Act3D: Infinite Resolution Action Detection Transformer for Robot Manipulation

Anonymous Author(s)
Affiliation
Address
email

Abstract: 3D perceptual representations are well suited for robot manipulation as they easily encode occlusions and simplify spatial reasoning. Many manipulation tasks require high spatial precision in end-effector pose prediction, typically demanding high-resolution 3D perceptual grids that are computationally expensive to process. As a result, most manipulation policies operate directly in 2D, foregoing 3D inductive biases. In this paper, we propose Act3D, a manipulation policy Transformer that casts 6-DoF keypose prediction as 3D detection with adaptive spatial computation. It takes as input 3D feature clouds projected from one or more camera views, iteratively samples 3D point grids in free space in a coarse-to-fine manner, featurizes them using relative spatial attention to the physical feature cloud, and selects one as the end-effector position. Act3D sets a new state-of-the-art in RLbench, an established robot manipulation benchmark. Specifically, our model achieves 10% absolute improvement over the previous SOTA multi-view policy on 74 RLbench tasks and 22% absolute improvement with 3x less compute over the previous SOTA 3D policy. In thorough ablations, we show the importance of relative spatial attention, large-scale vision-language pre-trained 2D backbones, and weight tying across coarse-to-fine attentions. Our code and models will be publicly available upon publication.

Keywords: Learning from Demonstrations, Manipulation, Transformers

1 Introduction

Solutions to many robotic manipulation tasks can be represented as a sequence of 6-DoF robot end-effector poses. Many recent methods train manipulation policies to predict such poses directly from 2D videos and language instructions using supervision from demonstrations [1, 2, 3, 4, 5]. However, these methods are typically sample inefficient, often requiring thousands of trajectories, and cannot easily generalize across viewpoints and environments. Transporter networks [6] recently reformulated 4-DoF keypose prediction as pixel classification in a top-down scene image, inspired by object detection in computer vision [7, 8, 9]. This design choice of detecting end-effector poses in the scene using local features instead of regressing them from aggregated scene features, which we will call *action detection*, dramatically increased sample efficiency. However, Transporter Networks are limited to top-down 2D worlds and 4-DoF end-effector poses.

A core challenge in detecting actions for general 6-DoF manipulation is that end-effector 3D positions need to be predicted on 3D points in free space, not on 3D physical scene points. For example, end-effector 6 DoF poses relevant for a task, which we will call keyposes [10, 11], can be pre-grasp poses, back-off poses for articulated object interactions, or transition poses between different parts of a task. While it is straightforward to featurize 2D pixels or 3D physical points — we can featurize pixels with 2D backbones and back-project to 3D or use a 3D point cloud transformer [12] — it is less clear how to efficiently featurize points in free space to detect one as the end-effector position. 3D voxelization at high resolution is computationally demanding since we cannot use sparse 3D con-

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

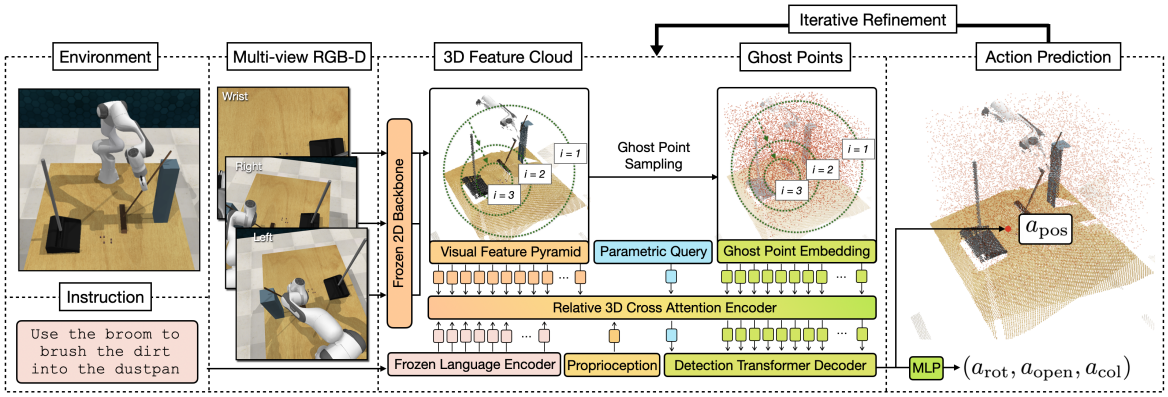


Figure 1: **Act3D architecture**. Act3D is a language-conditioned end-effector 6-DoF keypose predictor that learns 3D perceptual representations of arbitrary spatial resolution via recurrent coarse-to-fine 3D point sampling and featurization. Act3D featurizes multi-view RGB images with a pre-trained 2D backbone and lifts them in 3D using depth to obtain a multi-scale 3D scene feature cloud. It then iteratively predicts 3D foci of attention in the free space, samples 3D point grids in their vicinity, and featurizes the sampled 3D points using relative cross-attention to the physical scene feature cloud, language tokens, and proprioception. Act3D detects the 3D point that corresponds to the next best end-effector position using a detection Transformer head, and regresses the rotation, end-effector opening, and planner collision avoidance from the decoder's parametric query.

volutions [13, 14]: we do not know ahead of time which voxels will remain empty or would need to be featurized because they would contain the next end-effector pose. Recent work of PerAct [1] featurizes *all* 3D voxels (occupied or not) using the latent set bottlenecked self-attention operation of Perceiver [15], which is computationally expensive. Other methods work around this issue by avoiding featurizing points in free space and instead detecting a contact point and then regressing an offset from this contact point towards the end-effector predicted position [2, 16, 17]. This is a reasonable design choice but it does not fully exploit the action detection inductive bias.

