A Prior Work: Summary and Challenges

Goal-conditioned behavior cloning on play trajectories $\tau \sim D_{\text{play}}$ (Play-GCBC) is quite effective at learning a robust multi-task policy; however, since the policy is only conditioned on the goal, it fails to capture all the degrees of behavior variation that would achieve that goal [3]. Play-LMP addresses this by introducing a latent plan to capture not just *what* goal to reach, but also *how* to reach it. At a high level, Play-LMP learns to represent sampled play trajectories $\tau \sim D_{\text{play}}$ of fixed horizon H as plans in a latent space, denoted by vector z, using variational inference techniques. The method trains a behavior cloning agent to reproduce the humans actions $a_{1:H} \in \tau$ conditioned on the current plan z and the hindsight-labelled goal state o_H.

At a high level, Play-LMP encodes random sequences of environment states and robot actions as latent "plans," which then get passed to a policy that learns to decode these plans at each state into the corresponding human demonstrated action for that time step. They simultaneously learn to predict the latent plan from just the initial and final state in the environment, for use at test time. In practice, the sequences are uniformly sampled 1-2 second chunks from play. Our method, PLATO, samples *interactions* and intelligently learns a latent space from them, enable longer and variable horizon views of sequences of play.

Architecture: To represent play trajectories $\tau \sim D_{\text{play}}$ as plans, Play-LMP uses a posterior encoder E that maps the states $s_{1:H} \in \tau$ to a single latent plan distribution $z \sim \mathcal{N}(\mu_z, \sigma_z^2)$. To regularize the posterior, Play-LMP simultanously learns prior network E' that takes in just the first state s_1 and the goal environment state $s_H^o \in S^o$ to produce the latent plan distribution $z' \sim \mathcal{N}(\mu_{z'}, \sigma_{z'}^2)$. To decode plans into actions at a given state, Play-LMP utilizes a policy π that takes in a plan z, sampled from either the prior distribution (test time) or posterior distribution (train time), along with the current state s_t to predict the action a_t at that step.

E, E', and π are recurrent networks that operate over the fixed time horizon H and are learned end-to-end at training time. This architecture can be viewed as "encoding" trajectories in the environment into a lower dimensional plan z, and then reconstructing these plans into actions with the policy. PLATO similarly learns posterior, prior, and policy networks, but uses different inputs to each network based on sampling interactions and more meaningful goal environment states, as seen in Figure 1.

Training: E, E', and π are trained end-to-end by minimizing the following loss, equivalent to maximizing the evidence-based lower bound (ELBO) of the data likelihood under the posterior network E, given the learned prior network E':

$$\mathcal{L}_{LMP} = -\frac{1}{H}\sum_{t=0}^{H-1} \log(\pi(a_t|s_t, s_H^o, z)) + \beta \, \text{KL}(\mathcal{N}(\mu_z, \sigma_z^2) \parallel \mathcal{N}(\mu_{z'}, \sigma_{z'}^2)) \tag{2}$$

There may be a variety of ways to go from the initial state (s_0) to the final state (s_H^o); for example, in order to move a block to the right, we might grab-move the block or push block without grabbing. Play-LMP benefits from encoding these variations in the latent space. PLATO uses a similar variational loss as Play-LMP, but instead of reconstructing actions from the short, fixed horizon τ, reconstructs the interaction and pre-interaction trajectories $\tau^{(-}$ and $\tau^{(i)}$ conditioned on the affordance extracted from $\tau^{(i)}$ (see Eq. 1).

Test Time: At test time, the plan posterior network E cannot be used to output plans z, since it assumes access to the full trajectory. Therefore, the plan prior network E' is used to propose z' at test time, using only the initial (current) state s_t and the desired goal environment state (s_H^o). The policy then uses the resulting distribution over z' to predict actions online.

A.1 Play-LMP Strengths

Play-LMP is able to interpret the effects of short interactions with the environment. In doing so, it learns to propose plan distributions $p(z')$ for achieving a wide variety of goals using just the learned prior E'. By capturing all the possible variations of going between each pair of start states s_0 and goal environment states s_H^o during training, the Play-LMP policy is capable of executing a wide variety of behaviors at test time. As compared to Play-GCBC on sampled play windows, Play-LMP performs better on a wide range of manipulation tasks. Play-LMP and Play-GCBC are also

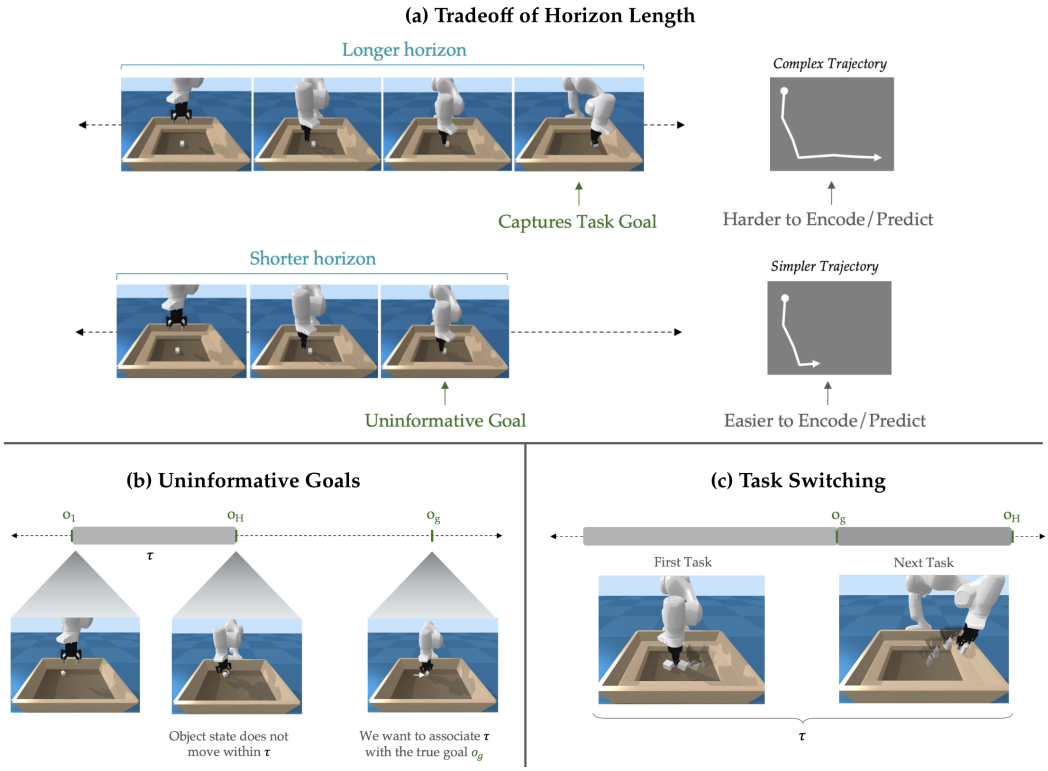

Figure 5: Challenges of short, fixed horizons, visualized for an example pushing task. (a) Choosing the horizon act is a balancing act between capturing task and goal information in the plan, and being able to predict plans (longer horizon means more complexity to capture) from just the initial and goal states. (b) If the horizon length is shorter than the current task, the goal environment state might be the same as the current state (if object state doesn't change within tau), and thus we will not properly link behaviors with the goals that induced these behaviors. (c) If the horizon is too long, We might capture multiple tasks within tau, and thus improperly assign the goal for the first task as being that of the next task. Our method, PLATO, addresses these challenges by biasing the latent space to learn from *object interactions* within play.

