RLBench tasks. Similar to our agent, C2FARM-BC also detects actions in a voxelized space, however it uses a coarse-to-fine-grain scheme to detect actions at two-levels of voxelization: 32^3 voxels with a 1^3m grid, and 32^3 voxels with a 0.15^3m grid after "zooming in" from the first level. Note that at the finest level C2FARM-BC has a higher resolution (0.47cm) than PERACT (1cm). We use the same ConvNet architecture as James et al. [25], but instead of training it with RL, we do BC with cross-entropy loss (from Section 3.4) and also condition it with language goals like in LingUNets [70, 16].

Multi-Task Performance. Table 1 compares the performance of Image-BC and C2FARM-BC against PERACT. With insufficient demonstrations, Image-BC has near zero performance on most tasks. Image-BC is disadvantaged with single-view observations and has to learn hand-eye coordination from scratch. In contrast, PERACT's voxel-based formulation naturally allows for in-

	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	4	4	4	0	0	0	0	0	20	8	0	8	0	0	0	0	0	0
C2FARM-BC [25]	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	0	0	0	4	0	0	0	0	0	0	4	0	0	0	0	0	0	0
C2FARM-BC [25]	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 2. **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 complete succces. PERACT outperforms C2FARM-BC [25], the most competitive baseline, with an average improvement of $+168\%$ with 10 demonstrations and $+266\%$ with 100 demonstrations across all tasks.

tegrating 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 $+\mathbf{168\%}$ **with 10 demonstrations and** $+\mathbf{266\%}$ **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. 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. See the simulation rollouts in the supplementary video to get a sense of the complexity of these evaluation tasks.

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.

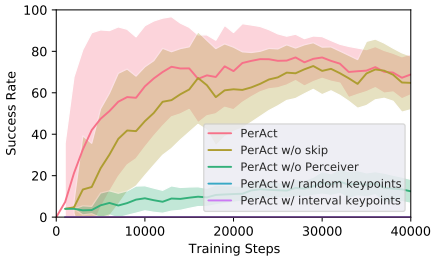

Figure 3. **Ablation Experiments.** Success rate of PERACT after ablating key components.

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.

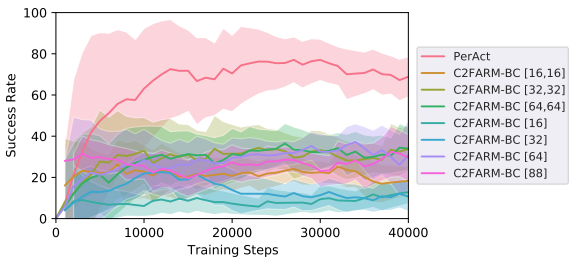

Figure 4. **Global vs. Local Receptive Field Experiments.** Success rates of PERACT against various C2FARM-BC [25] baselines

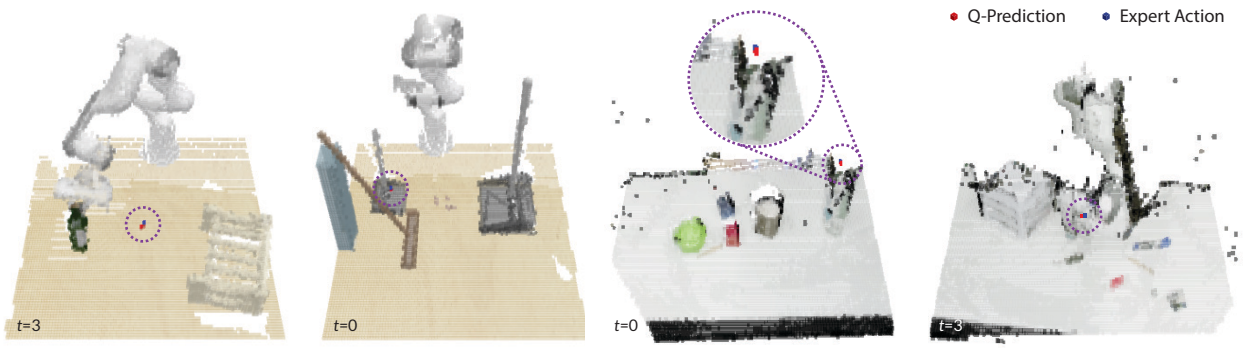

Figure 5. **Q-Prediction Examples**: Qualitative examples of translation $\mathcal{Q}$-Predictions from PERACT along with expert actions, highlighted with dotted-circles. The left two are simulated tasks, and the right two are real-world tasks. See Appendix H for more examples.

