Transformers are adaptable task planners

Anonymous Author(s)
Affiliation
Address
email

Code: https://anonymous.4open.science/r/temporal_task_planner-Paper148/

A Hardware Experiments

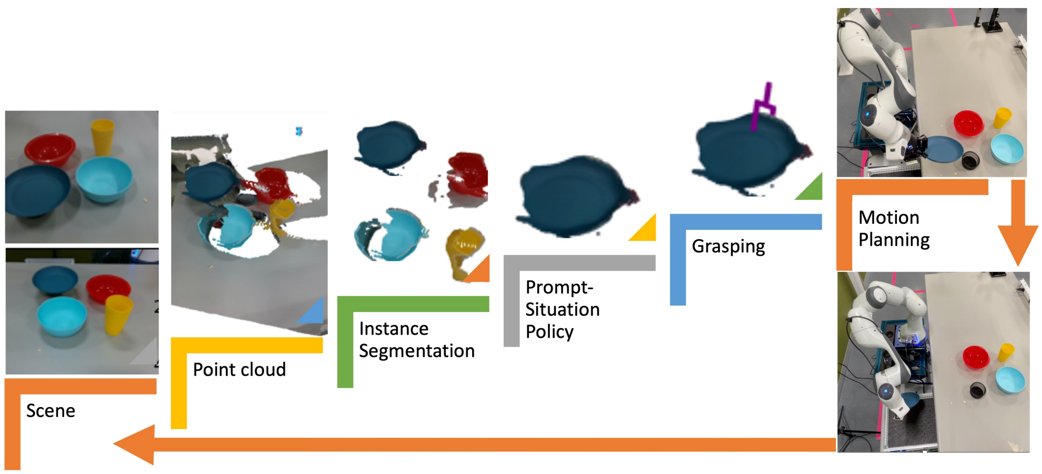


Figure 1: Pipeline for Real Hardware Experiments

A.1 Real-world prompt demonstration

Here we describe how we collected and processed a visual, human demonstration in the real-world to treat as a prompt for the trained TTP policy (Fig. 2). Essentially, we collect demonstration pointcloud sequences and manually segment them into different pick-place segments, followed by extracting object states. At each high-level step, we measure the state using three RealSense RGBD cameras[1], which are calibrated to the robot frame of reference using ARTags [2]. The camera output, extrinsics, and intrinsics are combined using Open3D [3] to generate a combined pointcloud. This pointcloud is segmented and clustered to give objects' pose and category using the algorithm from [4] and DBScan. For each object point cloud cluster, we identify the object pose based on the mean of the point cloud. For category information we use median RGB value of the pointcloud, and map it to apriori known set of objects. In the future this can be replaced by more advanced techniques like MaskRCNN [5]. Placement poses are approximated as a fixed, known location, as the place action on hardware is a fixed 'drop' position and orientation. The per step state of the objects is used to create the input prompt tokens used to condition the policy rollout in the real-world, as described in Section 3.2.

A.2 Hardware policy rollout

We zero-shot transfer our policy π trained in simulation to robotic hardware, by assuming low-level controllers. We use a Franka Panda equipped with a Robotiq 2F-85 gripper, controlled using