In this paper, we propose Act3D, a Transformer policy architecture for language-conditioned multi-task robot manipulation that casts 6-DoF end-effector keypose prediction as 3D detection with adaptive spatial computation. Act3D learns 3D perceptual representations of arbitrary spatial resolution via recurrent coarse-to-fine 3D point grid sampling and featurization. It first computes a physical scene 3D feature cloud by lifting 2D pre-trained features from one or more views using sensed depth. At each iteration, the model then samples 3D point grids in free space and featurizes them using relative spatial cross-attention [18] to the 3D physical feature cloud. The featurized 3D points are classified with a detection Transformer head [9, 19] to predict the grid center for the next iteration. All iterations share attention weights. Act3D detects the 3D point that corresponds to the end-effector's 3D position using a detection transformer head [9, 19], then regresses the 3D rotation and opening of the end-effector from the contextualized parametric query. At inference time, we can trade-off compute for higher spatial precision and task performance by sampling more points in free space than the model ever saw at training time.

We test Act3D in RLBench [20], an established benchmark for learning diverse robot manipulation policies from demonstrations. We set a new state-of-the-art in the benchmark in both single-task and multi-task settings. Specifically, we achieve a 10% absolute improvement over prior SOTA on the single-task setting introduced by HiveFormer [2] with 74 tasks and a 22% absolute improvement over prior SOTA in the multi-task setting introduced by PerAct [1] with 18 tasks and 249 variations. We also validate our approach on a Franka Panda with a multi-task agent trained from scratch on 8 real-world tasks with a total of just 100 demonstrations (see Figure 2). In thorough ablations, we show the importance of relative spatial attention, large-scale vision-language pre-trained 2D backbones, and weight tying across coarse-to-fine attentions.

2 Related Work

Learning robot manipulation from demonstrations Many recent work train multi-task manipulation policies that leverage Transformer architectures [1, 2, 3, 5, 21, 22] to predict robot actions from video input and language instructions. End-to-end image-to-action policy models, such as RT-1 [5], GATO [22], BC-Z [23], and InstructRL [3], directly predict 6-DoF end-effector poses from 2D video and language inputs. They require many thousands of demonstrations to learn spatial reasoning and generalize to new scene arrangements and environments. Transporter networks [6] and their subsequent variants [24, 25] formulate 4-DoF end-effector pose prediction as pixel classification in 2D overhead images. Their *action detection* inductive bias — parametrizing the action implicitly [26] by detecting end-effector poses in the scene using local features with translation and rotation equivariances [27] — dramatically increased sample efficiency over previous methods that regress end-effector poses by aggregating global scene features. However, they are limited to top-down 2D planar worlds with simple pick-and-place primitives. 3D policy models of C2F-ARM [4] and PerAct [1] voxelize the robot's workspace and are trained to detect the 3D voxel that contains the next end-effector keypose. Spatially precise 3D pose prediction requires the 3D voxel grid to be high resolution, which comes at a high computational cost. C2F-ARM [4] uses a coarse-to-fine voxelization in convolutional grids to handle computational complexity, while PerAct [1] uses Perceiver's latent bottleneck [15] to avoid voxel-to-voxel self-attention operations. Act3D avoids 3D voxelization altogether and instead represents the scene as a 3D feature cloud. It samples 3D points in the empty workspace and featurizes them using cross-attentions to the physical 3D point features.

Feature pre-training for robot manipulation Many 2D policy architectures bootstrap learning from demonstrations from frozen or finetuned 2D image backbones [28, 29, 23, 30] to increase experience data sample efficiency. Pretrained vision-language backbones can enable generalization to new instructions, objects, and scenes [31, 25]. In contrast, SOTA 3D policy models are typically trained from scratch from colored point clouds input [1, 4]. Act3D uses CLIP pre-trained 2D backbones [32] to featurize 2D image views and lifts the 2D features in 3D using depth information. We show that 2D feature pretraining gives a considerable performance boost over training from scratch.

Relative attention layers Relative attentions have shown improved performance in many 2D visual understanding tasks and language tasks [33, 34]. Rotary embeddings [35] implement relative attention efficiently by casting it as an inner-product in an extended position feature space. In 3D, relative attention is imperative as the coordinate system is arbitrary. 3D relative attentions have been used before in 3D Transformer architectures for object detection and point labelling [36, 37]. We show in Section 4 that relative attentions significantly boost performance of our model.

