SLAP: Spatial-Language Attention Policies

Anonymous Author(s)
Affiliation
Address
email

Abstract: Despite great strides in language-guided manipulation, existing work has been constrained to table-top settings. Table-tops allow for perfect and consistent camera angles, properties are that do not hold in mobile manipulation. Task plans that involve moving around the environment must be robust to egocentric views and changes in the plane and angle of grasp. A further challenge is ensuring this is all true while still being able to learn skills efficiently from limited data. We propose Spatial-Language Attention Policies (SLAP) as a solution. SLAP uses three-dimensional tokens as the input representation to train a single multi-task, language-conditioned action prediction policy. Our method shows an 80% success rate in the real world across eight tasks with a single model, and a 47.5% success rate when unseen clutter and unseen object configurations are introduced, even with only a handful of examples per task. This represents an improvement of 30% over prior work (20% given unseen distractors and configurations). In addition, we show how SLAPs robustness enables allows us to execute Task Plans from open-vocabulary instructions using a large language model for multi-step mobile manipulation. For videos, see the website: `https://robotslap.github.io`

1 Introduction

Transformers have demonstrated impressive results on natural language processing tasks by being able to contextualize large numbers of tokens over long sequences, and even show substantial promise for robotics in a variety of manipulation tasks [1, 2, 3]. However, when it comes to using transformers for *mobile* robots performing long-horizon tasks, we face the challenge of representing spatial information in a useful way. In other words, we need a fundamental unit of representation - an equivalent of a "word" or "token" - that can handle spatial awareness in a way that is independent of the robot's exact embodiment. We argue this is essential for enabling robots to perform manipulation tasks in diverse human environments, where they need to be able to generalize to new positions, handle changes in the visual appearance of objects and be robust to irrelevant clutter. In this work, we propose Spatial-Language Attention Policies (SLAP), that use a point-cloud based tokenization which can scale to a number of viewpoints, and has a number of advantages over prior work.

SLAP tokenizes the world into a varying-length stream of multi-resolution spatial embeddings, which capture a local context based on PointNet++ [4] features. Unlike ViT-style [1], object-centric [5, 3], or static 3D grid features [2], our PointNet++-based [4] tokens capture free-form relations between observed points in space. This means that we can combine multiple camera views from a moving camera when making decisions and still process arbitrary-length sequences.

Our approach leverages a powerful skill representation we refer to as "attention-driven robot policies" [6, 7, 8, 2, 9] operating on an input-space combining language with spatial information. Unlike other methods that directly predict robot motor controls [10, 1], these techniques predict goal poses in Cartesian space and integrate them with a motion planner [6, 8, 2] or conditional low-level policy [9] to execute goal-driven motion. This approach requires less data, but it still has limitations such as making assumptions about the input scene's size and camera position and long training times [7, 6, 2]. However, these methods fall into a different trap: they make strong assumptions

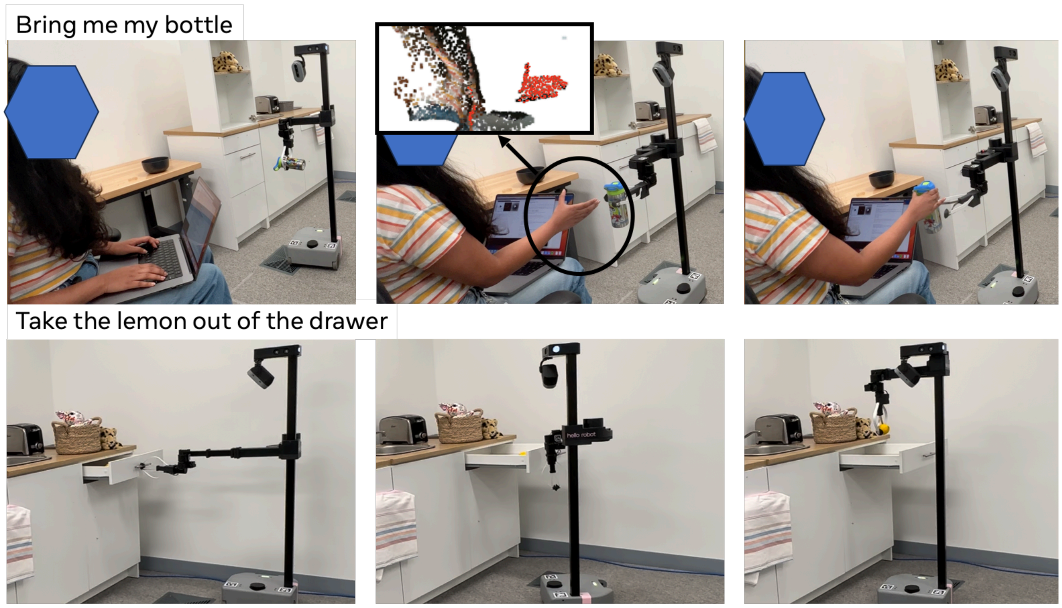

Figure 1: We propose SLAP, which allows us to learn skills for mobile manipulators to accomplish multi-step tasks given natural language goals. Our system works by training a language-conditioned *interaction prediction module*, which will determine which areas of a scene should be interacted with, in addition to an action policy which operates on predicted interaction points. This allows us to scale to more complex scenes, while predicting continuous actions.

about how big the input scene is [2], where the camera is [7, 6], and generally take a very long time to train [7, 6], meaning they could not be used too quickly teach policies in a new environment.

SLAP uses a hybrid policy architecture. The *interaction prediction* module determines which parts of the tokenized environment the robot focuses on, and a *relative action* module predicts parameters of continuous motion with respect to the interaction features in the world. SLAP generalizes better to unseen positions and orientations, as well as distractors, while being unrestricted by workspace size, and camera placement assumption, using fewer demonstrations and training in roughly a day.

We evaluate SLAP on two robot platforms. First, on a Franka Panda we can perform a direct skills comparison to the current state-of-the-art, PerAct, [2], where we demonstrate better performance with 80% success rate on 8 static real-world tasks on held-out scene configurations and a 47.5% success rate tested with out-of-distribution objects. Second, unlike prior work, we move beyond the stationary camera views and robot arms of a table-top setting, and demonstrate SLAP on the Hello Robot Stretch RE-2 mobile manipulator with an ego-centric camera and 6-DoF end-effector configuration. In this setting, we also include task planning to successfully execute natural language task instructions with 10 demonstrations over 5 learnt skills and 3 heuristic skills (Fig. 1).

2 Related Work

Attention-Based Policies. Attention-based policies have been widely studied in prior research and have been found to have superior data efficiency, generalization, and the ability to solve previously unsolvable problems [11, 9, 6, 12, 2, 13]. However, these approaches often rely on strong assumptions about the robot's workspace, such as modeling the entire workspace as a 2D image [12, 6, 7, 8] or a 3D voxel cube with predetermined scene bounds [2, 9]. This restricts their applicability and may lead to issues related to camera positioning, workspace location, and discretization size. Additionally, these works can be seen, at least partly, as applications of object detection systems like Detic [14] or 3DETR [15], but they lack the manipulation component.