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

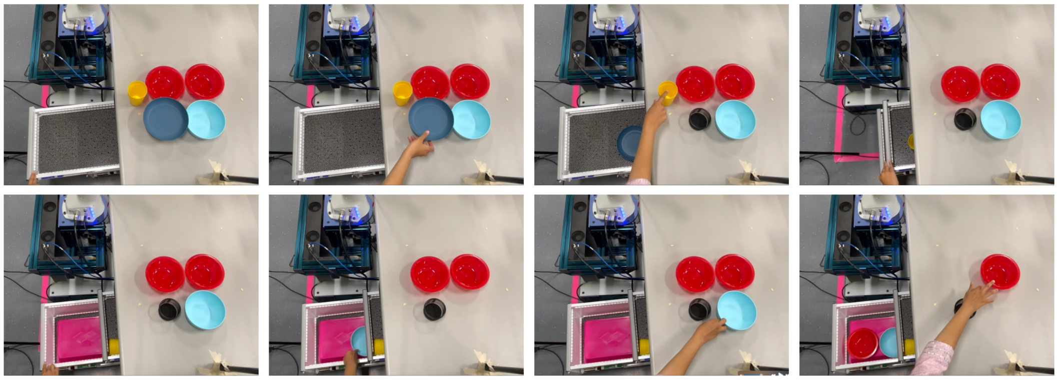


Figure 2: Human demonstration of real-world rearrangement of household dishes.

the Polymetis control framework [6]. Our hardware setup mirrors our simulation, with different categories of dishware (bowls, cups, plates) on a table, a "dishwasher" (cabinet with two drawers). The objective is to select an object to pick and place it into a drawer (rack) (see Fig. 2).

Once we collect the human prompt demonstration tokens, we can use them to condition the learned policy π from simulation. Converting the hardware state to tokens input to π follows the same pipeline as the ones used for collecting human demonstrations. At each step, the scene is captured using 3 Realsense cameras, and the combined pointcound is segmented and clustered to get object poses and categories. This information along with the timestep is used to generate instance tokens as described in Section 2 for all objects visible to the cameras. For visible already placed objects, the place pose is approximated as a fixed location. The policy π, conditioned on the human demo, reasons about the state of the environment, and chooses which object to pick. Next, we use a grasp generator from [7] that operates on point clouds to generate candidate grasp locations on the chosen object. We filter out grasp locations that are kinematically not reachable by the robot, as well as grasp locations located on points that intersect with other objects in the scene. Next, we select the top 5 most confident grasps, as estimated by the grasp generator, and choose the most top-down grasp. We design an pre-grasp approach pose for the robot which is the same final orientation as the grasp, located higher on the grasping plane. The robot moves to the approach pose following a minimum-jerk trajectory, and then follows a straight line path along the approach axes to grasp the object. Once grasped, the object is moved to the pre-defined place pose and dropped in a drawer. The primitives for opening and closing the drawers are manually designed on hardware.

The learned policy, conditioned on prompt demonstrations, is applied to two variations of the same scene, and the predicted pick actions are executed. Fig.3 shows the captured image from one of the three cameras, the merged point cloud and the chosen object to pick and selected grasp for the same. The policy was successful once with 100% success rate, and once with 75%, shown in Fig.**??**. The failure case was caused due to a perception error – a bowl was classified as a plate. This demonstrates that our approach (TTP) can be trained in simulation and applied directly to hardware. The policy is robust to minor hardware errors like a failed grasp; it just measures the new state of the environment and chooses the next object to grasp. For example, if the robot fails to grasp a bowl, and slightly shifts the bowl, the cameras measure the new pose of the bowl, which is sent to the policy. However, TTP relies on accurate perception of the state. If an object is incorrectly classified, the policy might choose to pick the wrong object, deviating from the demonstration preference. In the future, we would like to further evaluate our approach on more diverse real-world settings and measure its sensitivity to the different hardware components, informing future choices for learning robust policies.