3 Act3D

The architecture of Act3D is shown in Figure 1. It is a Transformer policy that, at each timestep t, predicts a 6-DoF end-effector pose from one or more RGB-D images, a language instruction, and proprioception information regarding the robot's current end-effector pose. The key idea is to *detect* 6 DoF end-effector future poses in the robot's workspace by learning 3D perceptual representations of free space of arbitrary spatial resolution via recurrent coarse-to-fine 3D point grid sampling and featurization. 3D point candidates (which we will call ghost points) are sampled, featurized and scored iteratively through relative cross-attention [18] to the physical 3D scene feature cloud, lifted from 2D feature maps of the input image views.

Following prior work [11, 1, 2, 3], instead of predicting an end-effector pose at each timestep, we extract a set of *keyposes* that capture bottleneck end-effector poses in a demonstration. A pose is a keypose if (1) the end-effector changes state (something is grasped or released) or (2) velocities approach near zero (a common occurrence when entering pre-grasp poses or entering a new phase of a task). The prediction problem then boils down to predicting the next (best) keypose action

given the current observation. At inference time, Act3D iteratively predicts the next best keypose and reaches it with a motion planner, following previous works [1, 2].

We assume access to a dataset of n demonstration trajectories. Each demonstration is a sequence of observations $O = \{o_1, o_2, .., o_t\}$ paired with continuous actions $A = \{a_1, a_2, .., a_t\}$ and, optionally, a language instruction l that describes the task. Each observation o_t consists of RGB-D images from one or more camera views; more details are in Appendix 6.2. An action a_t consists of the 3D position and 3D orientation (represented as a quaternion) of the robot's end-effector, its binary open or closed state, and whether the motion planner needs to avoid collisions to reach the pose:

$$a = \{a_{\text{pos}} \in R^3, a_{\text{rot}} \in H, a_{\text{open}} \in \{0, 1\}, a_{\text{col}} \in \{0, 1\}\}$$

Next, we go into details on the modules of Act3D.

Visual and language encoder Our visual encoder map multi-view RGB-D images into a multi-scale 3D scene feature cloud. We use a large-scale pre-trained 2D feature extractor followed by a feature pyramid network [38] to extract multi-scale visual tokens for each camera view. We then lift these 2D visual tokens to 3D by interpolating their depth values. The language encoder featurizes instructions with a large-scale pre-trained language feature extractor. We use the CLIP ResNet50 [32] visual encoder and the corresponding language encoder to exploit their common vision-language feature space for interpreting instructions and referential grounding. Our pre-trained visual and language encoders are frozen, not finetuned, during training of Act3D.

Iterative ghost point sampling and featurization To enable precise and computationally tractable keypose detection, we sample, featurize and select ghost points iteratively, first coarsely across the entire workspace, then finely in the vicinity of the ghost point selected as the focus of attention in the previous iteration. The coarsest ghost points attend to a global coarse scene feature cloud, whereas finer ghost points attend to a local fine scene feature cloud.

Relative 3D cross-attentions We featurize each of the 3D ghost points and a parametric query (used to select via inner-product one of the ghost points as the next best end-effector position in the decoder) independently through cross-attentions to the multi-scale 3D scene feature cloud, language tokens, and proprioception. Featurizing ghost points independently, without self-attentions to one another, enables sampling more ghost points at inference time to improve performance, as we show in Section 4. Our cross-attentions use relative 3D position information and are implemented efficiently with rotary positional embeddings [18]. Given a point $\mathbf{p} = (x, y, z) \in R^3$ and its feature $\mathbf{x} \in R^d$, the rotary position encoding function $\mathbf{PE}$ is defined as:

$$\mathbf{PE}(\mathbf{p}, \mathbf{x}) = \mathbf{M}(\mathbf{p})\mathbf{x} = \begin{bmatrix} \mathbf{M}_1 & & \\ & \ddots & \\ & & \mathbf{M}_{d/6} \end{bmatrix} \mathbf{x}, \; \mathbf{M}_k = \begin{bmatrix} \cos x\theta_k & -\sin x\theta_k & 0 & 0 & 0 & 0 \\ \sin x\theta_k & \cos x\theta_k & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos y\theta_k & -\sin y\theta_k & 0 & 0 \\ 0 & 0 & \sin y\theta_k & \cos y\theta_k & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos z\theta_k & -\sin z\theta_k \\ 0 & 0 & 0 & 0 & \sin z\theta_k & \cos z\theta_k \end{bmatrix}$$

where $\theta_k = \frac{1}{10000^{6(k-1)/d}}$. The dot product of two positionally encoded features is then

$$\mathbf{PE}(\mathbf{p}_i, \mathbf{x}_i)^T \mathbf{PE}(\mathbf{p}_j, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{M}(\mathbf{p}_i)^T \mathbf{M}(\mathbf{p}_j)\mathbf{x}_j = \mathbf{x}_i^T \mathbf{M}(\mathbf{p}_j - \mathbf{p}_i)\mathbf{x}_j$$

which depends only on the relative positions of points $\mathbf{p}_i$ and $\mathbf{p}_j$.

Detection Transformer decoder Once ghost points and the parametric query are featurized, the detection transformer head scores ghost point tokens via inner product with the parametric query to select one as the next best end-effector position a_{pos}. We then regress the end-effector orientation a_{rot} and opening a_{open}, as well as whether the motion planner needs to avoid collisions to reach the pose a_{col}, from the parametric query with a simple multi-layer perceptron (MLP).

