Transformers are adaptable task planners

Anonymous Author(s)
Affiliation
Address
email

Abstract: Every home is different, and every person likes things done in their particular way. Therefore, home robots of the future need to both reason about the sequential nature of day-to-day tasks and generalize to user's preferences. To this end, we propose a Transformer Task Planner (TTP) that learns high-level actions from demonstrations by leveraging object attribute-based representations. TTP can be pre-trained on multiple preferences and shows generalization to *unseen* preferences using a single demonstration as a prompt in a simulated dishwasher loading task. Further, we demonstrate real-world dish rearrangement using TTP with a Franka Panda robotic arm, prompted by a single human demonstration.

Keywords: Task Planning, Prompt, Preferences, Object-centric Representation

1 Introduction

Consider a robot tasked with loading a dishwasher. Such a robot has to account for task constraints (e.g. only an open dishwasher rack can be loaded), and dynamic environments (e.g. more dishes may arrive once the robot starts loading the dishwasher). Dishwasher loading is a canonical example of personal preferences, where everyone has a different approach which the robot should adapt to. Classical task planning deals with task constraints through symbolic task description, but such descriptions are difficult to design and modify for new preferences in complex tasks. Building easily adaptable long-horizon task plans, under constraints and uncertainty, is an open problem in robotics.

Machine learning (ML) enables learning complex tasks without extensive expert intervention: robotic navigation [1, 2, 3], in-hand manipulation [4, 5, 6, 7, 8], and planning [9, 10, 11, 12, 13]. Within task planning, ML is used to replace user-defined symbolic descriptions [14], deal with uncertainty [15], and adapt to preferences [16]. Recent work [17] has shown Transformer networks [18] can learn temporally-consistent representations, and exhibit generalization to new scenarios [19, 20, 21]. Our central question is: *Can a Transformer network learn task structure, adapt to user preferences, and achieve complex long-horizon tasks using no symbolic task representations?*

We hypothesize that task structure and preference are implicitly encoded in demonstrations. When loading a dishwasher, a user pulls out a rack before loading it, inherently encoding a structural constraint. They may place mugs on the top rack and plates on the bottom, encoding their preference. Learning user preferences from long-horizon demonstrations requires policies with temporal context. For example, a user might prefer to load the top rack before the bottom tray. The policy needs to consider the *sequence of actions* demonstrated, rather than individual actions. Transformers are well-suited to this problem, as they have been shown to learn long-range relationships [22], although not in temporal robotic tasks. We propose Transformer Task Planner (TTP) - an adaptation of a classic transformer architecture that includes temporal-, pose- and category-specific embeddings to learn object-oriented relationships over space and time. TTP generalizes beyond what was seen in demonstrations – to variable numbers of objects and dynamic environments. By pre-training TTP on multiple preferences, we build temporal representations that can be generalized to new preferences. The main contributions of our work are: (1) Introduce transformers as a promising architecture for learning task plans from demonstrations, using object-centric embeddings (2) Demonstrate that preference conditioned pre-training generalizes at test time to new, unseen preferences with a single

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

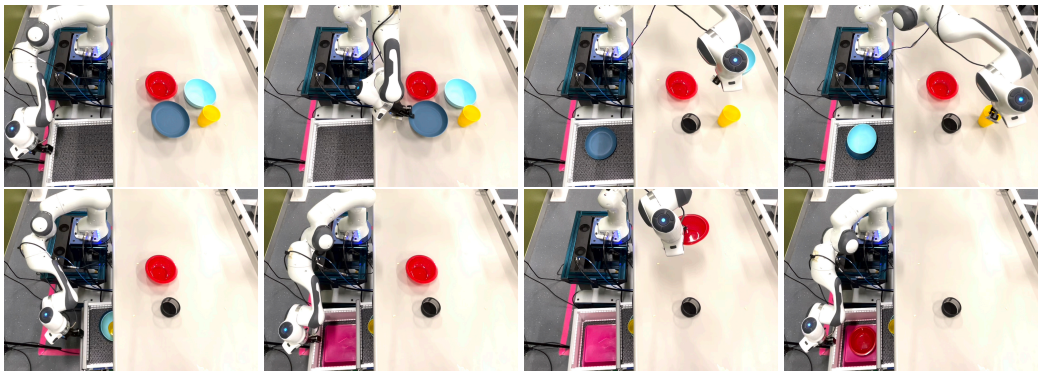


Figure 1: (Left-to-Right, Top-to-bottom) A Franka Emika Panda arm organizing 4 dishes into 2 drawers, following preference shown in a demonstration. The robot opens a top drawer, places objects in the top drawer, closes it, and does the same for the bottom drawer. The high-level policy that makes decisions about when to open drawers, what objects to pick, etc is learned in simulation and transferred zero-shot to the real-world.