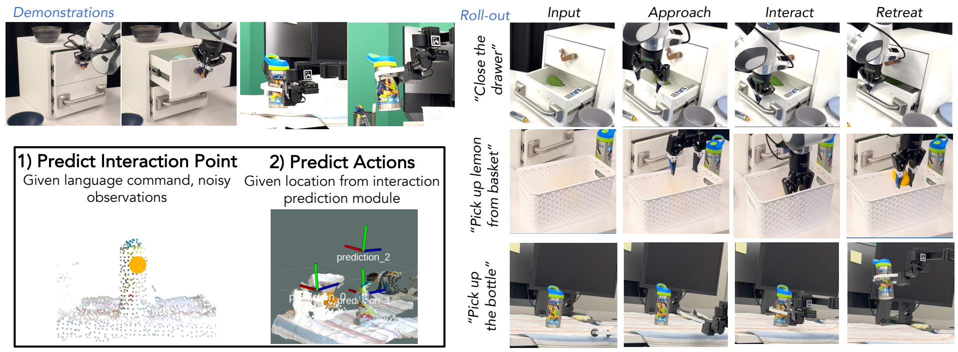

Figure 2: Spatial Language Attention Policies (SLAP) learn language-conditioned skills from few demonstrations in a wide variety of cluttered scenes. SLAP has two components: an "interaction prediction" module which localizes relevant features in a scene, and an "action prediction" module which uses local context to predict an executable action.

Compared to previous works, some recent studies focus on unstructured point clouds [11, 16]. These approaches demonstrate improved data efficiency and performance compared to traditional behavior cloning. For instance, Where2Act [11] and VAT-Mart [16] predict interaction trajectories, while UMPNet [17] supports closed-loop 6DoF trajectories. They share a common framework: a generalizable method to predict the interaction location and then predict local motion for the robot.

Training Quickly with Attention-Based Policies. CLIPort [7] and PerAct [2] are attention-based policies similar to Transporter Nets [6]. While fitting our definition of attention-based policies, they confine their workspace, use a rigid grid-like structure and treat action prediction as a discrete classification. While still a limited workspace, SPOT [12], demonstrated the usefulness of 2D attention-based policies for fast RL training, including sim-to-real transfer, and Zeng et al. [6] have shown these policies are valuable for certain real-world tabletop tasks like kitting.

Manipulation of Unknown Objects. Manipulation of unknown objects includes segmentation [18, 19], grasping [20, 21], placement [22], and multi-object rearrangement either from a goal image [23, 24] or from language instructions [25, 26]. These approaches rely, generally, on first segmenting relevant objects out, and then predicting how to grasp them and where to move them using separate purpose-built models, including for complex task and motion planning [27].

Language and Robotics. Language is a natural and powerful way to specify goals for multi-task robot systems. Several recent works [10, 1, 28] use a large-language model for task planning to combine sequential low-level skills and assume to learn the low-level skills with IL or RL. To realistically handle language task diversity, we need to learn these skills quickly. SLAP is more sample-efficient than prior IL or RL approaches. In PaLM-E [3], textual and multi-modal tokens are interleaved as inputs to the Transformer for handling language and multimodal data to generate high-level plans for robotics tasks. Our approach is a spatial extension of this strategy.

Language for Low-Level Skills. A number of works have shown how to learn low-level language-conditioned skills, e.g. [7, 2, 1, 29]. Like our work, Mees et al. [29] predicts 6DoF end effector goal positions end-to-end and sequences them with large language models. They predict a 2D affordance heatmap and depth from RGB; We do not predict depth, but specifically look at robustness and generalization, where theirs is trained from play data in mostly-fixed scenes. Shridhar et al. [2] predict a 3D voxelized grid and show strong real-word performance with relatively few examples, but don't look at out-of-domain generalization and are limited to a coarse voxelization of the world.

3 Approach

Most manipulation tasks necessarily involve interacting with environment objects [11]. We define an 'atomic skill' as a task that can be specified by an interaction point, and a sequence of relative offsets from this interaction point. For example, `pick('mug')` is an atomic skill as it can be defined in terms of an interaction point on the 'mug' and subsequent relative waypoints for approach, grasp, and lift actions. Similarly, `open('drawer')` is an atomic skill for which the interaction point is on the drawer handle, and relative waypoints from it can be defined for approach, grasp, and pull.

We train a two-phase language-conditioned policy $\pi(x, l)$, which takes visual observation x and a language command l as inputs and predicts an *interaction point* p_I, as well as a set of *relative motions*, which are offsets from this point, instead of absolute coordinates.However, any realistic task given to a home robot by a user typically involves more than one atomic skill. Our system breaks down a high-level natural language task description ($\mathcal{T}$) into a sequence of atomic skill descriptions $\{l_j\}$ and uses them to condition the atomic skill motion policies. Our full paradigm is as follows:

$$\begin{aligned}
&\mathcal{T} \rightarrow \{l_0, ..., l_n\} \rightarrow \{\pi_j(x_j, l_j)\}_n \\
&\qquad \forall j \in n, \quad \pi_j := (\pi_I, \pi_R), \text{ where:} \\
&\qquad\qquad \pi_I(x_j, l_j) \rightarrow p_I \qquad \text{(3D interaction point)} \\
&\qquad\qquad \pi_R(x_j, l_j, p_I) \rightarrow \{\mathbf{a}\}_m \quad \text{(sequence of actions)}
\end{aligned}$$

The interaction point p_I is predicted by an **Interaction Prediction Module** π_I, and the continuous component of the action by a **Relative Action Module** π_R. The Interaction Prediction Module π_I predicts *where the robot should attend to*; it is a specific location in the world, where the robot will be interacting with the object as a part of its skill, as shown in Fig. 2. π_R predicts a relative action sequence with respect to this contact point in the Cartesian space. These actions are then provided as input to a low-level controller to execute the trajectory. These models are trained using labeled expert demonstrations; a complete overview of the training process is shown in Fig. 4. Overall, the system outputs a sequence of end-effector actions $\mathbf{a}$.

3.1 Scene Representation

The input observation x is a structured point-cloud (PCD) in the robot's base-frame, constructed by combining the inputs from a sequence of pre-defined scanning actions. This point cloud is then preprocessed by voxelizing at a 1mm resolution to remove duplicate points from overlapping camera views. The pointcloud is then used as input into both π_I and π_R.

For π_I, we perform a second voxelization, this time at 5mm resolution. This creates the down-sampled set of points P, such that the interaction point $\hat{p_I} \in P$. This means π_I has a consistently high-dimensional input and action space - for a robot looking at its environment with a set of N aggregated observations, this can be 5000-8000 input "tokens" representing the scene.

While SLAP discretizes the world similar to prior work [30, 31, 2], we can do so selectively, at a higher resolution, and capture fine local details even in large scenes. We couple this with a set-based learning formulation which allows us to attend to fine details in a data-efficient manner.

3.2 Interaction Prediction Module

We use our insight about tasks being shaped around an interaction point to make learning more robust and more efficient: instead of predicting the agent's motion directly, we formulate our π_I to solely focus on predicting a specific point $p_I \in P$, representing a single $5mm$ voxel that is referred to as the "interaction point". This formulation is akin to learning object affordance and can be thought of as similar to prior work like Transporter Nets in 2D [6]. We hypothesize that predicting attention directly on visual features, even for manipulation actions, will make SLAP more general. We use a PerceiverIO [32] backbone to process the data, based on prior work on language-conditioned real-world policies [2].

We first pass our input point cloud through two *modified set abstraction* layers [4] which result in a sub-sampled point-cloud with each point's feature capturing the local spatial structure around it at two different resolutions. This encourages the classifier to pay attention to *local structures* rather than a specific point that may not be visible in real-world settings. We concatenate the CLIP [33] tokenized natural language command with the encoded point cloud to create an input sequence.