Training Act3D is trained supervised from input-action tuples from a dataset of manipulation demonstrations. These tuples are composed of RGB-D observations, language goals, and keypose actions $\{(o_1, l_1, k_1), (o_2, l_2, k_2), ...\}$. During training, we randomly sample a tuple and supervise

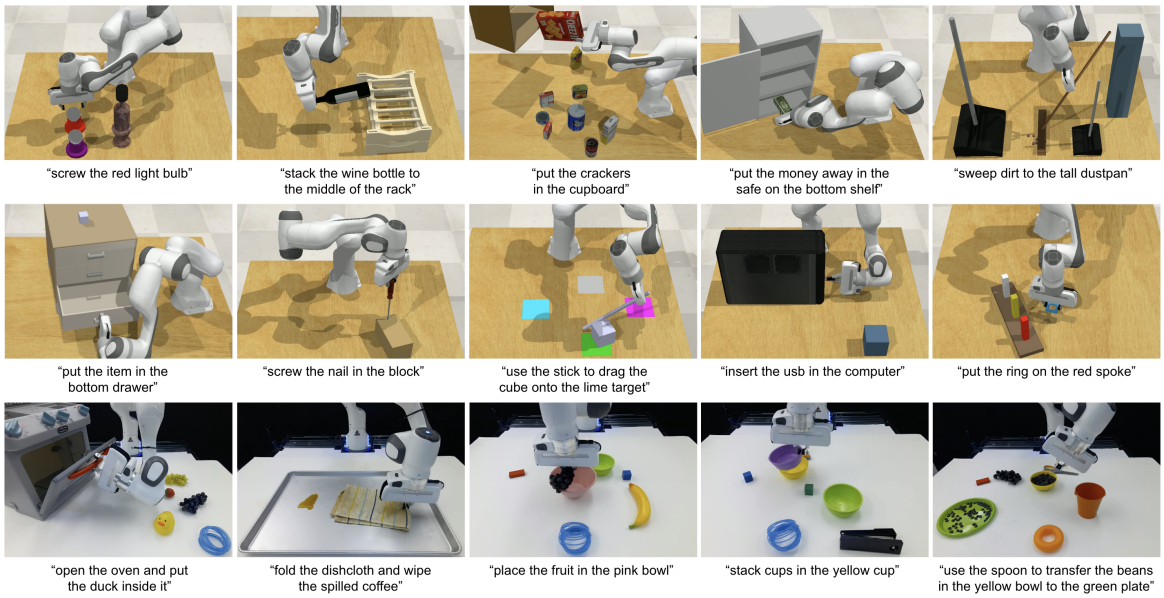


Figure 2: **Tasks.** We conduct experiments on 92 simulated tasks in RLBench [20] (only 10 shown), and 8 real-world tasks (only 5 shown). Please see the supplementary video for video results of our model in simulation and in the real world.

Act3D to predict the keypose action k given the observation and goal (o, l). We supervise position prediction a_{pos} at every round of coarse-to-fine with a softmax cross-entropy loss over ghost points, rotation prediction a_{rot} with a MSE loss on the quaternion prediction, and binary end-effector opening a_{open} and whether the planner needs to avoid collisions a_{col} with binary cross-entropy losses.

Implementation details We extract two feature maps per 256x256 input image view: 32x32 coarse visual tokens and 64x64 fine visual tokens. We use three ghost point sampling stages: first across the entire workspace (roughly 1 meter cube), then in a 16 centimeter diameter ball, and finally in a 4 centimeter diameter ball. The coarsest ghost points attend to a global coarse scene feature cloud (32x32xn_{cam} coarse visual tokens) whereas finer ghost points attend to a local fine scene feature cloud (the closest 32x32xn_{cam} out of the total 64x64xn_{cam} fine visual tokens). During training, we sample 1000 ghost points in total split equally across the three stages. At inference time, we can trade-off extra prediction precision and task performance for additional compute by sampling more ghost points than the model ever saw at training time ($10,000$ in our experiments). We'll show in ablations in Section 4 that our framework is robust to these hyper-parameters but tying weights across sampling stages and relative 3D cross-attention are both crucial for generalization. We use 2 layers of cross-attention and an embedding size 60 for single-task experiments and 120 for multi-task experiments. Training samples are augmented with random crops of RGB-D images and ± 45.0 yaw rotation perturbations (only in the real world as this degrades performance in simulation as we'll show in Section 4). We use a batch size 16 on a Nvidia 32GB V100 GPU for 200k steps (one day) for single-task experiments, and a batch size 48 on 8 Nvidia 32GB V100 GPUs for 600K steps (5 days) for language-conditioned multi-task experiments.

4 Experiments

We test Act3D in learning from demonstrations single-task and multi-task manipulation policies in simulation and the real world. In the multi-task setting, task and goal conditioning are given as input through language instructions. We conduct our simulated experiments in RLBench [20], an established simulation benchmark for learning manipulation policies, for the sake of reproducibility and benchmarking. Our experiments aim to answer the following questions: **1.** How does Act3D

compare against SOTA 2D multiview and 3D manipulation policies in single-task and multi-task settings? **2.** How does the test performance change with varying number of training demonstrations? **3.** How does Act3D generalize across camera viewpoints in comparison to existing 2D multiview policies? **4.** How do design choices such as relative 3D attention, pre-trained 2D backbones, weight-tied attention layers, and the number of coarse-to-fine sampling stages impact performance?