shown be robust to minor perturbations in the environment, which can be attributed to the broad state-action-goal distributions found in human play data.

A.2 Play-LMP Limitations

Several conceptual issues arise from the goal labeling strategy. Play-LMP labels the last state in the sampled play sequence as the goal for all earlier states (hindsight relabelling). The assumption embedded in this method is that the last state of any sequence of length H in play adequately represents the human's goal when choosing their actions (i.e., that the goal implies the actions). This can lead to the following issues relating to proper credit assignment:

Challenges of Fixed and Short Horizons: Inability to Capture Skills with Variable or Long Horizons: As discussed, choosing the horizon length is a balancing act between picking capturing longer horizon task goals and behaviors in the latent space (posterior) and ensuring the predictability of those skills from just the start and goal states at test time (prior). Due to this trade-off, Play-LMP can fail to capture the true goals when play consists of tasks whose horizon length is variable and/or large. In these settings, the longer horizon goal states will be out of distribution for the prior, and thus the policy can fail to produce the correct behavior. This trade-off in horizon lengths is visualized in Fig. 5 (a) for a pushing example.

Challenges of Short Horizons: Uninformative Goals: On the flip side, short sampled windows will often not contain any changes to the environment. One can imagine if the robot is re-orienting or servoing to an object, the goal state from the end of the short window will not be any different than the starting state, and thus uninformative for the prior network, even if the trajectory itself is

nontrivial – see Fig. 5 (b), where τ only captures the reaching phase of pushing. Thus Play-LMP might learn to map many sequences in play to the null prior, $E'(s, s^o)$, where s^o are the same environment state features in s.

Challenges of Fixed Horizons: Task Switching: Another issue related to sampling windows randomly from play is that we might sample a window that contains a logical boundary between two tasks. Thus, the labelled goal (last state in the window) belongs to the second task, and will not necessarily inform the behavior in the first task – see Fig. 5 (c), where τ contains both a pushing and lifting task back to back. Thus the prior will likely not be able to predict the correct plan distribution for the policy to use, and will incur a high KL penalty that might shape the latent space disproportionately.

Challenges of Random Trajectory Sampling: Even if we could perfectly label goals for each play trajectory τ, it might be the case that not all sub-sequences should be "equal" in terms of their contribution to the latent space structure. For example, a sequence involving a robot reaching an object does not contain as many critical states – states where the policy must be precise and accurate – as a sequence involving picking up a block or rotating it in-hand. With Play-LMP, both of these types of sequences would be weighted equally in the construction of the latent space. We posit that the latent space should attend more to critical states in play like object interactions; therefore, prioritizing encoding sequences from object interactions will enable a more useful and information rich latent plan space.

Fundamentally, the issues above relate to a failure of *credit assignment*, stemming from the short and fixed length of the sampled play sequences: the inferred goal does not actually represent the demonstrator's true goal. Our method PLATO considers tasks with variable horizon by leveraging object interactions with the environment. In the next section, we outline the implementation details for PLATO.

B PLATO Implementation Details

Next we will discuss the training procedure and low-level implementation details for PLATO, involving further algorithm details, network architectures, and hyperparameter choices.

B.1 Training Procedure

Algorithm 2 PLATO Training

1: Given: $H^{(i)}$, $H^{(-)}$, play data D_{play}, interaction criteria $f^{(i)}$,
2: $D_{\text{play}}^{(-)}, D_{\text{play}}^{(i)}, D_{\text{play}}^{(+)} = f^{(i)}(D_{\text{play}})$ ▷ Split into interactions
3: Initialize E, E', π
4: **while** not converged **do**
5: $\quad \tau^{(-)}, \tau^{(i)}, \tau^{(+)} \sim D_{\text{play}}^{(-)}, D_{\text{play}}^{(i)}, D_{\text{play}}^{(+)}$
6: $\quad$ Sample $o_g \sim \{o_t^{(+)}\}$
7: $\quad p(z) \leftarrow E(\tau^{(i)})$ ▷ Posterior Affordance Distribution
8: $\quad p(z') \leftarrow E'(o_1^{(i)}, o_g)$ ▷ Prior Affordance Distribution
9: $\quad z \sim p(z)$
10: $\quad \tilde{a}_{1:H^{(i)}}^{(i)} \leftarrow \pi(s_{1:H^{(i)}}^{(i)}, o_g, z)$ ▷ Policy in Interaction
11: $\quad \tilde{a}_{1:H^{(-)}}^{(-)} \leftarrow \pi(s_{1:H^{(-)}}^{(-)}, o_g, z)$ ▷ Policy in Pre-Interaction
12: $\quad$ Compute $\mathcal{L}_{\text{PLATO}}$ with Eq. (1) and update π, E, E'.

We now outline the training procedure for PLATO in greater detail, with Algorithm 1 reproduced above in Algorithm 2 for convenience. First, we sample segmented pre-interaction, interaction, and post-interaction periods from the play dataset. We then sample fixed length windows $\tau^{(i)} = (s_1, a_1, ..., s_{H^{(i)}})$ from the interaction period and $\tau^{(-)} = (s_1, a_1, ..., s_{H^{(-)}})$ from the pre-interaction period (Line 5 in Alg. 2). Note that the post-interaction period is just the next pre-interaction period, and thus is still sampled for the next interaction. In the pull example in Fig. 1, $\tau^{(i)}$ is the pulling motion, and $\tau^{(-)}$ is the reaching motion before pulling. We sample fixed horizon snapshots of

each period for computational efficiency; however, the duration between $\tau^{(-)}$ and $\tau^{(i)}$ can vary tremendously, and so we are still able to capture variable and long horizon chains of events despite only sampling fixed horizon windows within each period.