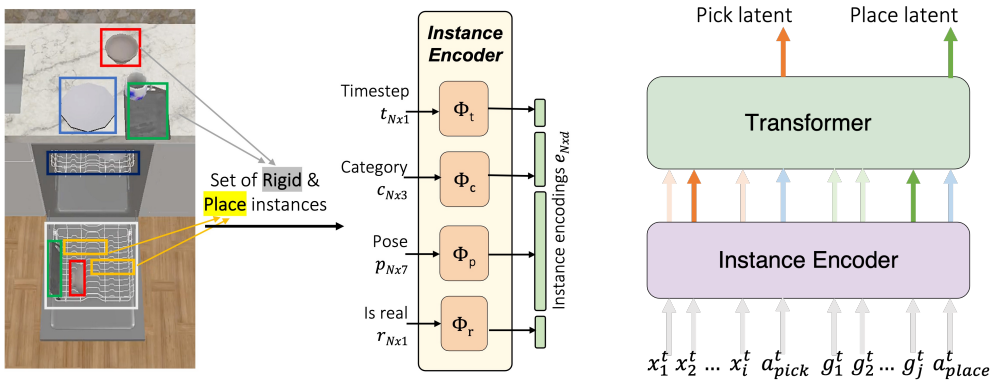
user demonstration. Our experiments use a complex high-dimensional dishwasher loading environment (Fig. 4) with several challenges: complex task structure, dynamically appearing objects and human-specific preferences. TTP successfully learns this task from seven preferences with 80 demonstrations each, and generalizes to unseen scenes and preferences and outperforms competitive baselines [23, 24]. Finally, we transfer TTP to a rearrangement problem in the real-world, where a Franka arm places dishes in two drawers, using a single human demonstration (Fig. 1).

2 Transformer Task Planner (TTP)

We introduce TTP, a Transformer-based policy architecture for learning sequential manipulation tasks. We assume low-level 'generalized' pick-place primitive actions that apply to both objects like plates, bowls, and also to dishwasher door, racks, etc. TTP learns a high-level policy for pick-place in accordance with the task structure and preference shown in demonstrations. The following sections are described with dishwasher-loading as an example, but our setup is applicable to most long-horizon manipulation tasks with 'generalized' pick-place.

State-Action Representations We consider a high-level policy that interacts with the environment at discrete time steps. At every timestep t, we receive observation $\boldsymbol{o}_t$ from the environment which is passed through a perception pipeline to produce a set of rigid-body instances $\{x_i\}_{i=1}^{n}$, corresponding to the n objects currently visible. We express an *instance* as: $x_i = \{p_i, c_i, t\}$, where p_i is the pose of the object, c_i is its category, and t is the timestep while recording $\boldsymbol{o}_t$ (Fig. 2a). For example, for a bowl at the start of an episode, p is its current location w.r.t a global frame, c is its category (bowl), and $t = 0$. The pick *state* S_t^{pick} is described as the set of instances visible in $\boldsymbol{o}_t$: $S_t^{pick} = \{x_0, x_1, \cdots, x_n\}$. S_t^{pick} is passed to a learned policy π to predict a pick action $\boldsymbol{a}_t$. We describe the *action* $\boldsymbol{a}_t$ in terms of what to pick and where to place it. Specifically, the pick action chooses one instance from x_i observed in $\boldsymbol{o}_t$: $\pi(S^{pick}) \rightarrow x_{target}$, where $x_{target} \in \{x_i\}_{i=1}^{n}$.

Once a pick object is chosen, a similar procedure determines where to place it. We pre-compile a list of discrete placement poses corresponding to viable place locations for each object-category. These poses densely cover the dishwasher racks, and are created by randomly placing objects in the dishwasher and measuring the final pose they land in. All possible placement locations for the picked object category, whether free or occupied, are used to create a set of place instances $\{g_j\}_{j=1}^{l}$. Similar to x_i, $g_j = \{p_j, c_j, t\}$, consists of a pose, category and timestep. A boolean value r in the attribute set distinguishes the two instance types. The place state is $S^{place} = S^{pick} \cup \{g_j\}_{j=1}^{l}$. The *same* policy π then chooses where to place the object (Fig.2). Note that the input for predicting place includes both objects and place instances, since the objects determine whether a place instance is free to place or not. x_{target} and g_{target} together make action $\boldsymbol{a}$, sent to a low-level pick-place policy. The policy π is modeled using a Transformer [18].



(a) Scene to Instance embeddings

(b) Instances to prediction

Figure 2: (Left) Architecture overview of how a scene is converted to a set of instances. Each instance is comprised of attributes, i.e. pose, category, timestep, and whether it is an object or place instance. (Right) Instance attributes (gray) are passed to the encoder, which returns instance embeddings for pickable objects (red), placeble locations (green), and <ACT> embeddings (blue). The transformer outputs a chosen pick (red) and place (green) instance embedding.