4.1 Evaluation in simulation

Datasets We test Act3D in RLbench in two settings to ensure a clear comparison with prior work: a single-task setting with 74 tasks proposed by HiveFormer [2] and a multi-task multi-variation setting with 18 tasks and 249 variations proposed by PerAct [1]; more details are in Appendix 6.3.

Baselines We compare Act3D with the following state-of-the-art manipulation policy learning methods: **1.** InstructRL [3], a 2D policy that directly predicts 6 DoF poses from image and language conditioning with a pre-trained vision-and-language backbone. **2.** PerAct [1], a 3D policy that voxelizes the workspace and detects the next best voxel action through global self-attention. **3.** HiveFormer [2] and Auto-λ [16], hybrid methods that detect a contact point within an image input, then regress an offset from this contact point. We report numbers from the papers when available.

Evaluation metric We evaluate policies by task completion success rate, the proportion of execution trajectories that lead to goal conditions specified in language instructions.

Single-task manipulation results We consider 74 tasks grouped into 9 categories, as proposed by HiveFormer [2]. Each method is trained with 100 demonstrations and evaluated on 500 unseen episodes. We show single-task quantitative results of our model and baselines in Figure 3. Act3D **reaches 83% success rate, an absolute improvement of 10% over InstructRL [3], prior SOTA in this setting**, and consistently outperforms it across all 9 categories of tasks. With only 10 demonstrations per task, Act3D is competitive with prior SOTA using 100 demonstrations per task.

Multi-task manipulation results We consider 18 tasks with 249 variations, as proposed by PerAct [1]. Each task includes 2-60 variations, which test generalization to test goal configurations that involve novel object colors, shapes, sizes, and categories. This is a more challenging setup than before, since the previous setting only tested generalization to novel arrangements of the same objects. Each method is trained with 100 demonstrations per task split across variations, and evaluated on 500 unseen episodes per task. We show multi-task quantitative results of our model and PerAct in Figure 3. Act3D reaches 65% success rate, an absolute improvement of 22% over PerAct, prior SOTA in this setting, consistently outperforming it across most tasks. **With only 10 demonstrations per task, Act3D outperforms PerAct using 100 demonstrations per task.** Note that Act3D also uses less than a third of PerAct's training computation budget: PerAct was trained for 16 days on 8 Nvidia V100 GPUs while we train for 5 days on the same hardware.

4.2 Evaluation in real-world

In our real-world setup, we conduct experiments with a Franka Emika Panda robot and a single Azure Kinect RGB-D sensor; more details are in Appendix 6.1. We designed 8 tasks (Figure 2) involving interactions with multiple types of objects, spanning liquid, articulated objects, and deformable objects. For each task, we collected 10 to 15 human demonstrations and trained a languaged-conditioned multi-task model on all data. We report the success rate on 10 episodes per task in Table 1. Act3D can capture semantic knowledge in demonstration well and performs reasonably well on all tasks, even with a single camera input. One major failure case comes from noisy depth sensing: when the depth image is not accurate, the selected point results in imprecise action prediction. Leveraging multi-

Task	# Train	Success
reach target	10	10/10
duck in oven	15	6/10
wipe coffee	15	7/10
fruits in bowl	10	8/10
stack cups	15	6/10
transfer beans	15	5/10
press handsan	10	10/10
uncrew cap	10	8/10

Table 1: Real-world tasks.

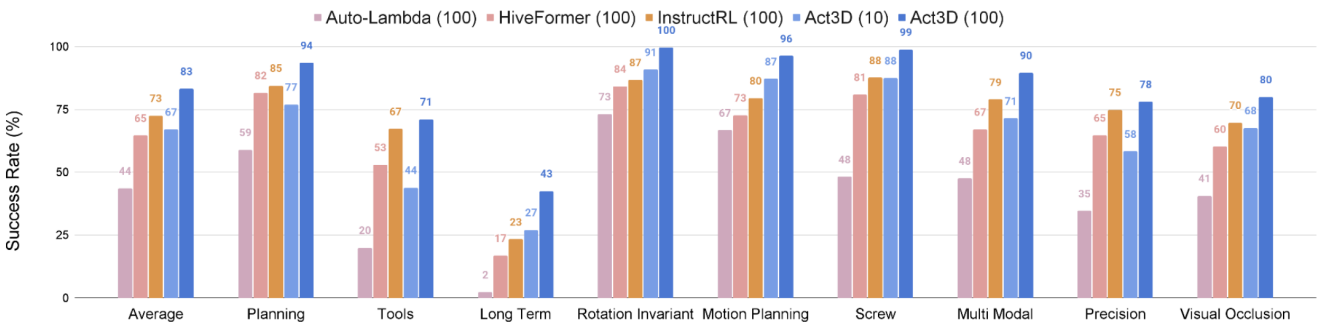
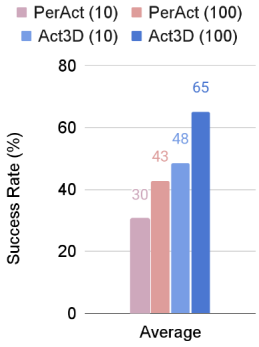


Figure 3: **Single-task performance.** On 74 RLBench tasks across 9 categories, Act3D reaches 83% success rate, an absolute improvement of 10% over InstructRL [3], prior SOTA in this setting.