A.3 Transforming hardware to simulation data distribution

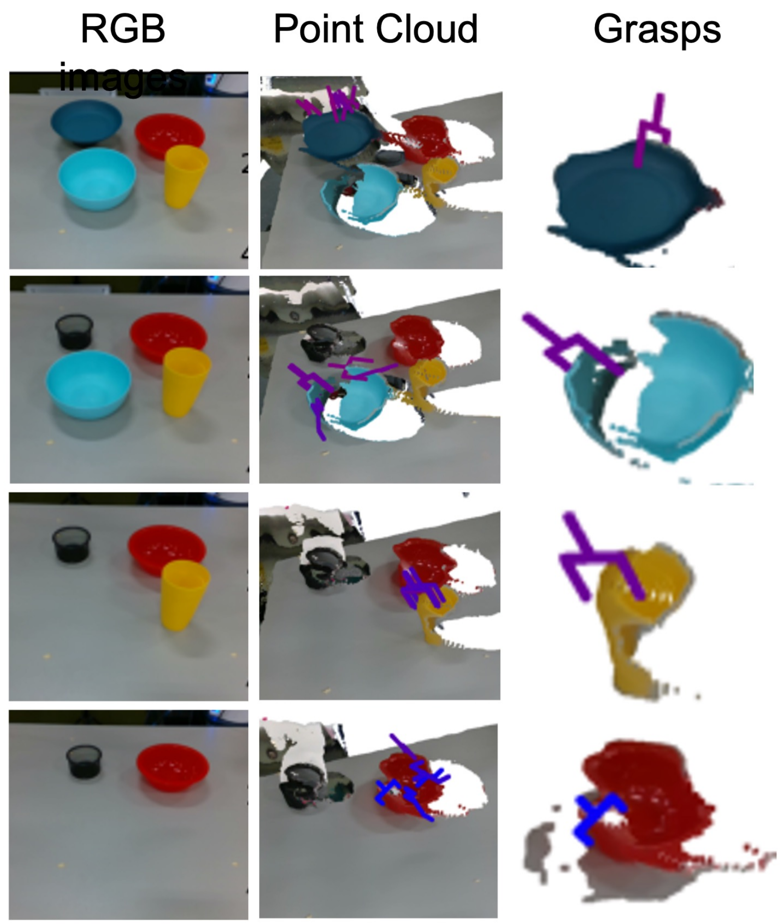


Figure 3: Point cloud and grasps for different objects during policy rollout.

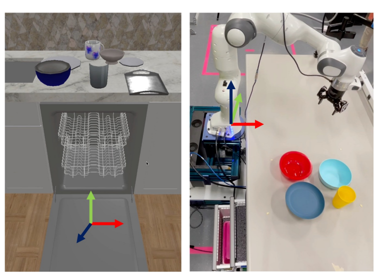


Figure 4: Coordinate Frame of reference in simulation (left) and real world setting (right). Red is x-axis, green is y-axis and blue is z-axis.

The policy trained in simulation applies zero-shot to real-world scenarios, but it requires a coordinate transform. Fig. 4 shows the coordinate frame of reference in simulation and real world setting. Since our instance embedding uses the poses of objects, it is dependant on the coordinate frame that the training data was collected in. Since hardware and simulation are significantly different, this coordinate frame is not the same between sim and real. We build a transformation that converts hardware measured poses to the simulation frame of reference, which is then used to create the instance tokens. This ensures that there is no sim-to-real gap in object positions, reducing the challenges involved in applying such a simulation trained policy to hardware. In this section we describe how we convert the real world coordinates to simulation frame coordinates for running the trained TTP policy on a Franka arm.

We use the semantic work area in simulation and hardware to transform the hardware position coordinates to simulation position coordinates. We measure the extremes of the real workspace by manually moving the robot to record positions and orientations that define the extents of the workspace for the table. The extents of the drawers are measured by placing ARTag markers. We build 3 real-to-sim transformations using the extents for counter, top rack and bottom rack: Let $X \in \mathbb{R}^{3\times N}$

contain homogeneous $xz-$ coordinates of a work area, along its column, as follows:

$$X = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots \\ z^{(1)} & z^{(2)} & \cdots \\ 1 & 1 & \cdots \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots \end{bmatrix} \tag{1}$$