Instance Encoder Each object and goal instance x_i, g_j is projected into a higher-dimensional vector space (Fig. 2a). Such embeddings improve the performance of learned policies [18, 25]. For the pose embedding Φ_p, we use a positional encoding scheme similar to NeRF [25] to encode the 3D positional coordinates and 4D quaternion rotation of an instance. For category, we use the dimensions of the 3D bounding box of the object to build a continuous space of object types and process this through an MLP Φ_c. For each discrete 1D timestep, we model Φ_t as a learnable embedding in a lookup table, similar to the positional encodings in BERT [26]. To indicate whether an instance is an object or placement location, we add a 1D boolean value vectorized using a learnable embedding function Φ_r. The concatenated d-dimensional embedding for an instance at timestep t is represented as $f_e(x_i) = \Phi_t||\Phi_c||\Phi_p||\Phi_r = e_i$. The encoded state at time t can be represented in terms of instances as: $S_t^{enc} = [e_0, e_1, \cdots e_N]$. We drop $()^{enc}$ for brevity.

Demonstrations are state-action sequences $\mathcal{C} = \{(S_0, a_0), (S_1, a_1), \cdots, (S_{T-1}, a_{T-1}), (S_T)\}$. Here S_i is the set of object and place instances and a_i are the pick-place actions chosen by expert at time i. At every time step, we record the state of the objects, the pick instance chosen by the expert, place instances for the corresponding category, and the place instance chosen by the expert. Expert actions are assumed to belong to the set of visible instances in S_i. However, different experts can exhibit different preferences over a_i. For example, one expert might choose to load bowls first in the top rack, while another expert might load bowls last in the bottom rack. In the training dataset, we assume labels for which preference each demonstration belongs to, based on the expert used for collecting the demonstration. Given K demonstrations per preference $m \in \mathcal{M}$, we have a dataset for preference m: $\mathcal{D}_m = \{\mathcal{C}_1, \cdots, \mathcal{C}_K\}$. The complete training dataset consists of demonstrations from all preferences: $\mathcal{D} = \bigcup_{m=1}^{M} \mathcal{D}_m$. During training, we learn a policy that can reproduce all demonstrations in our dataset. This is challenging, since the actions taken by different experts are different for the same input, and the policy needs to disambiguate the preference. At test time, we generalize the policy to both unseen scenes and unseen preferences.

2.1 Prompt-Situation Transformer

We use Transformers [18], a deep neural network that operates on sequences of data, for learning a high-level pick-place policy π. The input to the encoder is a d-dimensional token[1] per instance $e_i \in [1, ..N]$ in the state S. In addition to instances, we introduce special <ACT> tokens[2], a zero vector for all attributes, to demarcate the end of one state and start of the next. These

[1]Terminology borrowed from natural language processing where tokens are words; here, they are instances.
[2]Similar to $< CLS >$ tokens used for sentence classification.

tokens help maintain the temporal structure; all instances between two <ACT> tokens in a sequence represent one observed state. A trajectory τ, without actions, is: $\tau = \left[S_{t=0}, \text{<ACT>}, \cdots, S_{t=T-1}, \text{<ACT>}, S_{t=T}\right]$. To learn a common policy for multiple preferences, we propose a prompt-situation architecture (Fig. 3). The prompt encoder receives one demonstration trajectory as input, and outputs a learned representation of the preference. These output prompt tokens are input to a situation decoder, which also receives the current state as input. The decoder π is trained to predict the action chosen by the expert for the situation, given a prompt demonstration. The left half is prompt encoder ψ and the right half is the situation decoder or policy π acting on given state. The prompt encoder $\psi : f_{slot} \circ f_{te} \circ f_e$ consists of an instance encoder f_e, transformer encoder f_{te}, and a slot-attention layer f_{slot} [27]. ψ takes the whole demonstration trajectory τ_{prompt} as input and returns a fixed and reduced preference embedding $\gamma = \psi(\tau_{\text{prompt}})$ of sequence length H. Slot attention is an information bottleneck, which learns semantically meaningful representations of the prompt.

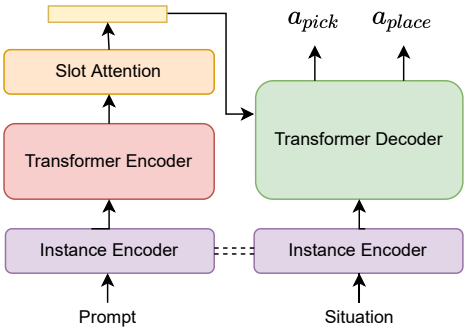


Figure 3: Prompt-Situation Architecture. The left is the prompt encoder which takes as input a prompt demonstration and outputs a learned preference embedding. The right half is the situation decoder, which conditioned on preference embedding from prompt encoder, acts on the current state.