	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
PerAct	68	80	32	72	72	56	**68**	84	**72**	80	16	68	32	60	36	68	**12**	**36**
Act3D	**92**	**93**	**66**	**93**	**82**	**92**	58	**94**	64	**94**	**82**	**90**	**90**	**92**	**52**	**92**	6	12

	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
PerAct	**28**	24	16	44	20	12	0	16	**16**	**20**	56	48	4	0	0	0	0	0
Act3D	26	**47**	**75**	**95**	**32**	**80**	**27**	**51**	7	8	**98**	**99**	**7**	**27**	**8**	**9**	**1**	**3**

Figure 4: **Multi-task performance.** On 18 RLBench tasks with 249 variations, Act3D reaches 65% success rate, an absolute improvement of 22% over PerAct [1], prior SOTA in this setting.

view input for error correction could improve this, and we leave this for future work. Please refer to our supplementary video for more qualitative videos of the robot executing the tasks.

4.3 Ablations

We ablate design choices of Act3D. We perform most ablations in the single-task setting on 5 tasks: pick cup, put knife on chopping board, put money in safe, slide block to target, take umbrella out of stand; ablate whether to predict if the motion planner needs to avoid collisions to reach of the pose on all 74 tasks; and the choice of pre-trained 2D backbone in the multi-task setting with all 18 tasks.

Generalization across camera viewpoints: We vary camera viewpoints at test time for both Act3D and HiveFormer [2]. The success rate drops to 20.4% for HiveFormer, a relative 77% drop, while Act3D achieves 74.2% success rate, a 24% relative drop. This shows detecting actions in 3D makes Act3D more robust to camera viewpoint changes than multiview 2D methods that regress offsets.

Weight-tying and coarse-to-fine sampling: All 3 stages of coarse-to-fine sampling are necessary: a model with only 2 stages of sampling and regressing an offset from the position selected at the second stage suffers a 4.5% performance drop. Tying weights across stages and relative 3D positional embeddings are both crucial; we observed severe overfitting without, reflected in respective 17.5% and 42.7% performance drops. Fine ghost point sampling stages should attend to local fine visual features with precise positions: all stages attending to global coarse features leads to a 8.3% performance drop. Act3D can effectively trade off inference computation for performance: sampling 10,000 ghost points, instead of the 1,000 the model was trained with, boosts performance by 4.9%.

Pre-training 2D features: We investigate the effect of the pre-trained 2D backbone in the multi-task setting where language instructions are most needed. A ResNet50 [32] backbone pre-trained

Table 2: **Ablations.**

	Model	Average success rate in single-task setting (5 tasks)
Core design choices	Best Act3D model (evaluated in Fig. 3)	**98.1**
	Only 2 stages of coarse-to-fine sampling: full workspace, 16 cm ball, regress an offset	93.6
	No weight tying across stages	80.6
	Absolute 3D positional embeddings	55.4
	Attention to only global coarse visual features	89.8
	Only 1000 ghost points at inference time	93.2
Viewpoint changes	Best Act3D model (evaluated in Fig. 3)	**74.2**
	HiveFormer	20.4
Augmentations	No image augmentations	**91.6**
	With rotation augmentations	86.2
Hyperparameter sensitivity	Double sampling ball diameters: 32 cm and 8 cm	96.6
	Halve sampling ball diameters: 8 cm and 2 cm	91.2
	500 ghost points at training time	95.8
	2000 ghost points at training time (need 2 GPUs)	**98.4**
		Multi-task setting (18 tasks)
Backbone	CLIP ResNet50 backbone	**65.1**
	ImageNet ResNet50 backbone	53.4
	No backbone (raw RGB)	45.2

with CLIP improves success rate by 8.7% over a ResNet50 backbone pre-trained on ImageNet, and by 16.9% over using raw RGB as the visual token features.

Augmentations: Random crops of RGB-D images boost success rate by 6.5%, but yaw rotation perturbations drop it by 11.9%. This is in line with PerAct [1] results in RLBench.

Hyperparameter sensitivity: Act3D is robust to hyperparameters. Doubling the diameter of ghost point sampling balls from (16 cm, 4 cm) to (32 cm, 8 cm) drops success rate by 1.5% and halving it to (8 cm, 2 cm) by 6.9%. Halving the total number of ghost points sampled from 1,000 to 500 drops success rate by 2.3% whereas doubling it to 2,000 increases success rate by 0.3%. We use 1,000 ghost points in our experiments to allow training with a single GPU per task.

4.4 Limitations and future work

Our framework currently has the following limitations: **1.** Act3D sometimes fails in very high-precision tasks, like screwing and insertions, requiring temporally fine-grain closed-loop control. **2.** Act3D doesn't handle manipulation of articulated object well, such as opening/closing doors, fridges, and ovens, which require a more precise trajectory than the one supplied by a motion planner that connects keyposes with collision-free straight lines. Learning-based trajectory prediction [39, 40] would help. **3.** Currently, for long horizon tasks our policy would need to predict all keyposes one by one. A hierarchical framework that would predict language subgoals for subtasks [41, 42, 43] and feed those to our action predictor would allow better re-usability of skills across tasks. All keypose prediction methods share the listed limitations. We leave these for future work.