As the required transformation from real to simulation involves scaling and translation only, we have 4 unknowns, namely, $\boldsymbol{a} = [\alpha_x, \alpha_y, x_{trans}, z_{trans}]$. Here α_x, α_z are scaling factors and x_{trans}, z_{trans} are translation offset for x and z axis respectively. To solve $X_{sim} = AX_{hw}$, we need to find the transformation matrix $A = \hat{\boldsymbol{a}} = \begin{bmatrix} \alpha_x & 0 & x_{trans} \\ 0 & \alpha_z & z_{trans} \\ 0 & 0 & 1 \end{bmatrix}$.

$$X_{sim} = \hat{\boldsymbol{a}} X_{hw} \tag{2}$$

Rewriting the system of linear equations, (3)

$$\implies \begin{bmatrix} x_{sim}^{(1)} \\ z_{sim}^{(1)} \\ x_{sim}^{(2)} \\ z_{sim}^{(2)} \\ \vdots \end{bmatrix} = \begin{bmatrix} x_{hw}^{(1)} & 0 & 1 & 0 \\ 0 & z_{hw}^{(1)} & 0 & 1 \\ x_{hw}^{(2)} & 0 & 1 & 0 \\ 0 & z_{hw}^{(2)} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \boldsymbol{a}^T \tag{4}$$

(5)

Let the above equation be expressed as $Y_{sim} = Z_{hw} a^T$ where $Y_{sim} \in \mathbb{R}^{2N\times 1}$, $Z_{hw} \in \mathbb{R}^{2N\times 4}$, and $a^T \in \mathbb{R}^{4\times 1}$. Assuming we have sufficient number of pairs of corresponding points in simulation and real world, we can solve for $\boldsymbol{a}$ by least squares $a = (Z_{hw}^T Z_{hw})^{-1} Z_{hw}^T Y_{sim}$. The height y_{sim} is chosen from a look-up table based on y_{hw}. Once we compute the transformation A, we store it for later to process arbitrary coordinates from real to sim, as shown below.

```python
def get_simulation_coordinates(xyz_hw: List[float], A: np.array) -> List:
    xz_hw = [xyz_hw[0], xyz_hw[2]]
    X_hw = get_homogenous_coordinates(xz_hw)
    X_sim_homo = np.matmul(A, X_hw)
    y_sim = process_height(xyz_hw[1])
    X_sim = [X_sim_homo[0]/X_sim_homo[2], y_sim, X_sim_homo[1]/X_sim_homo[2]]
    return X_sim
```

The objects used in simulation training are different from hardware objects, even though they belong to the same categories. For example, while both sim and real have a small plate, the sizes of these plates are different. We can estimate the size of the objects based on actual bounding box from the segmentation pipeline. However, it is significantly out-of-distribution from the training data, due to object mismatch. So, we map each detected object to the nearest matching object in simulation and use the simulation size as the input to the policy. This is non-ideal, as the placing might differ for sim versus real objects. In the future, we would like to train with rich variations of object bounding box size in simulation so that the policy can generalize to unseen object shapes in the real world.

B Simulation Setup

B.1 Dataset

"Replica Synthetic Apartment 0 Kitchen" consists of a fully-interactive dishwasher with a door and two sliding racks, an adjacent counter with a sink, and a "stage" with walls, floors, and ceiling. We use selected objects from the ReplicaCAD [8] dataset, including seven types of dishes (cups, glasses, trays, small bowls, big bowls, small plates, big plates) which are loaded into the dishwasher. Fig. 5 shows a human demonstration recorded in simulation by pointing and clicking on desired