As described in Sec. 3 in the main text, the goal object state o_g for this chain of events can be selected as any object state after the interaction period's last object state $o^{(i)}_{H^{(i)}}$ and before the end of the post-interaction period (Line 6 in Alg. 2). During the interaction to post-interaction range, we know that the object trajectory is determined only by the interaction window actions and obstacles in the scene. For example, if we grab a block and then launch it along the table, the post-interaction period will consist of the block sliding; any state along that slide directly results from grabbing and launching. Thus the goal o_g can correctly be attributed to affordance z.

Having sampled $\tau^{(-)}$, $\tau^{(i)}$, and o_g, PLATO learns a goal-conditioned policy to reproduce the actions in both $\tau^{(-)}$ and $\tau^{(i)}$. Our insight is that much of the diversity in task-relevant behavior is contained during the interaction period, so instead of encoding $\tau^{(-)}$ and $\tau^{(i)}$ separately, we only encode $\tau^{(i)}$ with posterior E (Line 7 in Alg. 2). Now, the policy during interaction is conditioned on the $z \sim E(\tau^{(i)})$. Importantly, the policy during pre-interaction also conditions only on z, representing the *future* interaction (Line 10-11 in Alg. 2).

B.2 Architecture

We follow a similar implementation as Play-LMP for the posterior, prior, and policy networks, as described in [3]. The posterior, prior, and policy networks are implemented as a Bidirectional GRU-RNN, an MLP, and a Unidirectional GRU-RNN, respectively. Input trajectories to the posterior include both robot and object state information, but aligning with Play-LMP we leave out actions for the posterior input, as including actions empirically worsens performance. Actions are target positions and orientations of the robot, since these are flexible and intuitive enough for humans to operate. All action reconstruction losses use Mean Absolute Error (deterministic actions), since empirically we found little benefit to using probabilistic actions with a negative log-likelihood loss. Specific architecture choices for each environment and method are detailed in Table 1, as determined by extensive hyperparameter sweeps. For PLATO, we set $H^{(i)} = H^{(-)} = H$, and pre-interaction reconstruction loss weight $\alpha = 1$. Included in this hyperparameter sweep are minor horizon variations as employed in Relay Policy Learning [4], which we found not to benefit policy learning for our settings.

PLATO additionally uses a "soft-boundary length" parameter (S) to allow for some flexibility in what is considered the boundary of interaction and pre-interaction during sampling. If c_s and c_e are the true contact start and end indices, then $\tau^{(i)}$ is sampled between $c_s - S$ and $c_e + S$. Likewise, $\tau^{(-)}$ is sampled from between 0 and $c_s + S$. This creates overlap between the pre-interaction and interaction regions, which we found empirically is necessary such that the policy can be trained contiguously across the contact border. In practice, we set $S = H/2$, since this allows for full coverage of the contact border during sampling.

C Experimental Details

In this section we outline the environments, our real world setup, tasks, data collection, and evaluation process we employed. Each environment has substantial variability, and scripted policies are similarly designed to be quite diverse with sizeable injected noise.

C.1 Environments and Tasks

For all simulated environments, the contact signal used in simulation is the binary contact information between the robot and the rest of the scene, which can easily be computed in pybullet (3D) and pymunk (2D).

Block2D: The first environment is a 2D continuous block manipulation environment implemented with the PyMunk 2D physics engine [25]. Blocks in the environment are sized, massed, and positioned randomly at every reset. The ego agent (red) can interact with these blocks in the presence of gravity, with a special "tether" action that creates a link constraint if the ego agent is close enough to the block. This environment is meant as a 2D analog to more challenging block manipulations.

Environment	Method	β	H (seconds)	$\lvert z \rvert$	π-hidden	E-hidden	E'-width
Block2D	Play-GCBC	N/A	2	N/A	128	N/A	N/A
	Play-LMP	1e-3	2	16	64	128	128
	PLATO	1e-3	2	16	64	128	128
	Play-GCBC (H)	N/A	2	N/A	256	N/A	N/A
	Play-LMP (H)	1e-3	2	16	256	128	256
	PLATO (H)	1e-3	2	16	256	128	256
3D-Flat	Play-GCBC	N/A	4	N/A	128	N/A	N/A
	Play-LMP	1e-3	4	64	128	128	256
	PLATO	1e-4	4	64	128	128	256
3D-Platforms	Play-GCBC	N/A	4	N/A	128	N/A	N/A
	Play-LMP	1e-4	4	64	128	128	256
	PLATO	1e-4	3	64	128	128	256
Mug-3D	Play-GCBC	N/A	4	N/A	256	N/A	N/A
	Play-LMP	1e-4	4	64	256	256	256
	PLATO	1e-4	4	64	256	256	256
Playroom3D	Play-GCBC	N/A	4	N/A	256	N/A	N/A
	Play-LMP	1e-3	4	64	256	256	256
	PLATO	1e-4	4	64	256	256	256

Table 1: Best performing hyperparameters for each environment, method, and data source. β controls the regularization on the posterior from the learned prior. Each method is quite sensitive to this amount of regularization. H is the horizon length and controls the length of the sampled trajectory for the posterior encoder. All methods are quite sensitive to this as well. z is the latent vector, and $|z|$ is the latent dimensionality. π-hidden and E-hidden are the hidden sizes for π and E respectively, and these control the expressiveness of each network. E'-width is the width of the prior network. We do not vary the number of layers in each network, which are chosen to be the same as in prior work.

As shown in Figure 6, we constructed a set of block manipulation primitives to evaluate on: Push, Pull, Lift, Tip, and Side-Rotate.

Block3D-Flat: The second environment is a 3D block manipulation environment, involving a simulated Franka Emika Panda robot arm, 3D blocks, and a table playground area surrounded by walls. Similarly to Block2D, the block dimensions, initial positions, and masses here are randomly sampled. The Panda robot arm uses an operational space torque controller to exert realistic and bounded forces on the blocks. In this environment, we implement two dimensional Push primitives along the table surface, as well as a Top-Rotate primitive to control the z-axis rotation of the block in either direction (see Figure 6). Results for this environment, shown in Table 4, were not included in the main text due to space limitations.

Block3D-Platforms: The third environment builds on Block3D-Flat, but introduces platforms along the walls to enable even more complex primitives like lifting and placing. Here, we implement two dimensional Push primitives, and Lift-Place primitives in all cardinal directions on the table plane (see Figure 6).

Mug3D-Platforms: The fourth environment is like the Block3D-Platforms environment, but uses a challenging mug object of varied size and mass, instead of blocks. Here, we implement two dimensional Push primitives, and Lift-Place primitives in all cardinal directions on the table plane, and additionally a Mug Rotate primitive similar to Block3D-Flat (see Figure 6 for an example of Mug Rotate). These primitives each require unique types of affordances for a mug, and this environment is designed to test how learning from play methods can adapt to more precise manipulation tasks.