4.3 Global vs. Local Receptive Fields

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"*, *"open the middle drawer"*, and *"open the bottom drawer"*, and with a limited receptive field it is hard to distinguish the drawer handles, which are all visually identical. Figure 4 reports PERACT and C2FARM-BC agents trained with 100 demonstrations. Although the `open drawer` tasks can be solved with fewer demonstrations, here we want to ensure that insufficient data is not an issue. We include several versions of C2FARM-BC with different voxelization schemes. For instance, $[16, 16]$ indicates two levels of 16^3 voxel grids at 1m^3 and 0.15m^3, respectively. And $[64]$ indicates a single level of a 64^3 voxel grid without the coarse-to-fine-grain scheme. PERACT is the only agent that achieves $> 70\%$ success, whereas all C2FARM-BC versions perform at chance with $\sim 33\%$, indicating that the global receptive field of the Transformer is crucial for solving the task.

4.4 Real-Robot Results

We also validated our results with real-robot experiments on a Franka Emika Panda. See Appendix D for setup details. Without any sim-to-real transfer or pre-training, we trained a multi-task PERACT agent from scratch on 7 tasks with a total of just 53 demonstrations. See the supplementary video for qualitative results that showcase the diversity of tasks and robustness to scene changes. Table 3 reports success rates from small-scale evaluations. Similar to the simulation results, we find that PERACT is able to achieve $> 65\%$ success on simple short-horizon tasks like pressing hand-sanitizers from just a handful number of demonstrations. The most common failures involved predicting incorrect gripper open actions, which often lead the agent into unseen states. This could be addressed in future works by using HG-DAgger style approaches to correct the agent [14]. Other issues included the agent exploiting biases in the dataset like in prior work [16]. This could be addressed by scaling up expert data with more diverse tasks and task variants.

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

5 Conclusion and Limitations

We presented PERACT, a Transformer-based multi-task agent for 6-DoF manipulation. Our experiments with both simulated and real-world tasks indicate that the right problem formulation, i.e., detecting voxel actions, makes a substantial difference in terms of data efficiency and robustness.

While PERACT is quite capable, extending it to dexterous continuous control remains a challenge. PERACT is at the mercy of a sampling-based motion-planner to execute discretized actions, and is not easily extendable to N-DoF actuators like multi-fingered hands. See Appendix J for an extended discussion on PERACT's limitations. But overall, we are excited about scaling up robot learning with Transformers by focusing on *diverse* rather than narrow multi-task data for robotic manipulation.

References

[1] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeuRIPS)*, 2017.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Neural Information Processing Systems (NeurIPS)*, 2020.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2020.

[5] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

[6] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog*, 2, 2019.

[7] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. Hinton. Pix2seq: A language modeling framework for object detection. *arXiv preprint arXiv:2109.10852*, 2021.

[8] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *In Neural Information Processing Systems (NeurIPS)*, 2021.

[9] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[10] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

[11] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning (ICML)*, 2021.

[12] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2020.

[13] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[14] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning (CoRL)*, 2021.

[15] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[16] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *In Conference on Robot Learning (CoRL)*, 2021.

[17] J. J. Gibson. *The ecological approach to visual perception: classic edition*. Psychology Press, 2014.

[18] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[19] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018.

[20] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis. Single image 3d object detection and pose estimation for grasping. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[21] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. In *2017 IEEE international conference on robotics and automation (ICRA)*, 2017.

[22] X. Deng, Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox. Self-supervised 6d object pose estimation for robot manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[23] C. Xie, Y. Xiang, A. Mousavian, and D. Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on Robot Learning (CoRL)*, 2020.

[24] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, and J. Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.

[25] S. James, K. Wada, T. Laidlow, and A. J. Davison. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. In *Computer Vision and Pattern Recognition (CVPR)*, 2022.

[26] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *Conference on Robot Learning (CoRL)*, 2018.

[27] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel. Learning to Manipulate Deformable Objects without Demonstrations. In *Robotics: Science and Systems (RSS)*, 2020.

[28] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[29] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[30] S. Song, A. Zeng, J. Lee, and T. Funkhouser. Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[31] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox. 6-dof grasping for target-driven object manipulation in clutter. In *International Conference on Robotics and Automation (ICRA)*, 2020.