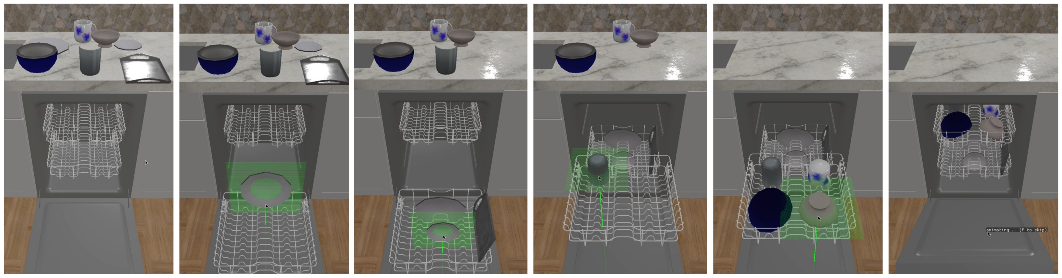


Figure 5: Human demonstration with point and click in simulation

object to pick and place. We initialize every scene with an empty dishwasher and random objects placed on the counter. Next, we generate dishwasher loading demonstrations, adhering to a given preference, using an expert-designed data generation script. Expert actions include, opening/closing dishwasher/racks, picking/placing objects in feasible locations or the sink if there are no feasible locations left. Experts differ in their preferences, and might choose different object arrangements in the dishwasher.

B.2 Expert Preferences

We define a preference in terms of expert demonstration 'properties', like which rack is loaded first with what objects? There are combinatorially many preferences possible, depending on how many objects we use in the training set. For example, Table 1 describes the preferences of dishwasher loading in terms of three properties - first loaded tray, objects in top and bottom tray. Each preference specifies properties such as which rack to load first and their contents. In Table 1, Preferences 1 & 2 vary in the order of which rack is loaded first, while 2 & 3 both load the bottom rack first with similar categories on top and bottom but with different orderings for these categories. Other preferences can have different combinations of objects loaded per rack.

Table 1: Three example preferences for dishwasher loading. Rack order and their respective contents (ordered by preference).

First?	*Top*	*Bottom*
Top	1. cups 2. glasses 3. small bowl	1. big plates 2. small plates 3. trays 4. big bowls
Bottom	1. cups 2. glasses 3. small bowl	1. big plates 2. small plates 3. trays 4. big bowl
Bottom	1. small plate 2. glasses 3. cups	1. big bowls 2. trays 3. big plates 4. small bowl

To describe a preference, let there be k properties, where each can take m_k values respectively. For example, a property to describe preference can be which rack is loaded first, and this can take two values; either top or bottom rack. The total number of possible preferences is $G = \prod_{i=1}^{k} m_i$.

In our demonstration dataset, we have 100 unique sessions per preference. Each session can act as a prompt to indicate preference as well as provide situation for the policy. Each session is about $\sim$ 30 steps long. With 7 preferences, this leads to $70,000 \times 30 = 2,100,000 \sim 2$ million total training samples, creating a relatively large training dataset from only 100 unique demonstrations per preference.

Individual task preferences differ in the sequence of expert actions, but collectively, preferences share the underlying task semantics. For example, the user always opens the dishwasher rack before loading it for all preferences. By jointly learning over all preferences, our policy can benefit from cross-preference data to learn task structure, and sparse per-preference data to learn expert preference.

B.3 Dynamically appearing objects

To add additional complexity to our simulation environment, we simulate a setting with dynamically appearing objects later in the episode. During each session, the scene is initialized with $p\%$ of maximum objects allowed. The policy/expert starts filling a dishwasher using these initialized objects. After all the initial objects are loaded and both racks are closed, new objects are initialized one-per-timestep to the policy. The goal is to simulate an environment where the policy does not have perfect knowledge of the scene, and needs to reactively reason about new information. The policy reasons on both object configurations in the racks, and the new object type to decide whether to 'open a rack and place the utensil' or 'drop the object in the sink'.

C Training

In this Section we describe details of the different components of our learning pipeline.

C.1 Baseline: GNN

Architecture We use GNN with attention. The input consists of 12 dimensional attribute inputs (1D-timestep, 3D-category bounding box extents, 7D-pose, 1D-is object or not?) and 12 dimensional one-hot encoding for the preference.

```
input_dim: 24
hidden_dim: 128
epochs: 200
batch_size: 32
```

Optimizer : Adam with $lr = 0.01$ and weight_decay= $1e-3$.