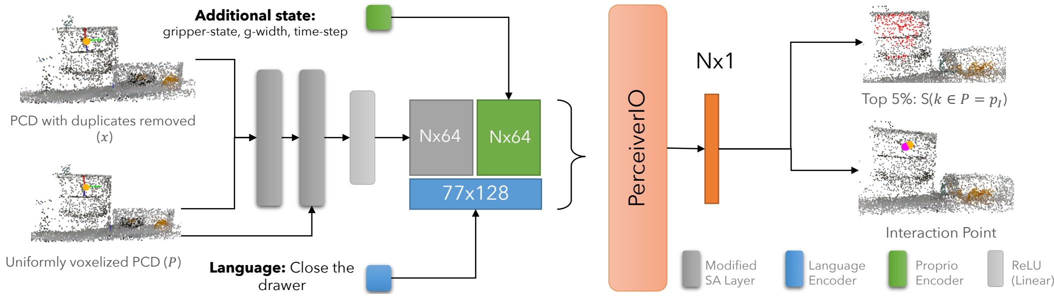

Figure 3: An overview of the architecture of the interaction prediction module. The point cloud is downsampled to remove duplicates and encoded using two modified set-abstraction layers. The SA layers generate a local spatial embedding which is concatenated with proprioceptive features - in our case, the current gripper state. Both spatial and language features are concatenated and input into a PerceiverIO transformer backbone. We then predict an interaction score per spatial feature and the *argmax* is chosen as the interaction site for command l.

Each point $i \in P$ in the point cloud is assigned a score with respect to task τ_j which results in the interaction point for that task, $p_I^j := \underset{x,y,z}{\text{argmax}}(S(i = p_I^j | l, x, P, \mathcal{D}^j))$, where $\mathcal{D}^j$ is the set of expert demonstrations provided for task τ_j. The IPM architecture overview is provided in Fig. 3. Note we also use semantic features from Detic in the Stretch experiment for training SLAP as an additional feature channel apart from the color-channels.

Modified Set Abstraction Layer. The default SA layer as introduced by Qi et al. [4] uses farthest point sampling (FPS) to determine which locations feature vectors are created. FPS ensures that subsampled point-cloud is a good representation of a given scene, without any guarantees about the granularity. However, it's very sensitive to the number of points selected - in most PointNet++-based policies, a fixed number of points are chosen using FPS [4]. However, SLAP must adapt to scenes of varying sizes, possibly with multiple views, and still not miss small details.

We propose an alternative PointNet++ set abstraction layer, which computes embeddings based on the original and an *evenly* downsampled version of the point-cloud, P. This results in a denser spatial embedding by considering a subset of all points within a certain radius of each-point in the downsampled point-cloud. This downsampled set of points guarantees we can attend to even small features, and allows us to predict an interaction point p_I from the PointNet++ aggregated features.

3.3 Relative Action Module

The relative action module relies on the interaction point predicted by the classifier and operates on a cropped point cloud, x_R, around this point to predict the actions associated with this sequence. As in the interaction prediction module, the model uses a cascade of modified *set abstraction* layers as the backbone to compute a multi-resolution encoding feature over the cropped point cloud. We train three multi-head regressors (described further below) over these features to predict the actions for the overall task. Specifically, π_R has three heads, one for each component of the relative action space: gripper activation g, position offset δp, and orientation q. Our LSTM-based architecture (details in B.1) can predict skills with variable number of actions (3,4 in our experiments).

Positions and orientations associated with the interaction action generally tend to be much closer to the crop-center thus we train one model per action to encourage each action to be learned according to its own distribution. Also note that the cropped input point-cloud is not perfectly centered at the ground truth interaction point $\hat{p_I}$, but rather with some noise added: $\hat{p_I}' = \hat{p_I} + \mathcal{N}(0, \sigma)$. This is done to force the action predictor to be robust to sub-optimal interaction point predictions by the interaction predictor module during real-world roll-outs. Thus, for each part of the action sequence, the keyframe position is calculated as: $p = p_I + \delta p$. When acting, the robot will move to (p, q) via a motion planner, and then will send a command to the gripper to set its state to g.

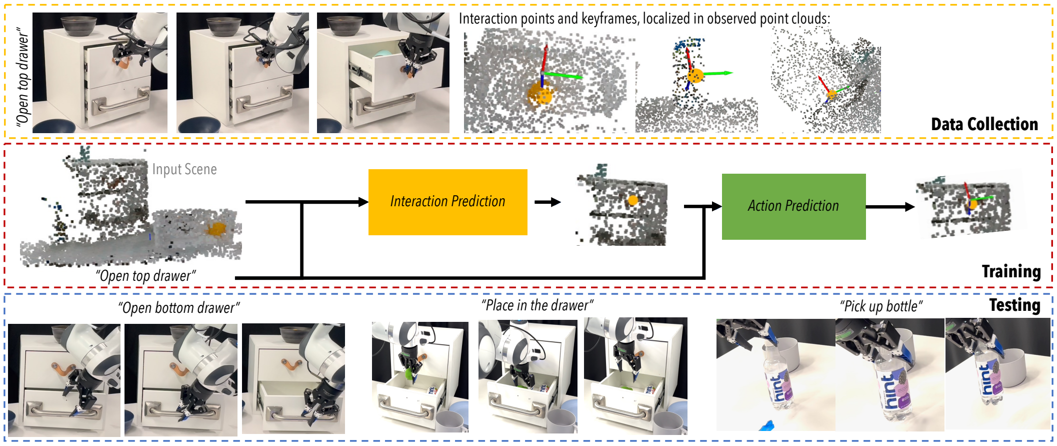

Figure 4: Illustration of the complete process for training SLAP. Demonstrations are collected and used to train the Interaction Prediction module and the Action Prediction Module separately.

3.4 Training SLAP

To collect data, an expert operator guides the robot through a trajectory, pressing a button to record *keyframes* representing crucial parts of a task. At each keyframe, we record the associated expert action $\hat{a} = (\hat{\delta p}, \hat{q}, \hat{g})$. We assume that low-level controllers exist - in our case, we use Polymetis [34] for the Franka arm and Hello Robot's controllers[1] for Stretch. Example tasks are shown in Fig. 6.

Interaction Prediction Module. We train π_I with a cross-entropy loss, predicting the interaction point p_I from the downsampled set of coarse voxels P. We additionally apply what we call a *locality loss* (L_{loc}), as per prior work [35]. Conceptually, we want to penalize points the further they are from the contact point, both to encourage learning relevant features as well as to aid in ignoring distractors. To achieve this, we define the locality loss as: $L_{loc} = \sum_{k \in P} \mathrm{softmax}(f_k) \|\hat{p}_I, k\|^2$, where f_k is the output of the transformer for point $k \in P$. The $softmax$ turns f_k into attention over the points, meaning that L_{loc} can be interpreted as a weighted average of the square distances. Points further from $\hat{p}_I$ are therefore encouraged to have lower classification scores. Combining our two losses, we obtain $L_I = CE(P, \hat{p}_I) + \frac{w}{|P|} L_{loc}$, where w is a scaling constant that implicitly defines how much spread to allow in our points.

Relative Action Module. To train π_R, we use the weighted sum of three different losses. We train $a = (p, q, g) = \pi_R(x_R)$ with an L2 loss over the δp, a quaternion distance metric for the loss on q based on prior work [36] and binary cross-entropy loss for gripper action classification (Sec. A.3).