[32] A. Mousavian, C. Eppner, and D. Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *International Conference on Computer Vision (ICCV)*, 2019.

[33] Z. Xu, H. Zhanpeng, and S. Song. Umpnet: Universal manipulation policy network for articulated objects. *Robotics and Automation Letters (RA-L)*, 2022.

[34] A. Simeonov, Y. Du, A. Tagliasacchi, J. B. Tenenbaum, A. Rodriguez, P. Agrawal, and V. Sitzmann. Neural descriptor fields: Se (3)-equivariant object representations for manipulation. *arXiv preprint arXiv:2112.05124*, 2021.

[35] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

[36] W. Yuan, C. Paxton, K. Desingh, and D. Fox. Sornet: Spatial object-centric representations for sequential manipulation. In *In Conference on Robot Learning (CoRL)*. PMLR, 2021.

[37] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[38] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *In International Conference on Computer Vision (ICCV)*, 2021.

[39] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. *Neural information processing systems (NeurIPS)*, 2021.

[40] K.-H. Lee, O. Nachum, M. Yang, L. Lee, D. Freeman, W. Xu, S. Guadarrama, I. Fischer, E. Jang, H. Michalewski, et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022.

[41] H. M. Clever, A. Handa, H. Mazhar, K. Parker, O. Shapira, Q. Wan, Y. Narang, I. Akinola, M. Cakmak, and D. Fox. Assistive tele-op: Leveraging transformers to collect robotic task demonstrations. *arXiv preprint arXiv:2112.05129*, 2021.

[42] R. Yang, M. Zhang, N. Hansen, H. Xu, and X. Wang. Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers. *arXiv preprint arXiv:2107.03996*, 2021.

[43] D. S. Chaplot, D. Pathak, and J. Malik. Differentiable spatial planning using transformers. In *International Conference on Machine Learning (ICML)*, 2021.

[44] J. J. Johnson, L. Li, A. H. Qureshi, and M. C. Yip. Motion planning transformers: One model to plan them all. *arXiv preprint arXiv:2106.02791*, 2021.

[45] S. Dasari and A. Gupta. Transformers for one-shot visual imitation. *arXiv preprint arXiv:2011.05970*, 2020.

[46] H. Kim, Y. Ohmura, and Y. Kuniyoshi. Transformer-based deep imitation learning for dual-arm robot manipulation. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.

[47] Y. Han, R. Batra, N. Boyd, T. Zhao, Y. She, S. Hutchinson, and Y. Zhao. Learning generalizable vision-tactile robotic grasping strategy for deformable objects via transformer. *arXiv preprint arXiv:2112.06374*, 2021.

[48] M. Shridhar and D. Hsu. Interactive visual grounding of referring expressions for human-robot interaction. In *Robotics: Science and Systems (RSS)*, 2018.

[49] C. Matuszek, L. Bo, L. Zettlemoyer, and D. Fox. Learning from unscripted deictic gesture and language for human-robot interactions. In *AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[50] M. Bollini, S. Tellex, T. Thompson, N. Roy, and D. Rus. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics*, pages 481–495. Springer, 2013.

[51] D. K. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research (IJRR)*, 2016.

[52] Y. Bisk, D. Yuret, and D. Marcu. Natural language communication with robots. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2016.

[53] J. Thomason, S. Zhang, R. J. Mooney, and P. Stone. Learning to interpret natural language commands through human-robot dialog. In *Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[54] J. Hatori, Y. Kikuchi, S. Kobayashi, K. Takahashi, Y. Tsuboi, Y. Unno, W. Ko, and J. Tan. Interactively picking real-world objects with unconstrained spoken language instructions. In *International Conference on Robotics and Automation (ICRA)*, 2018.

[55] Y. Chen, R. Xu, Y. Lin, and P. A. Vela. A Joint Network for Grasp Detection Conditioned on Natural Language Commands. *arXiv:2104.00492 [cs]*, Apr. 2021.

[56] V. Blukis, R. A. Knepper, and Y. Artzi. Few-shot object grounding for mapping natural language instructions to robot control. In *Conference on Robot Learning (CoRL)*, 2020.

[57] C. Paxton, Y. Bisk, J. Thomason, A. Byravan, and D. Fox. Prospection: Interpretable plans from language by predicting the future. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[58] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2011.