Reward function for GNN-RL Reward function for the RL policy is defined in terms of preference. The policy gets a reward of +1 every time it predicts the instance to pick that has the category according to the preference order and whether it is placed on the preferred rack.

C.2 Our proposed approach: TTP

Architecture We use a 2-layer 2-head Transformer network for encoder and decoder. The input dimension of instance embedding is 256 and the hidden layer dimension is 512. The attributes contribute to the instance embedding as follows:

```
C_embed: 16
category_embed_size: 64
pose_embed_size:  128
temporal_embed_size: 32
marker_embed_size: 32
```

For the slot attention layer at the head of Transformer encoder, we use:

```
num_slots: 50
slot_iters: 3
```

Optimizer We use a batch-size of 64 sequences. Within each batch, we use pad the inputs with 0 upto the max sequence length. Our optimizer of choice is SGD with momentum 0.9, weight decay 0.0001 and dampening 0.1. The initial learning rate is 0.01, with exponential decay of 0.9995 per 10 gradient updates. We used early stopping with patience 100.

C.3 Metrics

In Section 3 , we presented packing efficiency (PE) and edit distance (ED) metrics collected on a policy rollout. We present additional metrics about training progress and rollout here.

Category-token Accuracy indicates how well the policy can mimic the expert's action, given the current state. We monitor training progress by matching the predicted instance to the target chosen in demonstration (Fig. 6). We see that TTP is able to predict the same category object to pick perfectly (accuracy close to 1.0). However, this is a simpler setting that sequential decision making. During rollout, any error in a state could create a setting that is out-of-distribution for the policy. Thus, category token accuracy sets an upper bound for rollout performance, that is, while having high category token accuracy is necessary, it is not sufficient for high packing efficiency and inverse edit distance.

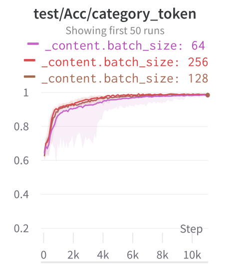


Figure 6: Category level accuracy grouped by batch size for prompt-situation training.

Temporal efficiency:

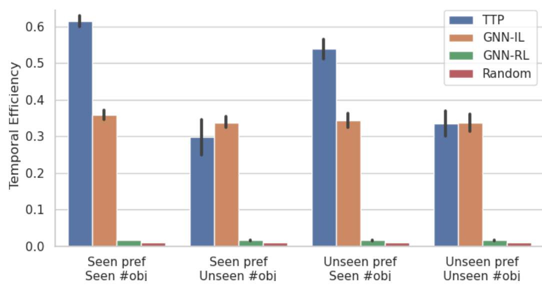


Figure 7: TE (SPL) metric for held-out test settings.

Just like SPL [9] for navigation agents, we define the efficiency of temporal tasks in policy rollout, in order to study how efficient the agent was at achieving the task. For episode $i \in [1, ..N]$, let the agent take p_i number of high-level interactions to execute the task, and the demonstration consists of l_i interactions for the initial state. We scale the packing efficiency PE_i of the policy by the ratio of steps taken by expert versus policy. Temporal efficiency is defined between 0 to 1, and higher is better. This value will be equal to or lower than the packing efficiency. This especially penalizes policies that present a 'looping' behavior, such as repeatedly open/close dishwasher trays, over policies that reach a low PE in shorter episodes (for example, by placing most objects in the sink). Fig 7 shows the temporal efficiency or SPL over our 4 main held-out test settings.