3.5 Task Planning

Consider a natural language instruction from a user such as 'put away the bottle on a table'. We decompose it to a sequence of atomic skills as: `goto('bottle')`, `pick_up('bottle')`, `goto('table')`, and `place_on('table')`. We programmatically create natural language and code templates for 16 task families and generated a dataset of 500k samples. We use LLaMA [37] models for in-context learning [38, 39] and adapter fine-tuning [40] to learn the mapping between natural language task instructions to the corresponding sequence of atomic skills.

4 Experiments

We report the success rate of our model for 8 real-world manipulation tasks in Table 2, and compare it against prior baselines trained using the same labeling scheme. Overall, we see an improvement of $1.6x$ over our best comparative baseline, PerAct [2]. We test each model under two different conditions: *Seen setting assumption;* i.e. those with seen distractor objects and objects placed roughly in

[1] https://github.com/hello-robot/stretch_ros

Skill Name	Seen PerAct	Seen SLAP	Unseen PerAct	Unseen SLAP
Open bottom drawer	00%	**80%**	00%	**60%**
Open top drawer	60%	**80%**	**40%**	**40%**
Close drawer	**100%**	**100%**	**40%**	**40%**
Pick lemon from basket	60%	**80%**	10%	**40%**
Pick bottle	**60%**	**60%**	**60%**	40%
Place into the drawer	60%	**80%**	40%	**60%**
Place into the basket	40%	**100%**	10%	**60%**
Place into the bowl	40%	**60%**	00%	**40%**
Average Success Rate	50%	80%	27.5%	47.5%
Improvement		1.6x		1.7x

Table 2: SLAP and PerAct [2] performance on real world Franka manipulation tasks. We evaluate both seen scenes (seen object positions and distractors), but in different arrangements, and unseen scenes with previously-unseen object positions and distractors. SLAP is notably better overall in both conditions.

Skill Name	SLAP (5 tries)
Open Drawer	60%
Close Drawer	100%
Take bottle	80%
Pour into bowl	80%

Table 3: SLAP on a mobile manipulator using a multi-task model across 4 skills. With semantic predictions added to our feature space, we see the model perform better on unseen scenes with new distractors and unseen relative position of the robot with respect to the scene

the same range of positions and orientations as in the training data in any relative arrangement (inlcuding unseen). Second, we test under *unseen setting assumptions;* i.e. those with unseen distractor objects and the implicated object placed significantly out of the range of positions and orientations already seen. We run 5 tests per scene setting per skill per model and report the percentage success numbers in Table 2. We compare our model against Perceiver-Actor (PerAct) [2]. We train each model for the same number of training steps and choose the SLAP model based on the best validation loss. For PerAct, we use the last checkpoint, per their testing practices [2].

We also run a per-skill evaluation of SLAP on Stretch under the *unseen setting assumption* (see Fig. 3) accomplished by adding unseen distractor objects to the scene and moving the robot base position within reachable distance of the object. Note demonstrations were taken on a different robot than the one policies were deployed on.

4.1 Longitudinal Task Execution on Stretch

We trained a multi-task model for the Stretch robot for five skills using 10 demonstrations each. This model was deployed in an end-to-end system which operates over code-list generated by a task-planner (as in Sec. 3.5). We ran 5 prompts end-to-end with 4 to 8 skills each, using ground truth plans - we verify the viability of generating these task plans in §4.2. These experiments are done under the *unseen setting* with the robot spawning anywhere with respect to the objects. For fair evaluation in low-data regime, we add some structure by specifying orthogonal viewing direction for objects. Once the agent finds the object of interest it fires an initial prediction using SLAP to find most promising interaction point. This prediction happens under any dynamic viewing angle of the object (we assume the robot can see the object). This dynamic prediction and pre-programmed viewing angle is used to *approach* the object at an orthogonal viewing angle where the model fires an actionable prediction for the full skill execution. We observe adding semantic features from Detic significantly improves IPM performance with unseen distractors (80% against 47.5% in Table 2) however we see failures when relative position is significantly perturbed.

	Success
Total	68.5
Heuristic	66.0
Learned	80.0

Table 1: End-to-end performance. Learned skills outperform heuristics except due to Detic failures.

4.2 Task planning with in-context learning and fine-tuning LLaMa

Previous work has shown the strength of language models as zero-shot planners [41], a result strengthened by improved techniques for "in-context learning" or prompting [42]. To verify that models can produce task plans with the skills we defined, we experiment with both in-context learning (IC) [43] of LLaMA [37] and adapter fine-tuning (FT) [40].

Figure 6: Examples of tasks executed on a Franka arm through our trained model in a clean setting. We trained numerous tasks (left) and tested on both seen and unseen objects (right).

Table 5 presents the models' verb, noun, and combined accuracies. Task Correctness is a binary score if the entire prediction was correct, and finally, latency is measured in seconds on a single A6000 with 16 GB RAM.[2] High Task Correctness from a small model verifies the compatibility of our skills with LLM task planning.

LLaMa		Verb	Noun	Acc.	Task Corr.	Lat. (sec.)
IC	7B	83	81	76	61	16.4
IC	30B	81	81	76	62	27.6
FT	7B	100	98	99	91	19.5

Figure 5: Fine-tuning (FT) outperforms in-context learning (IC) for same latency.

4.3 Ablations

Hybrid vs Monolithic Architecture (Table-top). For the same number of epochs, SLAP does better than PerAct on 6 of 8 tasks when tested in in-distribution setting and 5 of 8 tasks when tested in out-of-distribution settings. PerAct performs equally well as our model for 2 of 8 tasks on our in-distribution scenes. Similarly, for our "hard" generalization scenes, PerAct performed equally well in two cases, and actually outperformed SLAP when picking up a bottle. In failure cases, π_R predicted the correct trajectory, but not with respect to the right part of the object.

Unseen Scene Generalization. We see a drop in the success rate for both PerAct and SLAP when tested on out-of-distribution settings. PerAct would often predict the correct approach actions, but then it would fail to grasp accurately. With SLAP, however, we saw that p_I was predicted fairly accurately, but the regressor would fail for out-of-distribution object placements specifically because of bad orientation prediction. When π_I failed, it was because the position and orientation of the target object was dramatically different, *and* unseen distractors confused it. We see better results for SLAP under Stretch setting due to the addition of semantic features from Detic.

5 Conclusion

We proposed a method for learning visual-language policies for decision making in complex environments. SLAP is a novel architecture which combines the *structure* of a point-cloud based input with *semantics* from language and accompanying demonstrations to predict continuous end-effector actions for manipulation tasks. We demonstrate SLAP on two hardware platforms, including an end-to-end evaluation on a mobile manipulator, something not present in prior work.

5.1 Limitations

SLAP has high variance in out-of-distribution situations, resulting in complete failure if π_I fails to correctly identify the context. For π_R, multimodal or noisy data still poses issues; replacing π_R with a policy which can better handle this data, e.g. Diffusion Policies [45]. Overall system has multiple points-of-failure due to heuristic policies, unaligned language and vision models; end-to-end trainable architectures and cross-modal alignment could help.

[2]Adaptor fine-tuning increases the model size by $\sim$6%, which accounts for the additional latency compared to IC. We use standard inference libraries so results are comparable, but not optimized for runtime [44].

References

[1] 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.

[2] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. *arXiv preprint arXiv:2209.05451*, 2022.