5 Conclusion

We presented Act3D, a language-conditioned Transformer policy architecture to learn manipulation from demonstrations. From one or more posed RGB-D images and language instructions, it predicts 6-DoF robot end-effector keyposes by iteratively selecting and featurizing 3D point grids in the robot's workspace. Act3D sets a new state-of-the-art in RLBench, an established robot manipulation benchmark, and solves diverse manipulation tasks in the real world from a single RGB-D camera view and a handful of demonstrations. In thorough ablations, we showed the importance of relative 3D attentions, 2D feature pre-training, and weight tying during coarse-to-fine iterations.

References

[1] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR, 2023.

[2] P.-L. Guhur, S. Chen, R. G. Pinel, M. Tapaswi, I. Laptev, and C. Schmid. Instruction-driven history-aware policies for robotic manipulations. In *Conference on Robot Learning*, pages 175–187. PMLR, 2023.

[3] H. Liu, L. Lee, K. Lee, and P. Abbeel. Instruction-following agents with jointly pre-trained vision-language models. *arXiv preprint arXiv:2210.13431*, 2022.

[4] S. James, K. Wada, T. Laidlow, and A. J. Davison. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13739–13748, 2022.

[5] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[6] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pages 726–747. PMLR, 2021.

[7] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[9] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 213–229. Springer, 2020.

[10] B. Akgun, M. Cakmak, J. W. Yoo, and A. L. Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398, 2012.

[11] S. James and A. J. Davison. Q-attention: Enabling efficient learning for vision-based robotic manipulation. *IEEE Robotics and Automation Letters*, 7(2):1612–1619, 2022.

[12] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7:187–199, 2021.

[13] B. Graham. Sparse 3d convolutional neural networks, 2015.

[14] C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks, 2019.

[15] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver: General perception with iterative attention, 2021.

[16] S. Liu, S. James, A. J. Davison, and E. Johns. Auto-lambda: Disentangling dynamic task relationships. *arXiv preprint arXiv:2202.03091*, 2022.

[17] P. Parashar, J. Vakil, S. Powers, and C. Paxton. Spatial-language attention policies for efficient robot learning. *arXiv preprint arXiv:2304.11235*, 2023.

[18] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[19] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1290–1299, 2022.

[20] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[21] N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[22] 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.

[23] 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*, pages 991–1002. PMLR, 2022.

[24] D. Seita, P. Florence, J. Tompson, E. Coumans, V. Sindhwani, K. Goldberg, and A. Zeng. Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks, 2021.

[25] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022.

[26] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[27] H. Huang, D. Wang, R. Walters, and R. Platt. Equivariant transporter network. *arXiv preprint arXiv:2202.09400*, 2022.

[28] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation, 2022.

[29] S. Parisi, A. Rajeswaran, S. Purushwalkam, and A. Gupta. The unsurprising effectiveness of pre-trained vision models for control, 2022.

[30] L. Yen-Chen, A. Zeng, S. Song, P. Isola, and T.-Y. Lin. Learning to see before learning to act: Visual pre-training for manipulation, 2021.

[31] A. Stone, T. Xiao, Y. Lu, K. Gopalakrishnan, K.-H. Lee, Q. Vuong, P. Wohlhart, B. Zitkovich, F. Xia, C. Finn, and K. Hausman. Open-world object manipulation using pre-trained vision-language models, 2023.

[32] 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, 2021.

[33] P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations, 2018.

[34] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo. Swin transformer v2: Scaling up capacity and resolution, 2022.

[35] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding, 2022.

[36] X. Wu, Y. Lao, L. Jiang, X. Liu, and H. Zhao. Point transformer v2: Grouped vector attention and partition-based pooling, 2022.

[37] Y.-Q. Yang, Y.-X. Guo, J.-Y. Xiong, Y. Liu, H. Pan, P.-S. Wang, X. Tong, and B. Guo. Swin3d: A pretrained transformer backbone for 3d indoor scene understanding, 2023.

[38] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[39] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox. Motion policy networks. In *Conference on Robot Learning*, pages 967–977. PMLR, 2023.

[40] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[41] 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.

[42] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman, et al. Grounded decoding: Guiding text generation with grounded models for robot control. *arXiv preprint arXiv:2303.00855*, 2023.

[43] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

6 Appendix

6.1 Real-world Setup

Our real-robot setup contains a Franka Panda robotic arm equipped with a parallel jaw gripper, as shown in Figure 5. We get RGB-D input from a single Azure Kinect sensor at a front view at 30Hz. The image input is of resolution 1280×720, we crop and downsample it to 256×256. We calibrate the extrinsics of the camera with respect to the robot base using the `easy_handeye`[1] ROS package. We extract keyposes from demonstrations in the same was as in simulation. Our real-world multi-task policy is trained on 4 V100 GPUs for 3 days, and we run inference on a desktop with a single RTX4090 GPU. For robot control, we use the open-source `frankapy`[2] package to send real-time position-control commands to the robot.