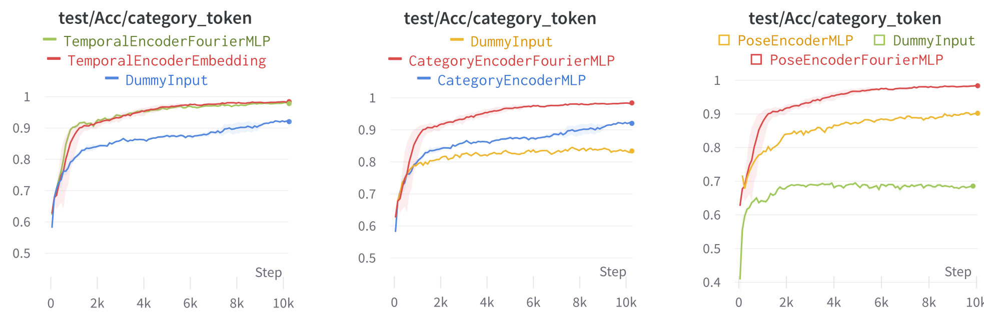
D Additional Ablation Experiments

In Section **??** we presented ablation experiments over number of demonstrations per preference used for training, and the number of unique preferences used. In this Section, we present additional ablation experiments over the design of instance encodings in TTP. Additionally, we also present results where we increase the temporal context of TTP and study its effect on performance.

D.1 Design of Instance Encoding

How much does temporal encoding design matter? Fig. 8a shows that learning an embedding per timestep or expanding it as fourier transformed vector of sufficient size achieves high success. On the other hand, having no timestep input shows slightly lower performance. Timestep helps in encoding the order of the prompt states. The notion of timestep is also incorportated by autoregressive masking in both the encoder and the decoder.

How much does category encoding design matter? In our work, we represent category as the extents of an objects' bounding box. An alternative would be to denote the category as a discrete set of categorical labels. Intuitively, bounding box extents captures shape similarity between objects and their placement implicitly, which discrete category labels do not. Fig. 8b shows that fourier transform of the bounding box achieves better performance than discrete labels, which exceeds the performance with no category input.



(a) Temporal encoding (b) Category encoding (c) Pose encoding

Figure 8: [Left-to-Right] Comparing different design choices of attribute encoders in terms of category token accuracy on held-out test prompt-situation session pairs.

How much does pose encoding design matter? We encode pose as a 7-dim vector that includes 3d position and 4d quaternion. Fig. 8c shows that the fourier transform of the pose encoding performs better than feeding the 7 dim through MLP. Fourier transform of the pose performs better because such a vector encodes the fine and coarse nuances appropriately, which otherwise either require careful scaling or can be lost during SGD training.

D.2 Markov assumption on the current state in partial visibility scenarios

Dynamic settings, as used in our simulation, can be partially observable. For example, when the rack is closed, the policy doesn't know whether it is full or not from just the current state. If a new object arrives, the policy needs to decide between opening the rack if there is space, or dropping the object in sink if the rack is full. In such partially observed settings, the current state may or may not contain all the information needed to reason about the next action. However, given information from states in previous timesteps, the policy can decide what action to take (whether to open the rack or directly place the object in the sink). With this in mind, we train a single preference pick only policy for different context history. As shown in Fig. 10, context window of size k processes the current state as well as k predecessor states, that is, in total $k+1$ states.

Let context history k refer to the number of previous states included in the input. Then the input is a sequence of previous k states' instances (including the current state), as shown in Fig. 10.

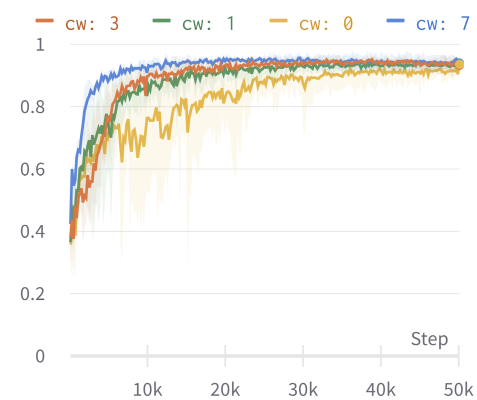


Figure 9: Plot showing category level accuracy for the held-out test sessions for single preference training with context windows. While larger context window size learns faster, the asymptotic performance for all context windows converges in our setting.