The situation decoder is a policy $\pi : f_{td} \circ f_e$ that receives as input N instance tokens from the current scene S, consisting of objects, as well as, placement instances separated by <ACT> tokens (Fig. 2b). The policy architecture is a transformer decoder [18] with self-attention layers over the N input tokens. This is followed by a cross-attention layer with preference embedding γ (H tokens) from the prompt encoder. We select the output of the situation decoder at the <ACT> token and calculate dot-product similarity with the N input tokens e_i. The token with the maximum dot-product is chosen as the predicted instance: $x_{pred} = \max_{e_i, i \in \{1,N\}} \left(\hat{x}_{\text{<ACT>}} \cdot e_i\right)$. The training target is extracted from the demonstration dataset $\mathcal{D}$, and the policy trained with cross-entropy to maximize the similarity of output latent with the expert's chosen instance embedding.

2.2 Multi-Preference Task Learning

We adopt a prompt-situation architecture for multi-preference learning. This design (1) enables multi-preference training by disambiguating preferences, (2) learns task-level rules shared between preferences (e.g. dishwasher should be open before placing objects), (3) can generalize to unseen preferences at test time, without fine-tuning. Given a 'prompt' demonstration of preference m, our policy semantically imitates it in a different 'situation' (i.e. a different initialization of the scene). To this end, we learn a representation of the prompt γ^m which conditions π to imitate the expert.

$$\psi(\tau^m_{\text{prompt}}) \rightarrow \gamma^m \tag{1}$$

$$\pi(S_{\text{situation}} | \gamma^m) \rightarrow a^m_t = \{x^m_{\text{pred}}, g^m_{\text{pred}}\} \tag{2}$$

We train neural networks ψ and π together to minimize the total prediction loss over all preferences using the multi-preference training dataset $\mathcal{D}$. The overall objective becomes:

$$\min_{m \sim M, \tau \sim \mathcal{D}_m, (S,a) \sim \mathcal{D}_m} \mathcal{L}_{CE}(a, \pi(S, \psi(\tau))) \tag{3}$$

For every preference m in the dataset, we sample a demonstration from $\mathcal{D}_m$ and use it as a prompt for all state-action pairs (S, a) in $\mathcal{D}_m$. This includes the state-action pairs from τ_{prompt} and creates a combinatorially large training dataset. At test time, we record one prompt demo from a seen or unseen preference and use it to condition π and $\psi : a = \pi(S, \psi(\tau_{\text{prompt}}))$. All policy weights are kept fixed during testing, and generalization to new preferences is zero-shot using the learned preference representation γ. Unlike [24], γ captures not just the final state, but a temporal representation of the whole demonstration. Building a temporal representation is crucial to encode demonstration

Figure 4: Dishwasher Loading demonstration in AI Habitat Kitchen Arrange Simulator. Objects dynamically appear on the counter-top (ii-iv), and need to be placed in the dishwasher. If the dishwasher racks are full, they land in sink (v)

preferences like order of loading racks and objects. Even though the final state is the same for two preferences that only differ in which rack is loaded first, our approach is able to distinguish between them using the temporal information in τ_{prompt}. To the best of our knowledge, our approach is the first to temporally encode preferences inferred from a demonstration in learned task planners.

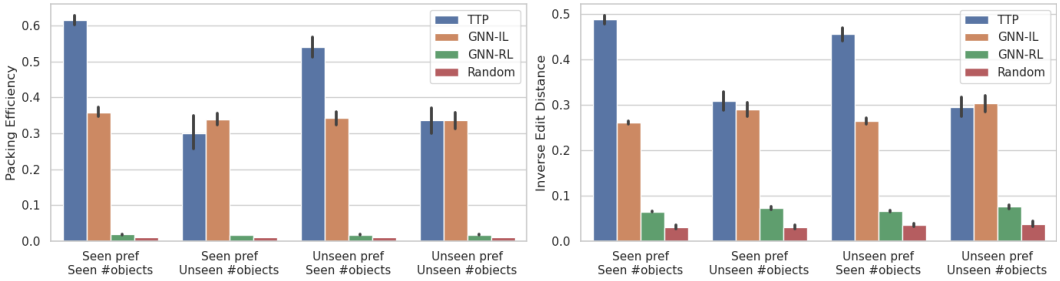
3 Experiments

We present the "Replica Synthetic Apartment 0 Kitchen"[3] (see figure 4, appendix and video), an artist-authored interactive recreation of the kitchen of the "Apartment 0" space from the Replica dataset [28]. We use selected objects from the ReplicaCAD [29] dataset, including seven types of dishes, and generate dishwasher loading demonstrations using an expert-designed data generation script (see Appendix B.1). Given 7 categories of dishes and two choices in which rack to load first, the hypothesis space of possible preferences is $2 \times 7!$. Our dataset consists of 12 preferences (7 train, 5 held-out test) with 100 sessions per preference. In a session, $n \in \{3, ..., 10\}$ instances are loaded in each rack. The training data consists of sessions with 6 or 7 objects allowed per rack. The held-out test set contains 5 unseen preferences and settings for $\{3, 4, 5, 8, 9, 10\}$ objects per rack. Additionally, to simulate a dynamic environment, we randomly initialize new objects mid-session on the kitchen counter. This simulates situations where the policy does not have full information of every object to be loaded at the start of the session, and has to learn to be reactive to new information.