[3] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[5] W. Yuan, C. Paxton, K. Desingh, and D. Fox. Sornet: Spatial object-centric representations for sequential manipulation. In *Conference on Robot Learning*, pages 148–157. PMLR, 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. PMLRG, 2021.

[7] 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.

[8] C. Huang, O. Mees, A. Zeng, and W. Burgard. Visual language maps for robot navigation. *arXiv preprint arXiv:2210.05714*, 2022.

[9] 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.

[10] 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.

[11] K. Mo, L. J. Guibas, M. Mukadam, A. Gupta, and S. Tulsiani. Where2act: From pixels to actions for articulated 3d objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6813–6823, 2021.

[12] A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager. ""Good robot!": Efficient reinforcement learning for multi-step visual tasks with sim to real transfer. *IEEE Robotics and Automation Letters*, 5(4):6724–6731, 2020.

[13] P.-L. Guhur, S. Chen, R. Garcia, M. Tapaswi, I. Laptev, and C. Schmid. Instruction-driven history-aware policies for robotic manipulations. *arXiv preprint arXiv:2209.04899*, 2022.

[14] X. Zhou, R. Girdhar, A. Joulin, P. Krähenbühl, and I. Misra. Detecting twenty-thousand classes using image-level supervision. In *ECCV*, 2022.

[15] I. Misra, R. Girdhar, and A. Joulin. An end-to-end transformer model for 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2906–2917, 2021.

[16] R. Wu, Y. Zhao, K. Mo, Z. Guo, Y. Wang, T. Wu, Q. Fan, X. Chen, L. Guibas, and H. Dong. Vat-mart: Learning visual action trajectory proposals for manipulating 3d articulated objects. *arXiv preprint arXiv:2106.14440*, 2021.

[17] Z. Xu, Z. He, and S. Song. Umpnet: Universal manipulation policy network for articulated objects. *arXiv preprint arXiv:2109.05668*, 2021.

[18] C. Xie, Y. Xiang, A. Mousavian, and D. Fox. Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics*, 37(5):1343–1359, 2021.

[19] Y. Xiang, C. Xie, A. Mousavian, and D. Fox. Learning rgb-d feature embeddings for unseen object instance segmentation. In *Conference on Robot Learning*, pages 461–470. PMLR, 2021.

[20] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox. 6-dof grasping for target-driven object manipulation in clutter. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6232–6238. IEEE, 2020.

[21] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444. IEEE, 2021.

[22] C. Paxton, C. Xie, T. Hermans, and D. Fox. Predicting stable configurations for semantic placement of novel objects. In *Conference on Robot Learning*, pages 806–815. PMLR, 2022.

[23] A. H. Qureshi, A. Mousavian, C. Paxton, M. C. Yip, and D. Fox. Nerp: Neural rearrangement planning for unknown objects. *arXiv preprint arXiv:2106.01352*, 2021.

[24] A. Goyal, A. Mousavian, C. Paxton, Y.-W. Chao, B. Okorn, J. Deng, and D. Fox. Ifor: Iterative flow minimization for robotic object rearrangement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14787–14797, 2022.

[25] W. Liu, C. Paxton, T. Hermans, and D. Fox. Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6322–6329. IEEE, 2022.

[26] W. Liu, T. Hermans, S. Chernova, and C. Paxton. Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects. *arXiv preprint arXiv:2211.04604*, 2022.

[27] A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett. Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1940–1946. IEEE, 2022.

[28] 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.

[29] O. Mees, J. Borja-Diaz, and W. Burgard. Grounding language with visual affordances over unstructured data. *arXiv preprint arXiv:2210.01911*, 2022.

[30] V. Blukis, C. Paxton, D. Fox, A. Garg, and Y. Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *Conference on Robot Learning*, pages 706–717. PMLR, 2022.

[31] S. Y. Min, D. S. Chaplot, P. Ravikumar, Y. Bisk, and R. Salakhutdinov. Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*, 2021.

[32] 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.

[33] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[34] Y. Lin, A. S. Wang, G. Sutanto, A. Rai, and F. Meier. Polymetis. https://facebookresearch.github.io/fairo/polymetis/, 2021.

[35] S. Powers, A. Gupta, and C. Paxton. Evaluating continual learning on a home robot, 2023.

[36] C. Paxton, Y. Bisk, J. Thomason, A. Byravan, and D. Foxl. Prospection: Interpretable plans from language by predicting the future. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6942–6948. IEEE, 2019.

[37] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[38] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *ArXiv*, 2020.

[39] E. Akyürek, D. Schuurmans, J. Andreas, T. Ma, and D. Zhou. What learning algorithm is in-context learning? investigations with linear models, 2023.

[40] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021.

[41] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[42] O. Ram, Y. Levine, I. Dalmedigos, D. Muhlgay, A. Shashua, K. Leyton-Brown, and Y. Shoham. In-Context Retrieval-Augmented Language Models, Jan. 2023. URL http://arxiv.org/abs/2302.00083. arXiv:2302.00083 [cs].

[43] J. Wei, J. Wei, Y. Tay, D. Tran, A. Webson, Y. Lu, X. Chen, H. Liu, D. Huang, D. Zhou, and T. Ma. Larger language models do in-context learning differently, Mar. 2023. URL http://arxiv.org/abs/2303.03846. arXiv:2303.03846 [cs].

[44] J. Fernandez, J. Kahn, C. Na, Y. Bisk, and E. Strubell. The Framework Tax: Disparities Between Inference Efficiency in Research and Deployment. *ArXiv*, 2023. URL https://arxiv.org/abs/2302.06117.

[45] 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.

[46] C. C. Kemp, A. Edsinger, H. M. Clever, and B. Matulevich. The design of stretch: A compact, lightweight mobile manipulator for indoor human environments. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3150–3157. IEEE, 2022.

[47] 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.

[48] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. *arXiv preprint arXiv:2210.13702*, 2022.

[49] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[50] N. M. M. Shafiullah, C. Paxton, L. Pinto, S. Chintala, and A. Szlam. Clip-fields: Weakly supervised semantic fields for robotic memory. *arXiv preprint arXiv:2210.05663*, 2022.

[51] B. Bolte, A. Wang, J. Yang, M. Mukadam, M. Kalakrishnan, and C. Paxton. Usa-net: Unified semantic and affordance representations for robot memory. *arXiv preprint arXiv:2304.12164*, 2023.

Appendix

Table of Contents

A Training

Below is expanded information on our training, algorithm, and data processing to improve reproducibility.

A.1 Data Collection and Annotation

When collecting an episode with the Franka arm, we first scan the scene with a pre-defined list of scanning positions to collect an aggregated x. In our case, we make no assumption as to what or how many these are, or how large the resulting input point cloud x is. With the Hello Robot Stretch [46], we collect data based on exactly where the robot is looking.

Then, we collect demonstration data using kinesthetic teaching for the Franka arm (demonstrator physically moves the robot) and via controller teleoperation for the Stretch robot. The demonstrator moves the arm through the trajectory associated with each task, explicitly recording the *keyframes* [47] associated with action execution. These represent the salient moments within a task – the bottlenecks in the tasks' state space, which can be connected by our low-level controller.

A.2 Data Processing

We execute each individual skill open-loop based on an initial observation. We use data augmentation to make sure even with relatively few examples, we still see good generalization performance.