[59] C. Lynch and P. Sermanet. Grounding language in play. *arXiv preprint arXiv:2005.07648*, 2020.

[60] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *arXiv preprint arXiv:2112.03227*, 2021.

[61] E. Johns. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In *International Conference on Robotics and Automation (ICRA)*, 2021.

[62] H. Moravec. Robot spatial perceptionby stereoscopic vision and 3d evidence grids. *Perception*, 1996.

[63] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, 1989.

[64] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning Transferable Visual Models From Natural Language Supervision. *arXiv:2103.00020*, 2021.

[65] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[66] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

[67] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[68] S. James, M. Freese, and A. J. Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.

[69] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 2018.

[70] D. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkhin, and Y. Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. In *2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[71] R. Ranftl, A. Bochkovskiy, and V. Koltun. Vision transformers for dense prediction. In *International Conference on Computer Vision (ICCV)*, pages 12179–12188, 2021.

[72] S. James and P. Abbeel. Coarse-to-fine q-attention with learned path ranking. *arXiv preprint arXiv:2204.01571*, 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 ___ 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 ___ 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 4. **Language-Conditioned Tasks in RLBench** [15].

A Task Details

Setup. Our simulated experiments are set in RLBench [15]. We select 18 out of 100 tasks that involve at least two or more variations to evaluate the multi-task capabilities of agents. While PERACT could be easily applied to more RLBench tasks, in our experiments, we were specifically interested grounding diverse language instructions, rather than learning one-off policies for single-variation tasks like *"[always] take off the saucepan lid"*. Some tasks were modified to include additional variations. See Table 4 for an overview. We report average keyframes extracted from the method described in Section 3.2.

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

In the following sections, we describe each of 18 tasks in detail. We highlight tasks that were modified from the original RLBench [15] codebase[2] and describe what exactly was modified.

A.1 Open Drawer

Filename: `open_drawer.py`

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

Modified: No.

Objects: 1 drawer.

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

A.2 Slide Block

Filename: `slide_block_to_color_target.py`

[2] https://github.com/stepjam/RLBench

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

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

Objects: 1 block and 4 colored target squares.

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

A.3 Sweep to Dustpan

Filename: `sweep_to_dustpan_of_size.py`

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

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

Objects: 5 dirt particles and 2 dustpans.

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

A.4 Meat Off Grill

Filename: `meat_off_grill.py`

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

Modified: No.

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

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

A.5 Turn Tap

Filename: `turn_tap.py`

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

Modified: No.

Objects: 1 faucet with 2 handles.

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

A.6 Put in Drawer

Filename: `put_item_in_drawer.py`

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

Modified: No.

Objects: 1 block and 1 drawer.

Success Metric: The block is inside the specified drawer.

A.7 Close Jar

Filename: `close_jar.py`

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

Modified: No.

Objects: 1 block and 2 colored jars.

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

A.8 Drag Stick

Filename: `reach_and_drag.py`

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

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

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

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

A.9 Stack Blocks

Filename: `stack_blocks.py`

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

Modified: No.

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

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

A.10 Screw Bulb

Filename: `light_bulb_in.py`

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

Modified: No.

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

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

A.11 Put in Safe

Filename: `put_money_in_safe.py`

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

Modified: No.

Objects: 1 stack of money, and 1 safe.

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

A.12 Place Wine

Filename: `place_wine_at_rack_location.py`

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

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

Objects: 1 wine bottle, and 1 wooden rack.

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

A.13 Put in Cupboard

Filename: `put_groceries_in_cupboard.py`

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

Modified: No.

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

Success Metric: The specified object is inside the cupboard.

A.14 Sort Shape

Filename: `place_shape_in_shape_sorter.py`

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

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

Objects: 5 shapes, and 1 sorter.

Success Metric: The specified shape is inside the sorter.

A.15 Push Buttons

Filename: `push_buttons.py`

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

Modified: No.

Objects: 3 buttons.

Success Metric: All the specified buttons were pressed.

A.16 Insert Peg

Filename: `insert_onto_square_peg.py`

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

Modified: No.

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

Success Metric: The square is on the specified spoke.

A.17 Stack Cups

Filename: `stack_cups.py`

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

Modified: No.

Objects: 3 tall cups.

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

A.18 Place Cups

Filename: `place_cups.py`

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

Modified: No.

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

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

B PERACT Details

In this section, we provide implementation details for PERACT.