Playroom3D: The fifth environment is similar to the one used in prior work, involving several dynamic objects: a block on the table, a drawer, a cabinet door, and buttons in the cabinet. This environment is challenging since it involves learning affordances over multiple objects, as well as

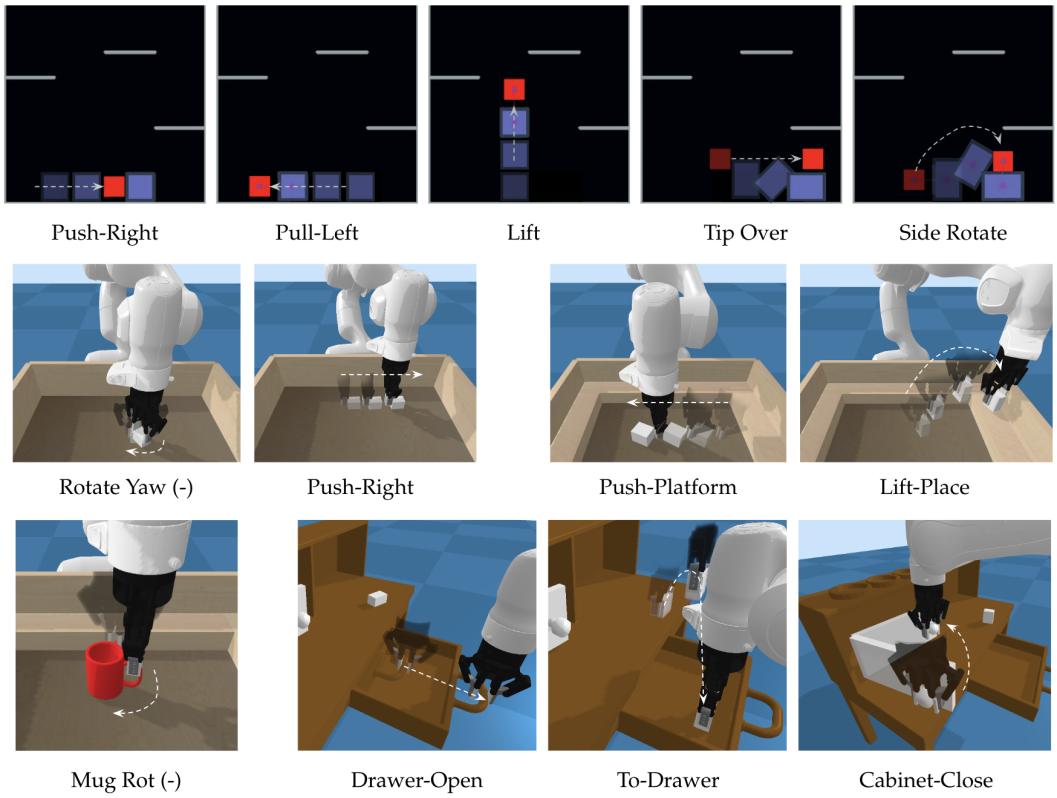

Figure 6: **First Row**: Block2D Environment Primitive Examples. Each of these primitives can be executed from a variety of object initial conditions, masses, and dimensions. **Second Row**: Block3D and Block3D-Platform Primitive Examples. Again, object initial conditions, masses, and dimensions are varied during play. The left two primitives shown are taken from **Block3D-Flat**, and the right two are taken from **Block3D-Platforms**. These represent a subset of the evaluation primitives in the 3D environment, and are meant to show the diversity of tasks and behaviors our method is evaluated on. **Third Row**: The left image shows an example primitive in **Mug3D-Platforms**. The right three images show sample tasks from **Playroom3D**.

interactions between them (like putting objects into the drawer or cabinet). For the cabinet, we test opening and closing actions, and likewise for the drawer. For the object, we test Pushing Left/Right primitives as well as moving the object To-drawer and To-cabinet (see Figure 6 for examples). These actions require many time-steps and have several bottleneck states. Play data also consists of moving the object From-drawer and From-cabinet, and button pressing (see Appendix D).

Real Robot Environment: Finally, we create a real robot environment that mirrors our simulation environment. Our setup is shown in Figure 7. We use three mounted Realsense SR300 cameras to robustly detect the pose of the green cube in the presence of occlusions from the robot, using OpenCV's Aruco tag detection for 3D pose estimation. As with our simulation experiments, we condition policies on full state of the environment (object 6D pose + end effector 6D pose + gripper state) rather than images. We add noise to the object state in simulation to match the noise seen in real world block state estimation. This along with an identical action space helps to reduce the sim-to-real gap and enables immediate deployment of simulation trained policies in the real world. Regardless, the dynamics of the object are still starkly different than those in our simulation dataset: the block is lightweight and slightly deformable, and has high friction with the table that can cause it to flip over instead of slide on the table. We test pushing primitives in this environment to demonstrate that PLATO is scalable to real world tasks with minimal data augmentation.

Real-World Deployment: To deploy this system more generically (including training on real world data), there are several key infrastructure additions beyond the setup we have shown here. In terms of hardware, the robot would need an additional contact sensing patch on the gripper or a force/torque sensor at the end effector to automatically detect interaction with the scene, as well as several cameras to observe the scene. Note that contact readings are only used during training (for the purpose

of interaction segmentation). Since our real world setup did not involve training on real world data, we did not need to add these additional sensors. However, we believe this modification should be quite straightforward. With pressure sensing (which is closest to what we do in simulation), there is a limitation that only interactions with the pressure sensing portion of the end effector will be counted during segmentation. In terms of software, a robust object 6D pose detection algorithm would be utilized to detect the object states under potential occlusion. With these additions, our method should readily scale to many real robot systems.

Example Task primitives for simulation evaluations are shown in Fig. 6. There is substantial within-task noise for scripted data to more closely resemble human data and real world conditions.

C.2 Task Complexity

We believe our set of tasks covers a wide spectrum of levels of complexity, from simpler pushing tasks to more complex door opening or object rotation tasks.