Data Augmentation. Prior work in RGB-D perception for robotic manipulation (e.g. [18, 48]) has extensively used a variety of data augmentation tricks to improve real-world performance. In this work, we use three different data augmentation techniques to randomize the input scene x used to train $p_I = \pi_I(x, l)$:

- *Elliptical dropout:* Random ellipses are dropped out from the depth channel to emulate occlusions and random noise, as per prior work [49, 18]. Number of ellipses are sampled from a Poisson distribution with mean of 10.
- *Multiplicative Noise:* Again as per prior work [49, 18, 22], we add multiplicative noise from a gamma process to the depth channel.
- *Additive Noise:* Gaussian process noise is added to the points in the point-cloud. Parameters for the Gaussian distribution are sampled uniformly from given ranges. This is to emulate the natural frame-to-frame point-cloud noise that occurs in the real-world.
- *Rotational Randomization:* Similar to prior work [2, 22, 25], we rotate our entire scene around the z-axis within a range of ± 45 degrees to help force the model to learn rotational invariance.
- *Random cropping:* with $p = 0.75$, we randomly crop to a radius around $\hat{p}_I + \delta$, where δ is a random translation sampled from a Gaussian distribution. The radius to crop is randomly sampled in $(1, 2)$ meters.

Data Augmentation for π_R. We crop the relational input $x_R \subset x$ around the ground-truth p_I, using a fixed radius $r = 0.1m$. We implement an additional augmentation for learning our action model. Since p_I is chosen from the discretized set of downsampled points P, we might in principle be limited to this granularity of response. Instead, we randomly shift both p_I and the positional action δp by some uniformly-sampled offset $\delta r \in \mathbb{R}^3$, with up to $0.025m$ of noise. This lets π_R adapt to interaction prediction errors of up to several centimeters.

A.3 Action Prediction Losses

Following [36] for the orientation, we can compute the angle between two quaternions θ as:

$$\theta = \cos^{-1}(2\langle \hat{q}_1, \hat{q}_2 \rangle^2). \tag{1}$$

We can remove the cosine component and use it as a squared distance metric between 0 and 1. We then compute the position and orientation loss as:

$$L_R = \lambda_p \|\delta p - \hat{\delta p}\|_2^2 + \lambda_q (1 - \langle \hat{q}, q \rangle) \tag{2}$$

where λ_p and λ_q are weights on the positional and orientation components of the loss, set to 1 and $1e-2$ respectively.

Predicting gripper action is a classification problem trained with a cross-entropy loss. For input we use the task's language description embedding and proprioceptive information about the robot as input, i.e. $s = (l, g_{act}, g_w, ts)$ where g_{act} is 1 if gripper is closed and 0 otherwise, g_w is the distance between fingers of the gripper and ts is the time-step. The gripper action loss is then:

$$L_g = \lambda_g CE(g, \hat{g}) \tag{3}$$

where λ_g is the weight on cross-entropy loss set to 0.0001. The batch-size is set at 1 for this implementation.

We train π_I and π_R separately for $n = 85$ epochs. At each epoch, we compare validation performance to the current best - if validation did not improve, we reset performance to the last best model.

A.4 Skill Weighting

In Stretch experiments, we used a wide range of skills with different error tolerances and corresponding variances. As a result,

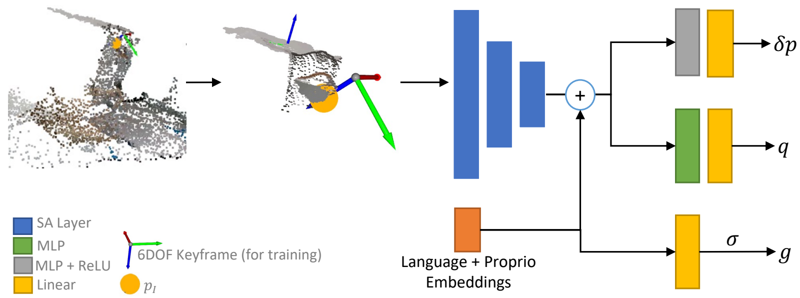

Figure 7: Regression model architecture with separate heads for each output. The point-cloud is cropped around the interaction point with some perturbation and passed to a cascade of set abstraction layers. Encoded spatial features are then concatenated with language and proprioception embeddings to predict position offset of action from interaction point, absolute orientation and gripper action as a boolean.

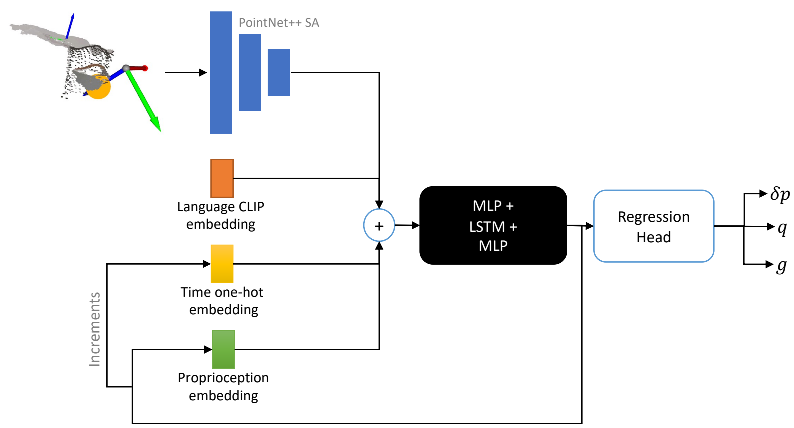

Figure 8: LSTM-based regression model architecture based on the regression head and PointNet++ embeddings introduced in Fig. 7. LSTM-based architecture shows higher stability in learning action distribution with wider distribution due to the conditioning effect.

B Relative Action Module

In our work, the Relative Action Module π_R is assumed to be some *local* policy which predicts end-effector poses. In our case, we implement two different versions of this policy, one which was used on the static Franka manipulator and one which was implemented on the Stretch. In both cases:

- The policy predicts an *end effector pose* relative to the predicted interaction point from π_I
- The policy is conditioned on a *local crop* around this interaction point.

B.1 Relative Action Model: MLP Implementation

Fig. 7 gives an overview of the MLP version of the regression model. The model takes in the cropped point cloud (augmented during training as discussed in Sec. A.2. We saw that injecting random noise to the interation point during training allowed the policy to, at test time, recover from failures (because it predicted an interaction point *near* the correct area, instead of at the correct position). However, we observed this architecture suffered when positional distribution of actions varied widely with respect to the interaction point position across tasks. Thus we implemented a LSTM version based on the MLP version which exhibited better performance in learning wider action distribution, based on our initial experiments.

Figure 9: Within distribution objects used at training time and out-of-distribution objects introduced during testing in our experiments.

C Tasks

We generate a dataset with more than 500k samples using natural language descriptions and corresponding atomic skill templates. We consider the following task families:
'bring x from y articulated'
'bring x from y articulated to pour in z'
'bring x from y articulated to wipe z'
'bring x from y surface'
'bring x from y surface to pour in z'
'bring x from y surface to pour in z then place on w surface'
'bring x from y surface to wipe z'
'move x from y articulated to z articulated'
'move x from y articulated to z surface'
'move x from y surface to z articulated'
'move x from y surface to z surface'
'take x from human to pour in z and place on y surface'
'take x from human to pour in z'
'take x from human to wipe z'
'take x from human to z articulated'
'take x from human to z surface'
For each task family, we define a corresponding template containing the sequence of atomic skills. To populate these templates and generate the data, we create a list of more than 150 movable objects kitchen objects, surfaces like `table`, `kitchen counter` and articulated objects like `drawer`, `cabinets`. For `pour` skill, we create a list of "spillable" items such as `cup of coffee`, or `bowl of jelly beans`. Similarly, for wipe skill, we have a list of items to wipe with such as `sponge`, or `brush`.