We train a 2-head 2-layer Transformer encoder-decoder with 256 input and 512 hidden dimensions, and 50 slots and 3 iterations for Slot Attention (more details in Appendix C). We test both in- and out-of-distribution performance in simulation. For in-distribution evaluation, 10 sessions are held-out for testing for each training preference. For out-of-distribution evaluation, we create sessions with unseen preferences and unseen number of objects. We evaluate trained policies on 'rollouts' in the simulation, a more complex setting than accuracy of prediction. Rollouts require repeated decisions in the environment, without any resets. A mistake made early on in a rollout session can be catastrophic, and result in poor performance, even if the prediction accuracy is high. For example, if a policy mistakenly fails to open a dishwasher rack, the rollout performance will be poor, despite good prediction accuracy. To measure success, we rollout the policy from an initial state and compare the performance of the policy with an expert demonstration from the same initial state. Note that the policy does not have access to the expert demonstration, and the demonstration is only used for evaluation. Specifically, we measure (1) how well is the final state packed and (2) how much did the policy deviate from the expert demonstration?

Packing efficiency: We compare the number of objects placed in the dishwasher by the policy to that in the expert's demonstration. Let a_i be number of objects in top rack and b_i be objects on bottom rack placed by the expert in the ith demonstration. If the policy adheres to the preference and places $\hat{a}_i$ and $\hat{b}_i$ on the top and bottom respectively, then the *packing efficiency (PE)* $= \sum_i \left(\frac{\hat{a}_i}{max(\hat{a}_i, a_i)} + \frac{\hat{b}_i}{max(\hat{b}_i, b_i)} \right)$. Packing efficiency is between 0 to 1, and higher is better. Note that if the policy follows the wrong preference, then PE is 0, even if the dishwasher is full.

[3] "Replica Synthetic Apartment 0 Kitchen" was created with the consent of and compensation to artists, and will be shared under a Creative Commons license for non-commercial use with attribution (CC-BY-NC).



(a) Packing efficiency (b) Inverse Edit Distance

Figure 5: Comparisons of TTP, GNN-IL/RL and a Random policy in simulation across two metrics. TTP shows good performance at the task of dishwasher loading on seen and unseen preferences, and outperforms the GNN and random baselines. However TTP's generalization to unseen # objects if worse, but still close to GNN-IL, and better than GNN-RL and random.

Inverse Edit distance: We also calculate is the *inverse edit distance* between the sequence of actions taken by the expert versus the learned policy. We compute the Levenshtein distance[4] (LD) between the policy's and expert's sequence of pick and place instances. Inverse edit distance is defined as $ED = 1 - LD$; higher is better. This measures the temporal deviation from the expert, instead of just the final state. If the expert preference was to load the top rack first and the learned policy loads the bottom first, PE would be perfect, but inverse edit distance would be low.

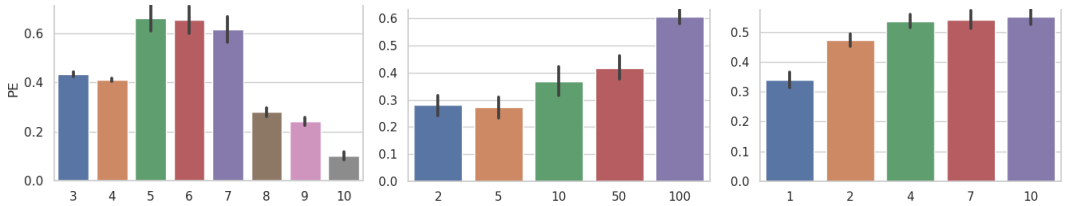
3.1 Evaluation on simulated dishwasher loading

Baselines: We compare our approach against Graph Neural Network (GNN) based preference learning from [23] and [24]. Neither of these works are directly suitable for our task, so we combine them to create a stronger baseline. We use ground-truth preference labels, represented as a categorical distribution and add them to the input features of the GNN, similar to [24]. **For unseen preferences, this is privileged information** that our approach does not have access to. We use the output of the GNN to make sequential predictions, as in [23]. Thus, by combining the two works and adding privileged information about ground-truth preference category, we create a GNN baseline which can act according to preference in a dishwasher loading scenario. We train this policy using imitation learning (IL), and reinforcement learning (RL), following [23]. GNN-IL is trained from the same set of demonstrations as TTP using behavior cloning. For GNN-RL, we use Proximal Policy Optimization [30] from [31]. GNN-RL learns from scratch by directly interacting with the dishwasher environment and obtaining a dense reward. For details, see Appendix C.1.