Firstly, we have incorporated diversity in primitives and variations in objects, which we emphasize is crucial and is not present in prior work such as Play-LMP. The tasks themselves cover a number of diverse primitives, where each "primitive" consists of variations in the goal, for example varying pushing distance or lifting placement location. The motions of the scripted primitives also have sizable variations in intermediate waypoints, speed, etc. Within all of our environments, we inject large variations in the positions, orientations, each size dimension, and masses of each of the blocks and mugs, which each require different strategies from the robot's perspective. Furthermore, grasping the mug involves a very precise interaction with the handle, which contrasts the wide, centered grasp used with blocks. This is in comparison to the closest prior work, i.e., Play-LMP tasks, which usually are projected to far fewer primitives and variations in the policy. The Play-LMP tasks largely involve either fixed object shapes with limited random pose initialization or static scene elements like buttons and constrained unchanging drawers. This might make it seem that these tasks are complex at the surface visual level, but we argue that our set of tasks and primitives require a much greater range of behaviors, such as grasping different shapes, rotating blocks to a wide spectrum of new orientations, and handling a variety of block masses, and thus these tasks are more complex. We would like to emphasize that visually interesting environments (e.g., added buttons or static objects as in Play-LMP), do not really add to the complexity of policy learning. What makes policy learning challenging is variations in behaviors and object properties, which we extensively test with our experiments. We believe that the performance of Play-LMP suffers in these settings precisely because the tasks are more complex for policy learning.

Secondly, our experiments include a visually interesting and complex environment, Playroom3D, which represents a more challenging version of the tasks used in Play-LMP involving some of the same underlying assets but having multiple randomized objects. To make the tasks even more challenging, we added the cabinet door opening and closing tasks. This "door opening" affordance was not included in the Play-LMP prior work, and is significantly harder to learn (see Play-LMP and Play-GCBC performance). We thus believe that our tasks are significantly more complex and diverse compared to prior work in this domain and adequately demonstrate the performance of our algorithm and a significant gap with prior work.

C.3 Evaluation Details

To evaluate each method, we evaluate across a set of environment-specific primitives by first running the primitive at the current state to generate a goal, then resetting and running the policy conditioned on that goal. We separately evaluate tasks by what we considered to be "similar" affordances. For example, push-left and push-right are considered as two separate tasks. However, there is sizeable variation in the set of goals that comprise each individual task. For example, with pushing, there is variation in the pushing distance, as well as all inherent variations in object properties in the environment (e.g., mass, shape, etc) discussed previously. Figure 8 provides some intuition that our affordance space learns to cluster by these directions. Success metrics are specified according to the task primitive being tested, usually involving the distance to either the goal object position or orientation. Evaluation times out after a fixed number of steps if the given method is unsuccessful. For stability, the latent vector z' is sampled once every $T \leq H$ steps and held constant during action decoding by the policy for the next T steps. For Play-LMP and PLATO, latent vectors are recomputed at the same frequency during evaluation, but the goal is fixed for the evaluation period.

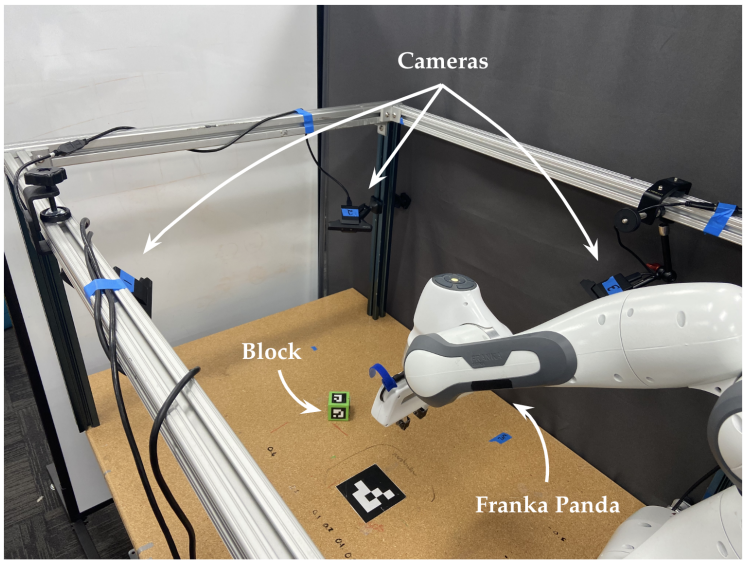

Figure 7: Block-Real environment. We use the Franka Emika Panda 7DOF robot arm for our experiments. The green cube we use for block manipulation tasks is shown in the middle, with ArUco tags on each face for pose detection. The 6D pose of the object is estimated via a multi-camera setup in order to be robust to occlusion by the robot, as shown here with the camera on the far right. The ArUco tag in the middle is used to calibrate the extrinsics of the cameras and localize the object frame of reference relative to the robot.

C.4 Data Collection

To evaluate our method across a wide variety of affordances available in the environment, we collect a large dataset of play data (roughly 15 hours) under artificial demonstrator agents that repetitively and randomly choose between the hand designed primitives for each environment. Proprioceptive state information includes robot end-effector 6D poses and twists, and object state information includes the object 6D poses, the 1D cabinet angle, and the 1D drawer open distance. In order to replicate human play as closely as possible, we add large amounts of variation to the primitives through randomized parameters. For example, with the pulling primitive, we vary the speed and duration of the pulling motion, as well as the reaching behavior. The boundaries between primitives are assumed to be unknown during training, like in human play data.

We also separately collect a smaller dataset (roughly 8 hours) of noisy human play data to evaluate the Block2D environments. From these experiments, we draw insights on the differences between human and scripted play data. Human play data for the Block2D task is collected using a keyboard control interface. Arrow keys control the ego agent position, while 'g' controls the grabbing tether action. One proficient user was used to collect the data, and was shown a set of tasks (the evaluation tasks) to perform during play with the instruction of trying to equally represent each task in their play, but they were also clearly informed that they were not limited to performing just these tasks. This user was given 15 minutes to practice in the environment, after which point data collection began. In future work, we hope to reduce the dependence on balanced, curated datasets and allow play to be truly freeform.

D Results and Analysis

In this section we discuss in greater depth our results, additional experiments, and tables with exact success rates to complement the bar-plot figures in the main text (Figures 2, 3, 4).