D Skills

Here we refer to atomic skills learned by SLAP as simple tasks or "tasks". This allows us to discuss corresponding "actions" that are defined in terms of the relative offset from the interaction point.

D.1 In vs. Out Of Distribution

We used a number of objects for our Franka manipulation experiments, which included both in- and out-of-distribution objects. One goal of SLAP is to show that our methods generalize much better than others to different types of scenes and different levels of clutter.

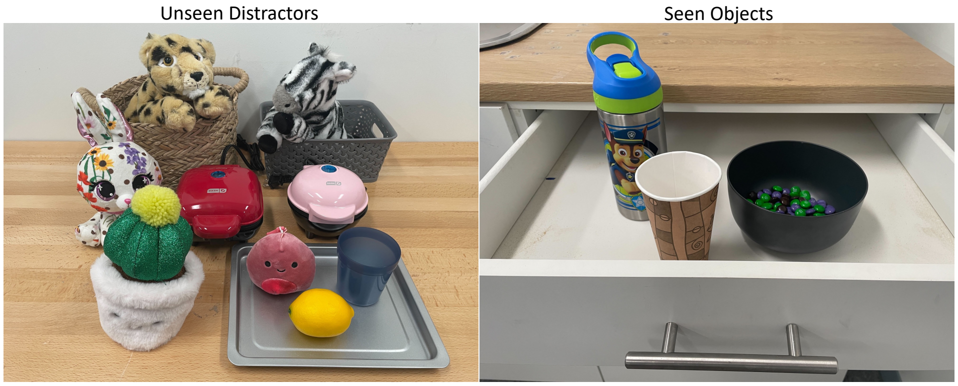

Figure 10: Seen objects and unseen distractors used in longitudinal experiments with Stretch.

Every real-world task scene had a sub-sample of all within-distribution objects.

1. Open the top drawer
    - *Task:* Grab the small loop and pull the drawer open. Drawer configuration within training data is face-first with slight orientation changes
    - *Action labeling:* Approach the loop, grab the loop, pull the drawer out
    - *Success metric:* When the drawer is open by 50% or more
2. Open the bottom drawer
    - *Task:* Grab the cylindrical handle and pull the drawer open. Drawer configuration within training data is face-first with slight orientation changes. Note significantly different grasp is required than for top drawer
    - *Action labeling:* Approach the handle, grab it, pull the drawer out
    - *Success metric:* When the drawer is open by 50% or more
3. Close the drawer
    - *Task:* This task is unqualified, i.e. the instructor does not say whether to close the top or bottom drawer instead the agent must determine which drawer needs closing from its state and close it. Align the gripper with the front of whichever drawer is open and push it closed. The training set always has only one of the drawers open, in a front-facing configuration with small orientation changes
    - *Action labeling:* Approach drawer from the front, make contact, push until closed
    - *Success metric:* When the drawer is closed to within 10% of its limit or when arm is maximally stretched out to its limit (when the drawer is kept far back)
4. Place inside the drawer
    - *Task:* Approach an empty spot inside the drawer and place whatever is in hand inside it
    - *Action labeling:* Top-down approach pose on top of the drawer, move to make contact with the surface and release the object, move up for retreat
    - *Success metric:* Object should be inside the drawer
5. Pick lemon from the basket
    - *Task:* Reach into the basket where lemon is placed and pick up the lemon
    - *Action labeling:*
    - *Success metric:* Lemon should be in robot's gripper
    - *Considerations:* Since the roll-out is open-loop and a lemon is spherical in nature, a trial was assigned success if the lemon rolled out of hand upon contact after the 2nd action. This was done consistently for both PerAct and SLAP.

6. Place in the bowl
    - *Task:* Place whatever is in robot's hand into the bowl receptacle
    - *Action labeling:* Approach action on top of the bowl, interaction action inside the bowl with gripper open, retreat action on top of the bowl
    - *Success metric:* The object in hand should be inside the bowl now
7. Place in the basket
    - *Task:* Place the object in robot's hand into the basket
    - *Action labeling:* Approach action on top of the free space in basket, interaction action inside the basket with gripper open, retreat action on top of the basket
    - *Success metric:* The object is inside the basket
8. Pick up the bottle
    - *Task:* Pick up the bottle from the table
    - *Action labeling:* Approach pose in front of the robot with open gripper, grasp pose with gripper enclosing the bottle and gripper closed, retreat action at some height from previous action with grippers closed
    - *Success metric:* The bottle should be in robot's gripper off the table

Notably, success for opening drawers is if the drawer is 50% open after execution; this is because sometimes the drawer is too close to the robot's base for it to open fully with a fixed-base Franka arm.

D.2 Language Annotations

Below we include the list of language annotations used in our experiments. Table 4 shows the language that was used to train the model; we're able to show some robustness to different language expressions. We performed a set of experiments on held-out, out-of-distribution language despite this not being the focus of our work; this test language is shown in Table 5.

D.3 Out of distribution Results from SLAP

We show more results for attention point predicted by π_I in Fig. 11. For the placing task, the agent has never seen a heavily cluttered drawer inside before, but it is able to find flat space which indicates placing affordance. For the bottle picking task, this sample has a lemon right next to the bottle which changes the shape of the point-cloud around the bottle. We see that π_I is able to find an interaction point albeit with placement different from expert and lower down on the bottle. Similarly the open top drawer sample has more heavy clutter on and around the drawer to test robustness.

Fig. 12 shows the prediction and generated trajectory for picking up a previously unseen bottle Note that while the models are able to detect the out of distribution bottle, the trajectory actually fails due to bottle being much wider and requiring more accuracy in grasping.

D.4 Motion Planning Failures

Our evaluation system has a simple motion planner which is not collision aware as a result we saw a number of task failures for both the models. However, we note that the frequency of task failures due to motion planning problems was higher for PerAct. We think it is because PerAct predicts each action of the same task as an entirely separate prediction trial, while SLAP forces continuity on the relative motions for the same task by centering them around the interaction point (see Fig. 12). That said, we also note with an advanced motion planner PerAct will not run into such issues as seen during our evaluations. Authors note in their own paper their heavy reliance on good motion planning solutions [2].

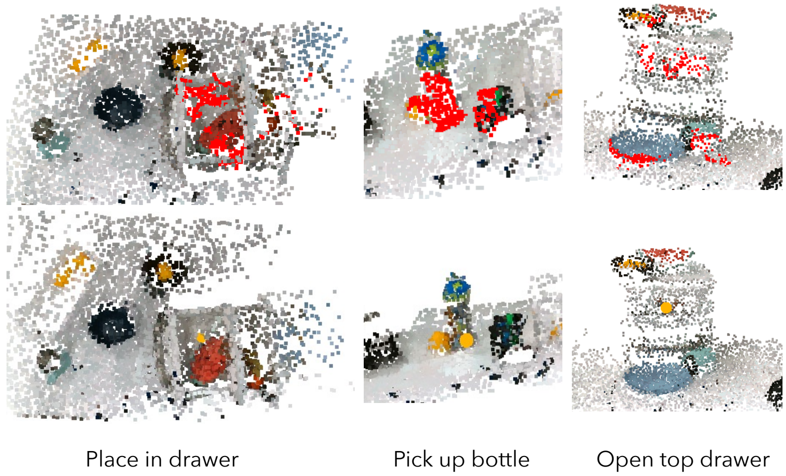

Figure 11: Examples of out of distribution predictions made by π_I. We show that it is able to handle heavy clutter around the implicated object to predict interaction points. Note that the prediction for bottle picking is sub-optimal in this example.