Input Observation. Following James et al. [25], our input voxel observation is a 100^3 voxel grid with 10 channels: $\mathbb{R}^{100\times100\times100\times10}$. The grid is constructed by fusing calibrated pointclouds with PyTorch's `scatter_` function[3]. The 10 channels are composed of: 3 RGB, 3 point, 1 occupancy, and 3 position index values. The RGB values are normalized to a zero-mean distribution. The point values are Cartesian coordinates in the robot's coordinate frame. The occupancy value indicates if a voxel is occupied or empty. The position index values represent the 3D location of the voxel with respect to the 100^3 grid. In addition to the voxel observation, the input also includes proprioception data with 4 scalar values: gripper open, left finger joint position, right finger joint position, and timestep (of the action sequence).

Input Language. The language goals are encoded with CLIP's language encoder [64]. We use CLIP's tokenizer to preprocess the sentence, which always results in an input sequence of 77 tokens (with zero-padding). These tokens are encoded with the language encoder to produce a sequence of dimensions $\mathbb{R}^{77\times512}$.

Preprocessing. The voxel grid is encoded with a 3D convolution layer with a 1×1 kernel to upsample the channel dimension from 10 to 64. Similarly, the proprioception data is encoded with a linear layer to upsample the input dimension from 4 to 64. The encoded voxel grid is split into 5^3 patches through a 3D convolution layer with a kernel-size and stride of 5, which results in a patch tensor of dimensions $\mathbb{R}^{20\times20\times20\times64}$. The proprioception features are tiled in 3D to match the dimensions of the patch tensor, and concattenated along the channel to form a tensor of dimensions $\mathbb{R}^{20\times20\times20\times128}$. This tensor is flattened into a sequence of dimensions $\mathbb{R}^{8000\times128}$. The language features are downsampled with a linear layer from 512 to 128 dimensions, and then appended to the tensor to form the final input sequence to the Perceiver Transformer, which of dimensions $\mathbb{R}^{8077\times128}$. We also add learned positional embeddings to the input sequence. These embeddings are represented with trainable `nn.Parameter`(s) in PyTorch.

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

[3] https://pytorch.org/docs/stable/generated/torch.Tensor.scatter_.html

Perceiver first computes cross-attention between the input sequence and the set of latent vectors of dimensions $\mathbb{R}^{2048\times512}$. These latents are randomly initialized and trained end-to-end. The latents are encoded with 6 self-attention layers, and then cross-attended with the input to output a sequence that matches the input-dimensions. This output is upsampled with a 3D convolution layer and tri-linear upsampling to form a voxel feature grid with 64 channels: $\mathbb{R}^{100\times100\times100\times64}$. Our implementation of Perceiver is based on an existing open-source repository[4].

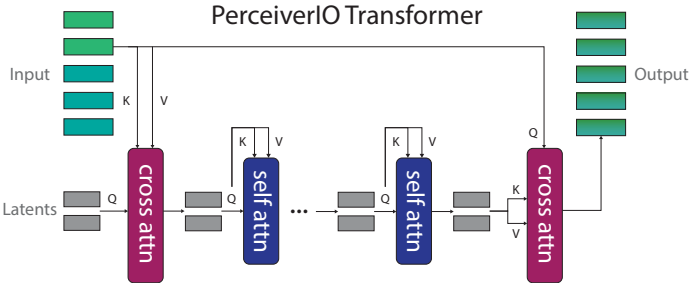

Figure 6. **Perceiver Transformer Architecture.** Perceiver is a latent-space transformer. Q, K, V represent queries, keys, and values, respectively. We use 6 self-attention layers in our implementation.

Decoding. For translation, the voxel feature grid is decoded with a 3D convolution layer with a 1×1 kernel to downsample the channel dimension from 64 to 1. This tensor is the translation $\mathcal{Q}$-function of dimensions $\mathbb{R}^{100\times100\times100\times1}$. For rotation, gripper open, and collision avoidance actions, the voxel feature grid is max-pooled along the 3D dimensions to form a vector of dimensions $\mathbb{R}^{1\times64}$. This vector is decoded with three independent linear layers to form the respective $\mathcal{Q}$-functions for rotation, gripper open, and collision avoidance. The rotation linear layer outputs logits of dimensions $\mathbb{R}^{216}$ (72 bins of 5 degree increments for each of the three axes). The gripper open and collide linear layers output logits of dimensions $\mathbb{R}^2$.