D.1 Detailed Results

Block2D: In Table 2, we see that PLATO substantially outperforms the baselines on each task. Play-LMP and Play-GCBC get roughly similar performance on most tasks, and struggle the most on tasks involving the tether action (Pulling and Side-Rotate). To further study the effects of encoding just the interaction period in the latent space, we implement PLATO-PRE, a version of PLATO

	Push-L	Push-R	Pull-L	Pull-R	Lift	Tip	Side-Rot
Play-GCBC	74.4(4.4)	84.5(2.9)	30.0(4.8)	18.6(3.4)	47.8(2.6)	72.8(3.8)	37.3(1.0)
Play-LMP	87.5(1.5)	89.9(1.0)	36.5(8.5)	17.9(4.0)	42.0(2.6)	81.0(2.0)	29.0(2.7)
PLATO-R	83.9(4.0)	86.6(6.12)	25.4(4.5)	38.1(0.5)	50.2(4.0)	79.4(2.6)	44.2(2.7)
PLATO-PRE	95.9(1.9)	95.7(1.0)	66(11.9)	67.1(7.4)	**80.9(7.1)**	84.3(1.6)	59(0.9)
PLATO	**99.1(0.5)**	**98.8(1.2)**	**69.0(3.8)**	**81.4(3.0)**	71.5(2.3)	**86.9(3.6)**	**73.8(1.0)**
Play-GCBC (H)	33.3(5.1)	52.4(3.0)	21.1(11.1)	49.3(3.4)	40.7(8.6)	52.0(4.7)	34.1(3.3)
Play-LMP (H)	54.8(4.7)	62.2(4.4)	27.9(7.4)	58.6(1.6)	29.2(7.2)	53.9(3.7)	29.2(1.2)
PLATO (H)	**81.3(3.0)**	**82.2(1.7)**	**65.3(2.7)**	**69.5(0.5)**	**65.6(2.8)**	**79.9(0.6)**	**54.0(1.7)**

Table 2: Block2D Success Rates in percentages in the form mean(std-err), trained over 3 random seeds and evaluated on various Push, Pull, Lift, Tip, and Side-Rotate primitives. Block sizes are randomized in each dimension, and blocks are initialized in random positions along the bottom of the grid. Our method PLATO outperforms all prior methods on both scripted and human data. PLATO-PRE includes pre-interaction information in the learned latent space (amounting to passing both $\tau^{(i)}$ *and* $\tau^{(-)}$ into the posterior E), but increases the training time compared to PLATO. PLATO-R incorporates the current robot state into the prior, representing the non object-centric version of PLATO. Object-centric methods that learn from interactions (PLATO, PLATO-PRE) perform much better than their counterparts.

	Push-L	Push-R	Pull-L	Pull-R	Lift	Tip	Side-Rot
PLATO	99.1(0.5)	98.8(1.2)	69.0(3.8)	81.4(3.0)	71.5(2.3)	86.9(3.6)	73.8(1.0)
PLATO-FC(4%)	100	95.9	61.3	57.0	78.4	93.0	68.3
PLATO-FC(8%)	97.0	96.7	35.0	65.6	80.4	95.6	48.7
PLATO-FC(12%)	94.8	100	61.5	10.1	60.8	92.3	63.6

Table 3: Contact Signal Ablation Experiment. Here we artificially add fake contact signals outside of interactions (e.g., during pre-interaction), causing false interactions to be segmented during training. We evaluate PLATO-FC(%), where % denotes the percentage of *interactions* that are false positives. We show results for a single seed of each PLATO for 4%, 8%, and 12%. Pulling tasks have some variance in performance under added false contact, however considering how much data is affected, PLATO is quite robust for all tasks.

in which the posterior encodes sampled trajectories from both the pre-interaction period $\tau^{(-)}$ and interaction period $\tau^{(i)}$, instead of just the interaction period. We see that PLATO and PLATO-PRE do roughly equivalently on average, with PLATO doing substantially better on Side-Rotate, but PLATO-PRE doing better on the lift primitive. Note that PLATO-PRE is slower to train since the posterior recurrent encoder receives a much longer sequence. We hypothesize that this is due to the tradeoff of various tasks between the complexity of the interaction and the complexity of pre-interaction behaviors. Overall, we can conclude that the including pre-interaction trajectories in the latent space (PLATO-PRE) is not uniformly better than only including interaction trajectories (PLATO), and thus does not warrant the added training time. This confirms our intuition that for complex interaction sequences like Side-Rotate, the interaction period contains enough information from the perspective of representation learning. Importantly, we see that framing play data through the lens of object interactions (PLATO, PLATO-PRE) results in much better policy learning than prior state-of-the-art methods.

Interestingly, performance for all methods is worse on the human generated data than scripted data. We attribute this to the fact that despite the significant noise added to the scripted primitives during data collection, scripted data still has cleaner and more successful demonstrations of each task than human data, which can contain many sub-optimal trajectories due to the challenges of teleoperation. For example, we observed that the Side-Rotate primitive is only successful in the human play dataset around 70% of the time. Additionally, humans can demonstrate the same task in many more ways than we could possibly script (e.g., by elongating the duration of a pulling motion or picking a wildly different spot to pull from), resulting in much more complex plans and policies to learn. This is supported by the fact that, as shown in Table 1, the best performing architectures for each method on human play data involved larger policy hidden sizes as compared to the best performing methods on scripted play data.

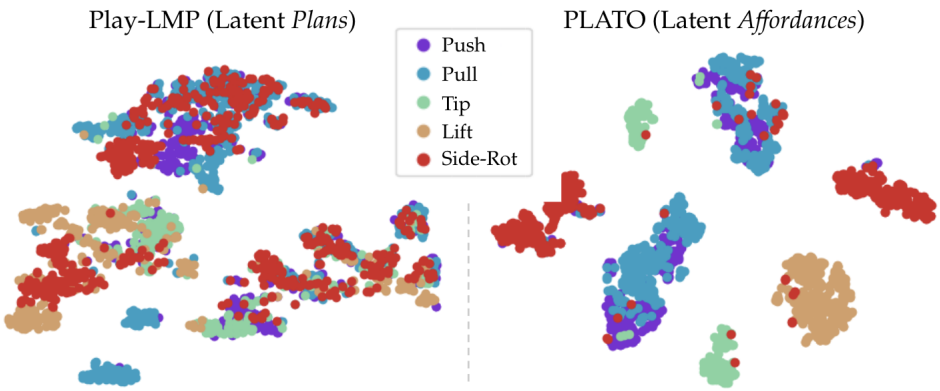

Figure 8: Learned latent spaces for Play-LMP (left) and our method PLATO (right) in the Block2D environment, plotted in 2 dimensions (from original 16 dimensions) with t-SNE. While there is overlap between tasks in the learned *plan* space in Play-LMP, tasks are much more separable with the learned *affordance* space in PLATO. We see separate clusters for each primitive likely relating to the direction of the primitive, and we also see within-cluster variation for each primitive. Note that push and pull have sizeable overlap since these represent similar motions (motion left and right on the ground). This clustering suggests that learning from interactions helps the posterior E to provide simpler and more informative latent representations for the policy.

For these 2D environments, we additionally examine the structure of the latent space learned by PLATO compared to prior work. Figure 8 shows example learned latent spaces for both Play-LMP and our method PLATO on Block2D; we see that by training on interactions, PLATO recovers a latent space that is more separable by task than Play-LMP despite not being presented with any task labels during training.