GNN-IL does not reach the same performance as TTP for in-distribution tasks (PE of 0.34 for GNN-IL vs 0.62 for TTP). Note that unseen preferences are also in-distribution for GNN-IL since we provide ground-truth preference labels to the GNN. Hence, there isn't a drop in performance for unseen preferences for GNN, unlike TTP. Despite having no privileged information, TTP outperforms GNN-IL in unseen preferences, and performs comparably on unseen #objects. Due to the significantly long time horizons per session (more than 30 steps), GNN-RL fails to learn a meaningful policy even after a large budget of 32,000 environment interactions and a dense reward (PE 0.017 ± 0.002 using GNN-RL). Lastly, we find that the random policy RP is not able to solve the task at all due to the large state and action space. TTP is able to solve dishwasher loading using unseen preferences well (PE 0.54). In contrast, classical task planners like [32] need to be adapted per new preference. This experiment shows that Transformers make adaptable task planners, using our proposed prompt-situation architecture. However, TTP's performance on unseen #objects deteriorates (0.62 for seen versus 0.34 on unseen #objects), and we look more closely at that next.

Generalization to unseen # objects: Fig. 6a examines PE on out-of-distribution sessions with lesser i.e. 3-5 or more i.e. 8-10 objects per rack. The training set consists of demonstrations with 6 or 7 objects per rack. The policy performs well on 5-7 objects, but poorer as we go further away from the

[4] `Levenshtein distance= textdistance(learned_seq, expert_seq) / len(expert_seq)`



(a) Performance decays for OOD number of objects, i.e 3-5 & 8-10

(b) Performance improves with the # of unique sessions in training

(c) Performance improves with the # of unique preferences in training

Figure 6: (a) Out-of-distribution generalization to #objects. (b-c) ablation experiments.

training distribution. Poorer PE is expected for larger number of objects, as the action space of the policy increases, and the policy is more likely to pick the wrong object type for a given preference. Poorer performance on 3-4 objects is caused by the policy closing the dishwasher early, as it has never seen this state during training. Training with richer datasets, and adding more randomization in the form of object masking might improve out-of-distribution performance of TTP.

3.2 Real-world dish-rearrangement experiments

We zero-shot transfer our policy 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 the Polymetis control framework [33]. For perception, we find the extrinsics of three Intel Realsense D435 RGBD cameras [34] using ARTags [35]. The camera output, extrinsics, and intrinsics are combined using Open3D [36] and fed into a segmentation pipeline [37] to generate object categories. For low-level pick, we use a grasp candidate generator [38] applied to the object point cloud, and use it to grasp the target object. Place is approximated as a 'drop' action in a pre-defined location. 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. 1).

We use a policy trained in simulation and apply it to a scene with four objects (2 bowls, 1 cup, 1 plate) through the hardware pipeline described above. We start by collecting a prompt human demonstration (more details in Appendix A)., The learned policy, conditioned a prompt demonstration, is applied to two variations of the same scene, and the predicted actions executed. The policy was successful once with 100% success rate, and once with 75%, shown in Figure 1, bottom. The failure case was caused due to a perception error – a bowl was classified as a plate. This demonstrates that such TTP can be trained in simulation and applied directly to hardware. The policy is robust to minor hardware errors, such as if a bowl grasp fails, it just repeats the grasping action (see video and Appendix A). However, it relies on accurate perception of the state. 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.

3.3 Ablations

We study the sensitivity of our approach to training hyperparameters. First, we vary the number of training sessions per preference and study generalization to unseen scenarios of the same preference. Figure 6b shows that the performance of TTP improves as the number of demonstrations increase, indicating that our model is not overfitting to the training set, and might benefit from further training samples. Next, we vary the number of training preferences and evaluate generalization performance to unseen preferences. Figure 6c shows that the benefits of adding additional preferences beyond 4 are minor, and similar performance is observed when training from 4-7 preferences. This is an interesting result, since one would as-

Table 1: Attribute Ablations

Pose	Cat	Time	PE
×	×	×	0.0
×	×	✓	0.0
×	✓	×	0.027
×	✓	✓	0.142
✓	×	×	0.411
✓	×	✓	0.419
✓	✓	×	0.517
✓	✓	✓	0.606

sume that more preferences improve generalization to unseen preferences. But for the kinds of preferences considered in our problem, 4-7 preference types are enough for generalization.

Finally, we analyze which instance attributes are the most important for learning in a object-centric sequential decision making task. We mask different instance attributes to remove sources of information from the instance tokens. As seen in Table 1, all components of our instance tokens play significant role (0.606 with all, versus the next highest of 0.517). The most important attribute is pose of the objects (without the pose, top PE is 0.142), followed by the category. The timestep is the least important, but the best PE comes from combining all three. (more details in Appendix D.1).