Our codebase is built on the ARM repository[5] by James et al. [25]. We will release our PyTorch implementation upon publication.

C Evaluation Workflow

C.1 Simulation

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

C.2 Real-Robot

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

[4] https://github.com/lucidrains/perceiver-pytorch
[5] https://github.com/stepjam/ARM

D Robot Setup

D.1 Simulation

All simulated experiments use the four camera setup illustrated in Figure 7. The front, left shoulder, and right shoulder cameras, are static, but the wrist camera moves with the end-effector. We did not modify the default camera poses from RLBench [15]. These poses maximize coverage of the tabletop, while minimizing occlusions caused by the moving arm. The wrist camera in particular is able to provide high-resolution observations of small objects like handles.

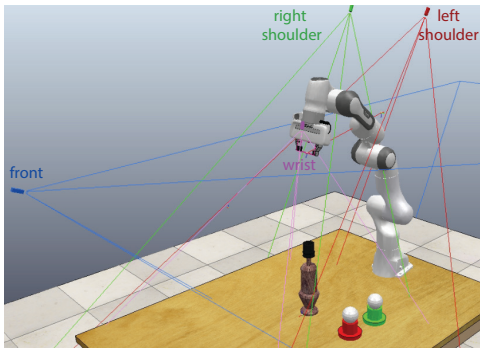

Figure 7. **Simulated Setup.** The four camera setup: front, left shoulder, right shoulder, and on the wrist.

D.2 Real-Robot

Hardware Setup. The real-robot experiments use a Franka Panda manipulator with a parallel-gripper. For perception, we use a Kinect-2 RGB-D camera mounted on a tripod, at an angle, pointing towards the tabletop. See Figure D for reference. We tried setting-up multiple Kinects for multi-view observations, but we could not fix the interference issue caused by multiple Time-of-Flight sensors. The Kinect-2 provides RGB-D images of resolution 512×424 at 30Hz. The extrinsics between the camera and robot base-frame are calibrated with the `easy_handeye` package[6]. We use an ARUCO[7] AR marker mounted on the gripper to aid the calibration process.

Data Collection. We collect demonstrations with an HTC Vive controller. The controller is a 6-DoF tracker that provides accurate poses with respect to a static base-station. These poses are displayed as a marker on RViz[8] along with the real-time RGB-D pointcloud from the Kinect-2. A user specifies target poses by using the marker and pointcloud as reference. These target poses are executed with a motion-planner. We use Franka ROS and MoveIt[9], which by default uses an RRT-Connect planner.

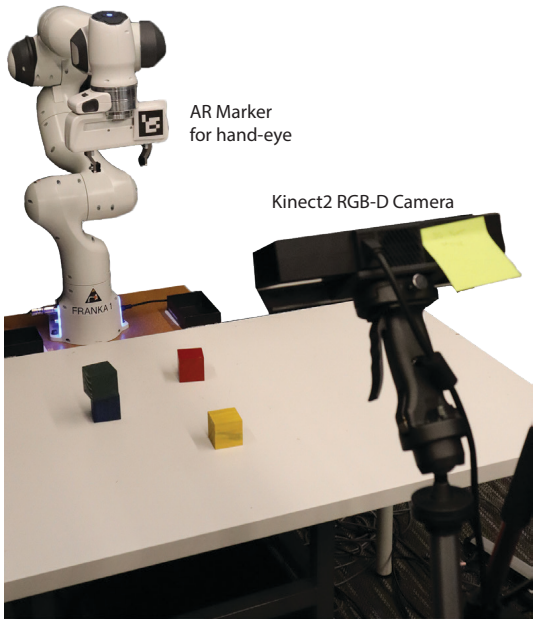

Figure 8. **Real-Robot Setup** with Kinect-2 and Franka Panda.

Training and Execution. We train a PERACT agent from scratch with 53 demonstrations. The training samples are augmented with ± 0.125m translation perturbations and $\pm 45^\circ$ yaw rotation perturbations. We train on 8 NVIDIA P100 GPUs for 2 days. During evaluation, we simply chose the last checkpoint from training (since we did not collect a validation set for optimization).