Fig 9 shows that TTP gets $> 90\%$ category level prediction accuracy in validation for all context windows. While larger context windows result in faster learning at the start of the training, the asymptotic performance of all contexts is the same. This points to the dataset being largely visible, and a single context window capturing the required information. In the future, we would like to experiment with more complex settings like mobile robots, which might require a longer context.

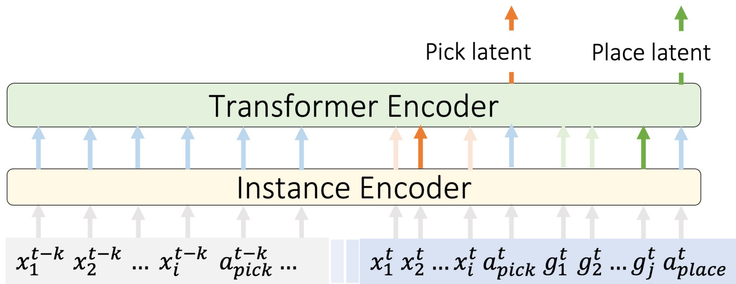


Figure 10: Processing with previous Context History k

E Limitations and Future scope

In Section 6, we briefly discussed the limitations and risks. Here we enlist more details and highlight future directions.

Pick grasping depends on accurate segmentation and edge detection Grasping policy depends on quality of segmentation and edge detection of the selected object. Due to noise in calibration, shadows and reflections, there are errors in detecting the correct edge to successfully grasp the object. For example, it is hard to grasp a plate in real setting. Plate is very close to the ground and the depth cameras cannot detect a clean edge for grasping. Therefore, in our work, we place the plate on an elevated stand for easy grasping. Grasping success also depends on the size and kind of gripper used.

Placement in real setting For placement, the orientation of final pose is often different from initial pose and may require re-grasping. The placement pose at final settlement is different from the robot's end-effector pose while releasing the object from its grasp. Similar to picking, placement accuracy will largely depend on approperiate size and shape of gripper used. Due to these reasons, placement in real world is an open challenging problem and we hope to address this future work.

Hardware pipeline issues due to calibration The resulting point cloud is generated noisy due to two reasons. First, incorrect depth estimation due to camera hardware, lighting conditions, shadows and reflections. Second, any small movements among cameras that affects calibration. If we have a noisy point cloud, it is more likely to have errors in subsequent segmentation and edge detection for grasp policy. Having sufficient coverage of the workspace with cameras is important to mitigate issues due to occlusions and incomplete point clouds.

Incomplete information in prompt The prompt session may not contain all the information to execute on the situation. For example, in a prompt session there might be no large plates seen, which in incomplete/ambiguous information for the policy. This can be mitigated by ensuring complete information in prompt demo or having multiple prompts in slightly different initialization.

References

[1] L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik. Intel realsense stereoscopic depth cameras. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 1–10, 2017.

[2] M. Fiala. Artag, a fiducial marker system using digital techniques. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 590–596. IEEE, 2005.

[3] Q.-Y. Zhou, J. Park, and V. Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018.

[4] Y. Xiang, C. Xie, A. Mousavian, and D. Fox. Learning rgb-d feature embeddings for unseen object instance segmentation. *arXiv preprint arXiv:2007.15157*, 2020.

[5] 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.

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

[7] H.-S. Fang, C. Wang, M. Gou, and C. Lu. Graspnet-1billion: A large-scale benchmark for general object grasping. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11444–11453, 2020.

[8] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, O. Maksymets, A. Gokaslan, V. Vondrus, S. Dharur, F. Meier, W. Galuba, A. Chang, Z. Kira, V. Koltun, J. Malik, M. Savva, and D. Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[9] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.