Additionally, in Table 3), we ablate the quality of the interaction signal (contact) in order to determine the important of clean contact signals. To accomplish this, we add in *false positive* contact signals to the data (e.g., during pre-interaction). This causes some percentage of the interactions sampled during training (denoted by FC(%) in Table 3) to actually be non-contact sequences. We see that PLATO is quite robust to added false positive contacts. The Pull task has high variance in performance here under added contacts, however there is no clear relationship between more contact noise and worse performance for any of the tasks. This is likely due to the fact that all models condition on the start and goal states, and false contact signals usually involve no change in the object state; therefore, from the model's perspective, sampled false interactions will be completely distinguishable from the task affordances that we care about at test time.

	Push-F	**Push-L**	**Push-B**	**Push-R**	**Rotate (+)**	**Rotate (-)**
GCBC	71.3(3.0)	65.7(10.0)	57.5(11.2)	50.1(7.8)	53.7(2.6)	42.4(1.9)
LMP	68.0(4.5)	63.0(4.4)	50.6(2.3)	28.0(9.0)	39.6(3.2)	40.7(3.4)
PLATO	**77.4(4.3)**	**89.4(5.5)**	**74.0(4.7)**	**84.0(5.0)**	**83.3(4.4)**	**78.6(6.7)**

Table 4: Block3D-Flat Success Rates in percentages in the form mean(std-err), trained over 3 random seeds and evaluated on pushing and rotation tasks. Again, our method PLATO outperforms all prior methods, especially on the harder rotation primitives. Interestingly, we also see that Play-LMP does not do as well as Play-GCBC on several primitives, likely due to the prior and policy being out of distribution at test time.

Block3D-Flat: Results for this block pushing and rotating environment (not presented in the main text due to space limitations) are shown in Table 4. Here again, PLATO is able to significantly outperform the baselines on all of the tasks, especially in the more fine-grained rotation motions, as well as on the pushing tasks.

Block3D-Platforms: Table 5 shows the results for Block3D-Platforms. PLATO outperforms the baseline methods in the pushing tasks, but the difference is substantially greater for the lifting tasks. These lifting tasks have longer, variable horizons and involve several bottleneck states (e.g., securely

	Push-F	Push-L	Push-B	Push-R	Lift-F	Lift-L	Lift-B	Lift-R
GCBC	95.5(1.5)	69.7(10.4)	81.8(6.7)	83.4(5.6)	18.2(10)	40.8(11.4)	31.9(2.3)	24.6(9.8)
LMP	84.7(7.3)	85.3(3.4)	90(2.9)	91.2(2.6)	47.2(5.6)	55.1(4.6)	48.6(10.9)	51.8(13.6)
PLATO	**98.3(0.4)**	**97.2(1.4)**	**97.8(2.2)**	**99.7(0.3)**	**75.6(1.9)**	**94.0(2.0)**	**81.0(1.1)**	**92.8(1.7)**

Table 5: Block3D-Platform Success Rates in percentages in the form mean(std-err), trained over 3 random seeds and evaluated on Lift-Place and Push primitives. PLATO is the only method able to do well on all evaluation primitives and do so consistently, for a wide variety of object dimensions and initial conditions.

grasping, clearing the platform, dropping on the platform). PLATO is the only method capable of performing well on both the longer horizon tasks (lifting) and shorter horizon tasks (pushing).

	Push-F	Push-L	Push-B	Push-R	Rot(+)	Rot(-)	Lift-F	Lift-L	Lift-B	Lift-R
GCBC	83.7(1.3)	91.3(1.5)	86.3(5.4)	93.0(2.5)	78.7(5.8)	81.0(0.6)	11.1(6.4)	27.7(2.7)	4.4(2.5)	20.6(1.3)
LMP	85.0(2.6)	87.6(3.8)	89.9(0.6)	90.6(3.1)	51.3(5.2)	63.3(3.2)	59.3(5.8)	74.3(4.7)	51.7(6.6)	64.8(5.3)
PLATO	**94.7(0.9)**	**92.0(2.1)**	**97.2(1.0)**	**98.7(0.7)**	**81.3(2.5)**	**86.3(6.1)**	**81.7(0.3)**	**84.3(3.0)**	**76.7(10)**	**85.2(3.6)**
LMP(S)	100	100	92.7	100	84.8	85.5	67.6	70.9	70.1	87.7
PLATO(S)	100	100	97.4	100	100	100	92.3	90.2	82.4	85.6
LMP(S+)	66.8	76.0	75.8	64.6	62.5	87.1	47.3	44.8	35.3	34.9
PLATO(S+)	88.3	87.1	77.4	90.9	87.4	86.7	64.4	71.0	52.4	72.3

Table 6: Mug3D-Platform Success Rates in percentages in the form mean(std-err), trained over 3 random seeds and evaluated on Lift-Place, Rotate, Push primitives. In the first block of the table, we see that PLATO is the only method able to do well on all evaluation primitives and do so consistently, for a wide variety of object dimensions and initial conditions. In the second block of this table (generalization experiments, one seed), (S) denotes that the method was trained on a subset of initial mug orientations (simpler tasks). While in principle play will reach other mug orientations, this still skews the distribution of mug orientations towards the initial predefined set. (S+) denotes methods trained on these subset of initial orientations (same models as S) but evaluated on the full range of mug orientations as used in the first block. We see that while LMP performs close to PLATO within distribution (subset of initial mug orientations, significantly less task diversity), PLATO is much more robust than LMP when presented with the full initial object state distribution at test time, showing the generalization capacity of PLATO.

Mug3D-Platforms: Table 6 contains both the results for Mug3D-Platforms presented in the main text, as well as an additional ablation experiment testing the generalization capacity of PLATO in this environment. The main experiments (first block of Table 6) show that similar to in Block3D-Platforms, PLATO outperforms the methods on all tasks, where the gap is less stark for pushing tasks but especially large on the lifting tasks. For the rotation tasks, interestingly Play-GCBC outperforms Play-LMP. We speculate that in cases like this, the Play-LMP policy might be too dependent on the plan z, and thus suffers at test time when the prior outputs an *approximate* z distribution given only partial information. In contrast, Play-GCBC is substantially worse than Play-LMP for the lifting task. This suggests that task variability is much larger for lifting than rotating and pushing, and hence the latent plan in Play-LMP helps the policy disambiguate between this variability. Overall, PLATO performs better and more consistently on all the tasks than either Play-GCBC or Play-LMP.