[6] https://github.com/IFL-CAMP/easy_handeye
[7] https://github.com/pal-robotics/aruco_ros
[8] http://wiki.ros.org/rviz
[9] http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/

E Data Augmentation

PERACT's voxel-based formulation naturally allows for data augmentation with SE(3) transformations. 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 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 range of $[0°, 0°, \pm 45°]$. The $45°$ limit ensures that the perturbed rotations do not go beyond what is kinematically reachable for the Franka arm. We did experiment with pitch and roll perturbations, but they substantially lengthened the training time. Any perturbation that pushed the discretized action outside the observation voxel grid was discarded. See the bottom row of Figure 10 for examples of data augmentation.

F Demo Augmentation

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. In this illustration, $\mathbf{k}_1$ and $\mathbf{k}_2$ are two keyframes that were extracted from the method described in Section 3.2. The orange circles indicate datapoints whose RGB-D observations are paired with the next keyframe action.

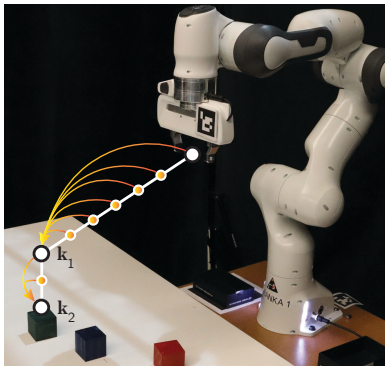

Figure 9. **Keyframes and Demo Augmentation.**

G Sensitivity Analysis

In Table 5, we investigate three factors that affect PERACT's performance: rotation data augmentation, number of Perceiver latents, and voxelization resolution. All multi-task agents were trained with 100 demonstrations per task and evaluated on 25 episodes per task. To briefly summarize these results: (1) $45°$ yaw perturbations improve performance on tasks with lots of rotation variations like `stack blocks`, but also worsen performance on tasks with constrained rotations like `place wine`. (2) PERACT with just 512 latents is competitive with (and sometimes even better than) the default agent with 2048 latents, which showcases the compression capability of the Perceiver architecture. (3) Coarse grids like 32^3 are sufficient for some tasks, but high-precision tasks like `sort shape` need higher resolution voxelization.