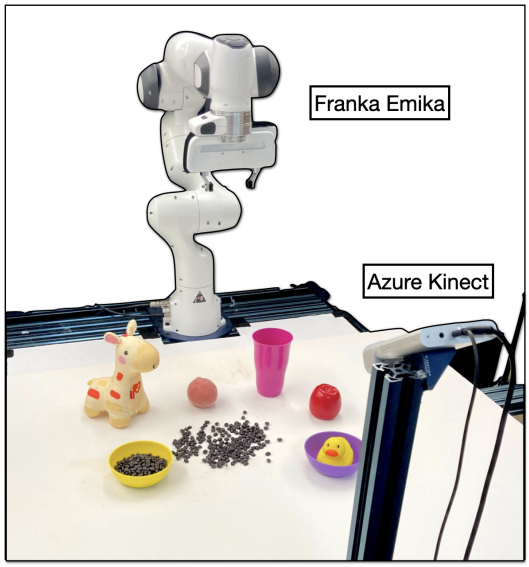


Figure 5: **Real-world setup.**

6.2 RLBench Simulation Setup

To ensure fair comparison with prior work, we use $n_{\text{cam}} \in \{3, 4\}$ cameras for simulated experiments depending on the evaluation setting. In our single-task evaluation setting first proposed by HiveFormer [2], we use the same 3 cameras they do $\{O_{\text{left}}, O_{\text{right}}, O_{\text{wrist}}\}$. In our multi-task evaluation setting first proposed by PerAct [1], we use the same 4 cameras they do $\{O_{\text{front}}, O_{\text{left}}, O_{\text{right}}, O_{\text{wrist}}\}$.

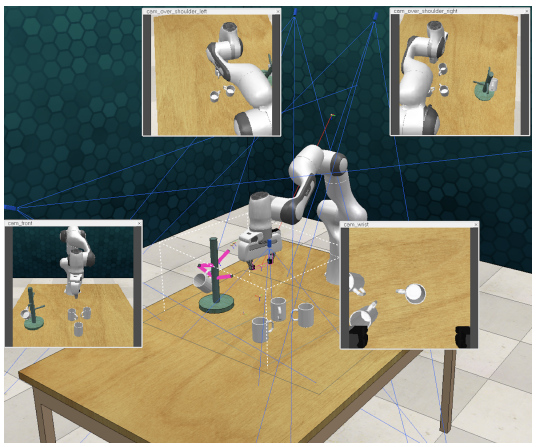


Figure 6: **RLbench simulation setup.**

[1] https://github.com/IFL-CAMP/easy_handeye
[2] https://github.com/iamlab-cmu/frankapy

6.3 RLBench Tasks

Task	Variation Type	# of Variations	Avg. Keyposes	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"

Figure 7: **PerAct [1] tasks.** We adopt the multi-task multi-variation setting from PerAct [1] with 18 tasks and 249 unique variations across object placement, color, size, category, count, and shape.

We adapt the single-task setting of HiveFormer [2] with 74 tasks grouped into 9 categories according to their key challenges. The 9 task groups are defined as follows:

- The **Planning** group contains tasks with multiple sub-goals (e.g. picking a basket ball and then throwing the ball). The included tasks are: basketball in hoop, put rubbish in bin, meat off grill, meat on grill, change channel, tv on, tower3, push buttons, stack wine.
- The **Tools** group is a special case of planning where a robot must grasp an object to interact with the target object. The included tasks are: slide block to target, reach and drag, take frame off hanger, water plants, hang frame on hanger, scoop with spatula, place hanger on rack, move hanger, sweep to dustpan, take plate off colored dish rack, screw nail.
- The **Long term** group requires more than 10 macro-steps to be completed. The included tasks are: wipe desk, stack blocks, take shoes out of box, slide cabinet open and place cups.
- The **Rotation-invariant** group can be solved without changes in the gripper rotation. The included tasks are: reach target, push button, lamp on, lamp off, push buttons, pick and lift, take lid off saucepan.
- The **Motion planner** group requires precise grasping. As observed in [81] such tasks often fail due to the motion planner. The included tasks are: toilet seat down, close laptop lid, open box, open drawer, close drawer, close box, phone on base, toilet seat up, put books on bookshelf.
- The **Multimodal** group can have multiple possible trajectories to solve a task due to a large affordance area of the target object (e.g. the edge of a cup). The included tasks are: pick up cup, turn tap, lift numbered block, beat the buzz, stack cups.
- The **Precision** group involves precise object manipulation. The included tasks are: take usb out of computer, play jenga, insert onto square peg, take umbrella out of umbrella stand, insert usb in computer, straighten rope, pick and lift small, put knife on chopping board, place shape in shape sorter, take toilet roll off stand, put umbrella in umbrella stand, setup checkers.
- The **Screw** group requires screwing an object. The included tasks are: turn oven on, change clock, open window, open wine bottle.

- The **Visual Occlusion** group involves tasks with large objects and thus there are occlusions from certain views. The included tasks are: close microwave, close fridge, close grill, open grill, unplug charger, press switch, take money out safe, open microwave, put money in safe, open door, close door, open fridge, open oven, plug charger in power supply