4 Prior Work

Object-centric representations for sequential manipulation Several works build object-centric pick-place representations using off-the-shelf perception methods [39, 40, 41, 42, 43, 44]. Once estimated, object states are used by a task and motion planner for sequential manipulation [32, 45], but objects in the scene are known. [46] combine motor learning with object-centric representation, but transfer of policies is challenging. Transporter Nets [47] use a visual encoder-decoder for table-top manipulation tasks; SORNet [48] extracts object-centric representations from RGB images and demonstrates generalization in sequential manipulation task. Inspired from these, we learn policies for dishwasher loading, choosing from visible objects to make pick-place decisions.

Transformers for sequence modeling Transformers in NLP [18] and vision [49] have focused on self-supervised pretraining due to abundant unsupervised data available. Recent works have repurposed transformers for other sequence modeling tasks [50, 51, 52, 53, 54, 55]. PlaTe [51] proposes planning from videos, while [53, 54, 55] model a sequential decision-making task. We consider long-horizon tasks with partially observable state features, and user-specific preferences.

Preferences and prompt training In literature, there are several ways of encoding preferences. [24] propose VAE to learn user preferences for spatial arrangement based on just the final state, while our approach models temporal preference from demonstrations. Preference-based RL learns rewards based on human preferences [56, 57, 58, 59], but do not generalize to unseen preferences. On complex long-horizon tasks, modeling human preferences enables faster learning than RL, even with carefully designed rewards [60]. We show generalization to unseen preferences by using prompts. Large language and vision models have shown generalization through prompting [61, 62]. Prompting can also be used to guide a model to quickly switch between multiple task objectives [63, 64, 65]. Specifically, language models learn representations that be easy transferred to new tasks in a few-shot setting [66, 67, 19, 61]. Our approach similarly utilizes prompts for preference generalization in sequential decision-making robotic tasks.

5 Conclusions and Limitations

We present Tranformer Task Planner (TTP): a high-level, sequential, preference-based policy from a single demonstration using a prompt-situation architecture. We introduced a simulated dishwasher loading dataset with demonstrations that adhere to varying preferences. TTP can solve a complex, long-horizon dishwasher-loading task in simulation and transfer to the real world.

We have demonstrated the TTP's strong performance in the dishwasher setting. This environment is both complex by virtue of its strict sequential nature and yet incomplete as we assume doors and drawers can be easily opened and perception is perfect. In real settings, the policy needs to learn how to recover from its low-level execution mistakes. More complex preferences may depend on differences in visual or textural patterns on objects, for which the instance encoder would require modifications to encode such attributes. An important question to address is how more complex motion plans interact with or hinder the learning objective, especially due to different human and robot affordances. Finally, prompts are only presented via demonstration, while language might be a more natural interface for users.

References

[1] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[2] J. Truong, D. Yarats, T. Li, F. Meier, S. Chernova, D. Batra, and A. Rai. Learning navigation skills for legged robots with learned robot embeddings. 2020.

[3] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *Robotics: Science and Systems (RSS)*, 2021.

[4] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation (2018). *arXiv preprint arXiv:1806.10293*, 2018.

[5] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[6] F. Wirnshofer, P. S. Schmitt, G. von Wichert, and W. Burgard. Controlling contact-rich manipulation under partial observability. In *Robotics: Science and Systems*, 2020.

[7] Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese. Keto: Learning keypoint representations for tool manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7278–7285. IEEE, 2020.

[8] A. Simeonov, Y. Du, B. Kim, F. R. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. *arXiv preprint arXiv:2011.08177*, 2020.

[9] G. Yang, A. Zhang, A. Morcos, J. Pineau, P. Abbeel, and R. Calandra. Plan2vec: Unsupervised representation learning by latent plans. In *Learning for Dynamics and Control*, pages 935–946. PMLR, 2020.

[10] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. *arXiv preprint arXiv:2010.11944*, 2020.

[11] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[12] D. Driess, J.-S. Ha, and M. Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. *arXiv preprint arXiv:2006.05398*, 2020.

[13] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175. PMLR, 2017.

[14] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Perez, L. P. Kaelbling, and J. Tenenbaum. Inventing relational state and action abstractions for effective and efficient bilevel planning. *arXiv preprint arXiv:2203.09634*, 2022.

[15] D. Gordon, D. Fox, and A. Farhadi. What should i do now? marrying reinforcement learning and symbolic planning. *arXiv preprint arXiv:1901.01492*, 2019.

[16] R. Kaushik, T. Anne, and J.-B. Mouret. Fast online adaptation in robotics through meta-learning embeddings of simulated priors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5269–5276. IEEE, 2020.

[17] R. Kaplan, C. Sauer, and A. Sosa. Beating atari with natural language guided reinforcement learning. *arXiv preprint arXiv:1704.05539*, 2017.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[19] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[20] V. Sanh, A. Webson, C. Raffel, S. H. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, T. L. Scao, A. Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

[21] X. Liu, K. Ji, Y. Fu, Z. Du, Z. Yang, and J. Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

[22] D. S. Chaplot, D. Pathak, and J. Malik. Differentiable spatial planning using transformers. In *International Conference on Machine Learning*, pages 1484–1495. PMLR, 2021.