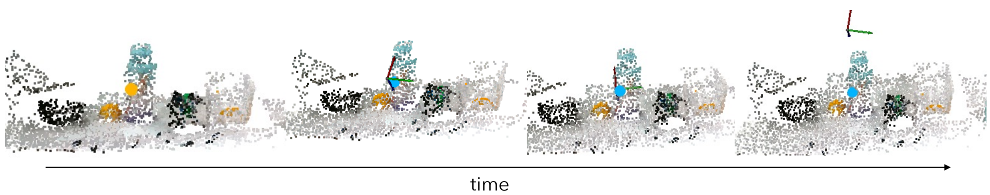

Figure 12: A generalization example of success for our model. The new bottle has same shape as the within distribution bottle but is much taller, different in color and wider in girth. The model is able to predict the interaction site and a feasible trajectory around it. We note though the execution of this trajectory was a failure; due to wider girth of the bottle the predicted grasp was not accurate enough to enclose the object.

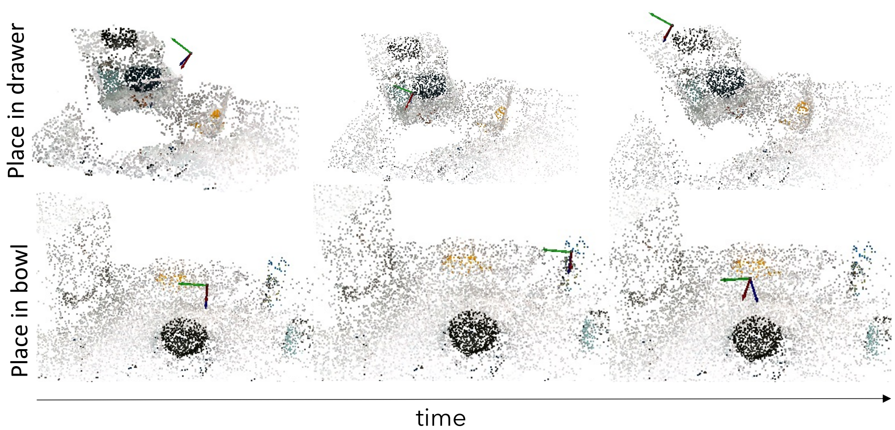

Figure 13: Examples of failure cases for our baseline, PerAct, for the "place in drawer" and "place in bowl" tasks. In the top example, the gripper is moved from drawer's side towards inside, instead of from the top as demonstrated by expert. The gripper ends up pushing off the drawer to the side as our motion-planner is not collision-aware. Note that SLAP does not exhibit such behaviors as π_R implicitly learns the collision constraints present in demonstrated data. In the bottom example, each action prediction is disjointed from previous and semantically wrong.

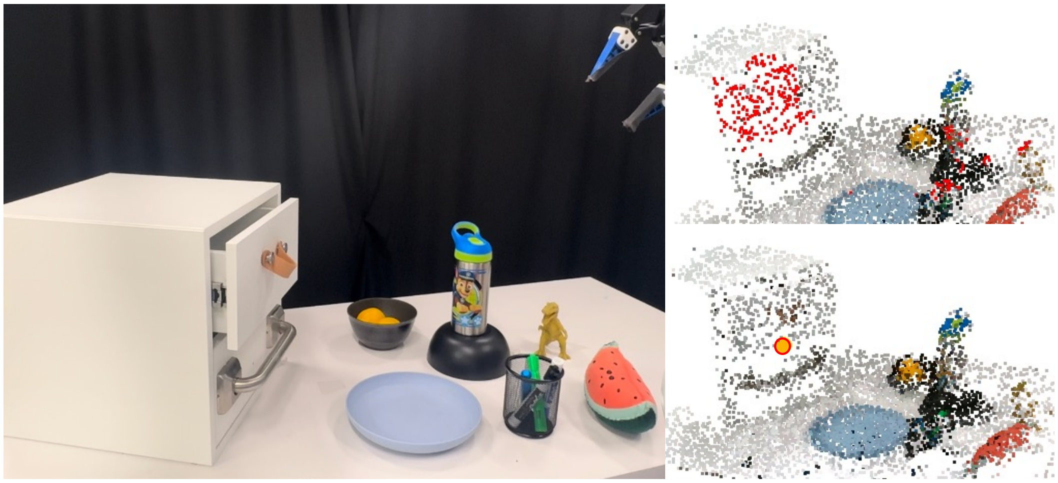

Figure 14: An out-of-distribution SLAP failure example where an extreme sideways configuration of the drawer is paired with unseen distractors for the "open top drawer" skill. We can see the attention mask ranking other distractors in its top 5% and failing to choose an optimal interaction point.

E Additional Analysis

E.1 Visualizing the Learned Attention

Since we use scores to choose the final interaction point, our classifier model is naturally interpretable, being able to highlight points of interest in a scene. We visualize this attention by selecting the points with the highest 5% of interaction score given a language command l.

E.2 Language Generalization

By using pretrained CLIP language embeddings to learn our spatial attention module π_I, our model can generalize to unseen language to some extent. We tested this by running an experiment where we evaluate performance on in-distribution scene settings, prompted by a held-out list of language expressions. We choose three representative tasks for this experiment and run 10 tests with 2 different language phrasings.

F Additional Related Work

We note some other related work that's relevant to SLAP, but not as directly relevant.

Vision-Language Navigation. Similar representations are often used to predict subgoals for exploration in vision-language navigation [30, 31, 8, 50, 51]. HLSM builds a voxel map [30], whereas FiLM builds a 2D representation and learns to predict where to go next [31]. VLMaps proposes an object-centric solution, creating a set of candidate objects to move to [8], while CLIP-Fields learns an implicit representation which can be used to make predictions about point attentions in responds to language queries [50], but does not look at manipulation. Similarly, USA-Net [51] generates a 3d representation with a lot of semantic features.

Task Name	Training Annotations
pick up the bottle	pick up a bottle from the table
	pick up a bottle
	grab my water bottle
pick up a lemon	pick the lemon from inside the white basket
	grab a lemon from the basket on the table
	hand me a lemon from that white basket
place lemon in bowl	place the lemon from your gripper into the bowl
	add the lemon to a bowl on the table
	put the lemon in the bowl
place in the basket	place the object in your hand into the basket
	put the object into the white basket
	place the thing into the basket on the table
open bottom drawer	open the bottom drawer of the shelf on the table
	pull the second drawer out
	open the lowest drawer
close the drawer	close the drawers
	push in the drawer
	close the drawer with your gripper
open top drawer	open the top drawer of the shelf on the table
	pull the first drawer out
	open the highest drawer
place in the drawer	put it into the drawer
	place the object into the open drawer
	add the object to the drawer

Table 4: Examples of language used to train the model.

Task Name	Held-Out Test Annotations
Pick up the bottle	Grab the bottle from the table
	Pick up the water bottle
Open the top drawer	Pull top drawer out
	Open the first drawer
Place into the drawer	Add to the drawer
	Put inside the drawer

Table 5: Examples of out-of-distribution language annotations used for evaluation