Table 5. **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

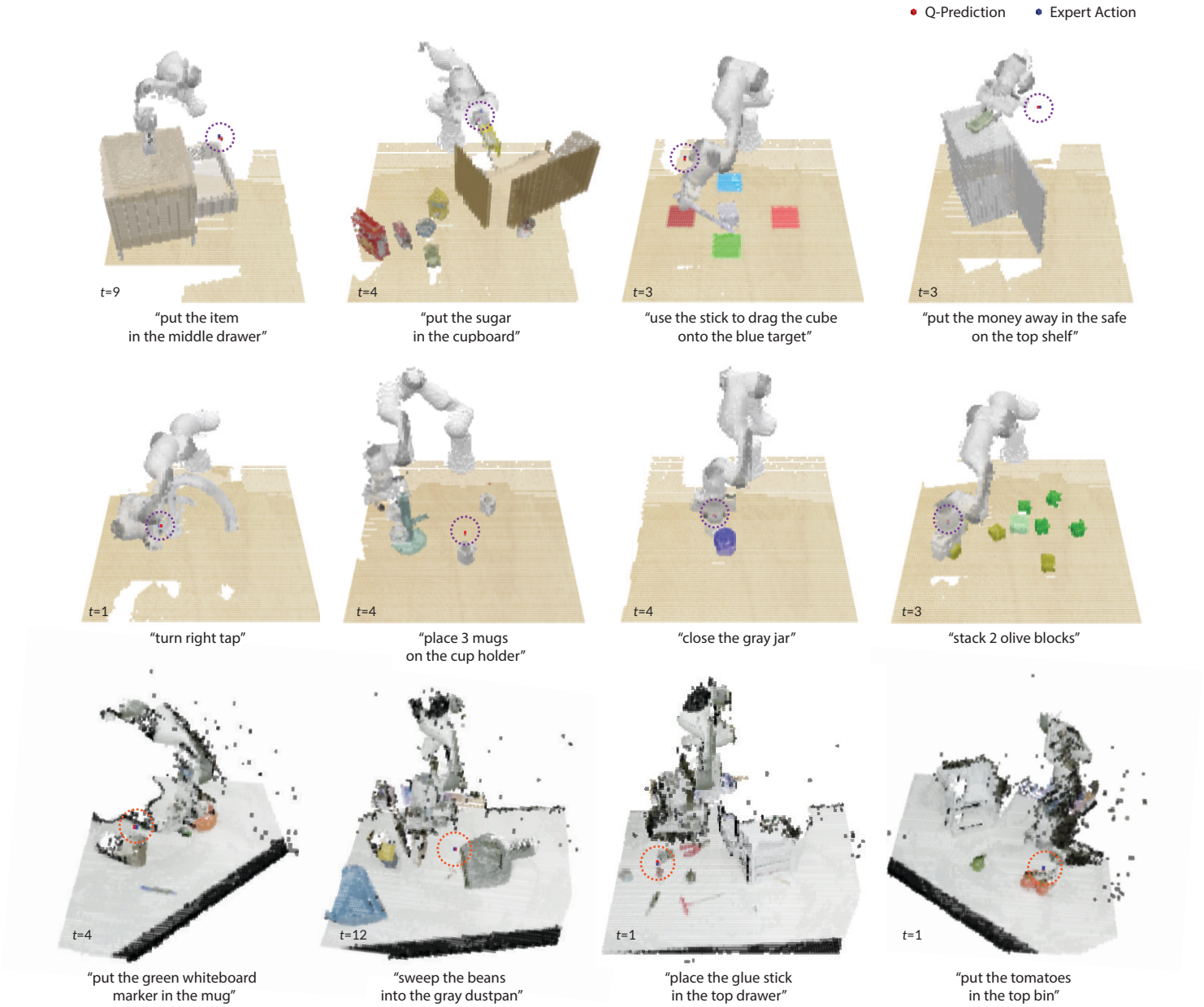

Figure 10. **Additional Q-Prediction Examples.** $\mathcal{Q}$-Prediction examples from PERACT. The top two rows are from simulated tasks without any data augmentation perturbations, and the bottom row is from real-world tasks with translation and yaw-rotation perturbations.

H Additional Q-Prediction Examples

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

I Things that did not work

In this section, we describe things we tried but did not work in practice.

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

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

Pre-trained vision features. Following CLIPort [16], we tried using pre-trained vision features from CLIP [64], instead of raw RGB values, to bootstrap learning and also to improve generalization to unseen objects. We ran CLIP's ResNet50 on each of the 4 RGB frames, and upsampled features with shared decoder layers in a UNet fashion. But we found this to be extremely slow, especially since the ResNet50 and decoder layers need to be run on 4 independent RGB frames. With this additional overhead, training multi-task agents would have taken substantially longer than 16 days. Future works could experiment with methods for pre-training the decoder layers on auxiliary tasks, and pre-extracting features for faster training.

Upsampling at multiple self-attention layers. Inspired by Dense Prediction Transformers (DPT) [71], we tried upsampling features at multiple self-attention layers in the Perceiver Transformer. But this did not work at all; perhaps the latent-space self-attention layers of Perceiver are substantially different to the full-input self-attention layers of ViT [4] and DPT [71].

Extreme rotation augmentation. In addition to yaw rotation perturbations, we also tried perturbing the pitch and roll. While PERACT was still able to learn policies, it took substantially longer to train. It is also unclear if the default latent size of $\mathbb{R}^{2048\times512}$ is appropriate for learning 6-DoF polices with such extreme rotation perturbations.

J Limitations and Risks

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

Sampling-Based Motion Planner. PERACT relies on a sampling-based motion planner to execute discretized actions. This puts PERACT at the mercy of randomized planner to reach poses. While this issue did not cause any major problems with the tasks in our experiments, a lot of other tasks are sensitive to the paths taken to reach poses. For instance, pouring water into a cup would require a smooth path for tilting the water container appropriately. This could be addressed in future works by using a combination of learned and sampled motion paths [72].

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

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

Generlization to Novel Objects. PERACT is an agent trained from scratch. Although we did not explicitly test for generalization to unseen objects, it might be feasible to train PERACT's action-

classifier on a broad range of objects and evaluate its ability to handle novel objects akin to how instance-segmentors and object-detectors are used. Alternatively, pre-trained vision features from multi-modal encoders like CLIP [64] or R3M [35] could be used to boostrap learning.

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

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

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

Deployment Risks. PERACT is an end-to-end framework for 6-DoF manipulation. Unlike some methods in Task-and-Motion-Planning (TAMP) that can provide theoretical guarantees on task completion, PERACT is a purely reactive system, whose performance can only be evaluated through empirical means. As such, for safe deployment in real-world applications, keeping humans in the loop both during training and testing, might help in mitigating some of these risks.