[23] Y. Lin, A. S. Wang, E. Undersander, and A. Rai. Efficient and interpretable robot manipulation with graph neural networks. *IEEE Robotics and Automation Letters*, 2022.

[24] I. Kapelyukh and E. Johns. My house, my rules: Learning tidying preferences with graph neural networks. In *Conference on Robot Learning*, pages 740–749. PMLR, 2022.

[25] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[27] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention. *Advances in Neural Information Processing Systems*, 33:11525–11538, 2020.

[28] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. D. Nardi, M. Goesele, S. Lovegrove, and R. Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[29] 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.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

[32] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *arXiv preprint arXiv:2010.01083*, 2020.

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

[34] 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.

[35] 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.

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

[37] 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.

[38] 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.

[39] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2642–2651, 2019.

[40] X. Deng, Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox. Self-supervised 6d object pose estimation for robot manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3665–3671. IEEE, 2020.

[41] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 1386–1383. IEEE, 2017.

[42] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis. Single image 3d object detection and pose estimation for grasping. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3936–3943. IEEE, 2014.

[43] Y. Yoon, G. N. DeSouza, and A. C. Kak. Real-time tracking and pose estimation for industrial objects using geometric features. In *2003 IEEE International conference on robotics and automation (cat. no. 03CH37422)*, volume 3, pages 3473–3478. IEEE, 2003.

[44] V. Jain, P. Agarwal, S. Patil, and K. Sycara. Learning embeddings that capture spatial semantics for indoor navigation. *arXiv preprint arXiv:2108.00159*, 2021.

[45] C. Paxton, N. Ratliff, C. Eppner, and D. Fox. Representing robot task plans as robust logical-dynamical systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5588–5595. IEEE, 2019.

[46] P. Florence, L. Manuelli, and R. Tedrake. Self-supervised correspondence in visuomot or policy learning. *IEEE Robotics and Automation Letters*, 5(2):492–499, 2019.

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

[48] W. Yuan, C. Paxton, K. Desingh, and D. Fox. SORNet: Spatial object-centric representations for sequential manipulation. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=mOLu2rODIJF.

[49] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[50] W. Liu, C. Paxton, T. Hermans, and D. Fox. Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects. *ArXiv*, abs/2110.10189, 2021.

[51] J. Sun, D.-A. Huang, B. Lu, Y.-H. Liu, B. Zhou, and A. Garg. Plate: Visually-grounded planning with transformers in procedural tasks. *IEEE Robotics and Automation Letters*, 7(2): 4924–4930, 2022.

[52] V. Jain, R. Jena, H. Li, T. Gupta, D. Hughes, M. Lewis, and K. Sycara. Predicting human strategies in simulated search and rescue task. *arXiv preprint arXiv:2011.07656*, 2020.

[53] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

[54] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

[55] A. L. Putterman, K. Lu, I. Mordatch, and P. Abbeel. Pretraining for language conditioned imitation with transformers, 2022. URL https://openreview.net/forum?id=eCPCn25gat.

[56] X. Wang, K. Lee, K. Hakhamaneshi, P. Abbeel, and M. Laskin. Skill preferences: Learning to extract and execute robotic skills from human feedback. In *Conference on Robot Learning*, pages 1259–1268. PMLR, 2022.

[57] K. Lee, L. Smith, A. Dragan, and P. Abbeel. B-pref: Benchmarking preference-based reinforcement learning. *arXiv preprint arXiv:2111.03026*, 2021.

[58] X. Liang, K. Shu, K. Lee, and P. Abbeel. Reward uncertainty for exploration in preference-based reinforcement learning. *arXiv preprint arXiv:2205.12401*, 2022.

[59] W. B. Knox, S. Hatgis-Kessell, S. Booth, S. Niekum, P. Stone, and A. Allievi. Models of human preference for learning reward functions. *arXiv preprint arXiv:2206.02231*, 2022.

[60] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[61] Y. Chen, Y. Liu, L. Dong, S. Wang, C. Zhu, M. Zeng, and Y. Zhang. Adaprompt: Adaptive model training for prompt-based nlp. *arXiv preprint arXiv:2202.04824*, 2022.

[62] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[63] C. Raffel, N. M. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683, 2020.

[64] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*, 2020.

[65] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu. Mass: Masked sequence to sequence pre-training for language generation. In *ICML*, 2019.

[66] T. Schick and H. Schütze. It's not just size that matters: Small language models are also few-shot learners. *arXiv preprint arXiv:2009.07118*, 2020.

[67] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*, 2021.

[68] O. J. Hénaff, A. Srinivas, J. De Fauw, A. Razavi, C. Doersch, S. Eslami, and A. van den Oord. Data-efficient image recognition with contrastive predictive coding. arxiv e-prints. *arXiv preprint arXiv:1905.09272*, 2019.

[69] 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.