The second set of experiments (second block in Table 6) illustrate the robustness of PLATO to state/action/goal distribution shift. We train Play-LMP and PLATO on a subset of initial mug orientations in the mug environment. Specifically, the initial randomized z-axis orientation (yaw) of the mug at the start of each episode will be just a 90 degree cut of the full 360 degree range. While in principle sequential play will be able to eventually see tasks demonstrated for orientations outside this range, this initialization greatly skews the distribution of mug orientations for all demonstrated tasks towards the starting set of orientations. The first two rows of the second block in Table 6 (S) show the performance of Play-LMP and PLATO when evaluated on the tasks used for training (90 degree cut for initial mug orientations). We see here that due to the lower variability in tasks, Play-LMP performs quite well within distribution, notably on the lifting tasks (in contrast to the results from the first block in the table). PLATO performs better than Play-LMP across all tasks, consistent with the results in the first block, although the gap is reduced due to limited task variability. The second two rows of the second block in Table 6 (S+) show the performance when the same models from the first two rows (S) are evaluated on the full swath of initial mug orientations.

	Push-L	Push-R	Cab-O	Cab-C	Dr-O	Dr-C	To-Cab	To-Dr	Fr-Cab	Fr-Dr	Btn1	Btn2
GCBC	29.1(3.5)	52.9(4.0)	47.7(21.4)	1.2(1.2)	86.7(6.4)	22.5(11)	52.2(12)	60.6(10)	6.3(4.1)	3.7(0.3)	68.7(11)	66.7(18)
LMP	25.5(2.5)	60.3(10.5)	61.2(8.3)	0(0)	65.3(13)	27.2(13)	34.7(6.2)	45.5(7.1)	1.0(1.0)	19.7(6.5)	63.2(9.2)	42.3(13)
PLATO	**76.3(15)**	**66.7(15)**	**100(0)**	**78.7(6/9)**	**100(0)**	**100(0)**	**77.3(3.7)**	**60.4(2.0)**	**11.7(1.8)**	**58.3(3.5)**	**70.9(2.1)**	**100(0)**

Table 7: Playroom3D Success Rates in percentages in the form mean(std-err), trained over 3 random seeds and evaluated on Push, Cabinet, Drawer, To/From-Cabinet/Drawer, and Button Pressing primitives. PLATO is the only method able scale to all evaluation primitives, for a wide variety of object sizes and initial conditions.

As explained previously, both Play-LMP and PLATO have seen a limited set of tasks with these orientations due to the sequential nature of play, but only PLATO is able to retain good performance across all tasks. This demonstrates that by biasing the latent space towards learning object affordances, PLATO is better able to capture the full distribution of demonstrated behaviors, even those infrequently demonstrated, and thus is more adept under distribution shift in the test time tasks.

Playroom3D: In Table 7, we show the results on Playroom3D. Surprisingly, pushing tasks are much harder in this environment, which we speculate is due to the presence of multiple objects and thus a higher variability in *how* objects can be interacted with. PLATO is able to retain good performance on the pushing tasks, in contrast to the baselines. For opening and closing the cabinet and drawer, we see that PLATO is able to do quite well, but Play-LMP and Play-GCBC perform quite poorly. There is quite a diversity in horizon lengths across these different tasks and different instantiations of each task, which we believe contributes to this large gap. For the Cabinet closing task, there are several critical bottleneck states, for example being able to servo the arm above and to the other side of the cabinet door to reach the cabinet handle, as well as precisely grasping the handle. PLATO is the only method that learns to consistently perform this task. Likewise for the drawer tasks, PLATO gets 100% success for all random seeds, while performance is substantially worse on the baselines. We also evaluated two additional challenging tasks in the Playroom3D environment that were demonstrated during play less frequently: From-Cabinet and From-Drawer (pull object out of open cabinet, lift it out of open drawer), also with substantial object and primitive variation. The performance on these tasks are lower compared to the other tasks as they require many preconditions and thus are not equally represented during our collected play data. From-Cabinet also requires a novel end effector orientation and grasping procedure in order to avoid collision with the cabinet and table walls. We speculate that due to the novel motion, limited examples, and the presence of other tasks in the data, all methods do notably worse on the From-Cabinet task, however From-Drawer performance for Play-LMP and PLATO is closer to To-Drawer performance since these tasks involve similar object lifting behaviors. However, even with fewer examples in a crowded dataset of other tasks, there is still a large gap between PLATO and the next best method. Additionally, we evaluate on two button pressing tasks, where the buttons are in the cabinet space and thus reaching involves avoiding the cabinet door. Here, the period of contact is relatively short, but PLATO is able to reliably achieve higher success in these tasks as well. Overall, this environment demonstrates that PLATO gracefully scales to more complex environments with more diversity in object affordances and robot behaviors.

BlockReal: Results for our real world experiments are shown in Table 8. Both Play-LMP and PLATO are trained entirely in simulation. We add minor data augmentation in simulation in the form of gaussian noise for the object state estimates, to match the observed noise using our real world multi-camera object state estimation infrastructure. Note that both models get 90%+ success in simulation on each task, since these tasks have relatively low behavior and object diversity. We see that when deployed on the real world setup with no additional data, PLATO experiences only a minor performance degradation, suggesting that learning a latent *affordance* space is more robust than learning a latent *plan* space. These results are consistent with what we see in the Mug3D-Platforms generalization experiments in Table 6, however here we are testing generalization to entirely unseen real world physics, in contrast to task distribution shift in those experiments.

D.2 Additional Analysis

Variation in Performance for Similar Tasks: For several environments, semantically similar tasks like push-left and push-forward seem to have notably different success rates across many methods. Interestingly, those differences are mainly across different "axes" of the task, for example push-left and push-right usually have similar performance, and push-forward and push-backward also have

	Push-Left	Push-Back	Push-Right	Push-Forward
LMP	8/10	6/10	8/10	7/10
PLATO	8/10	**8/10**	**9/10**	**10/10**

Table 8: Results for BlockReal on pushing tasks. These models are trained entirely in simulation with minor data augmentation. We evaluate these models on a real robot setup, and see that performance degrades less for PLATO than for Play-LMP when presented with the real world object and robot dynamics. Methods that use play data are robust to environment changes, consistent with results from prior work [3].

similar performance, but these two sets have a gap in performance. We speculate that this is because of biases present in the data or the model that favor one axis over another, for example the relative presence of each task, or environmental difficulties in performing that task.

Quality of Interaction Segmentation: In all our experiments, we are not assuming access to perfect interaction segmentation. In fact, all of our environments will sometimes have imperfect interaction signals due to the demonstrator having notable noise (even in scripted policies, which have added noise) – for example, brushing against the table, object, or cabinet door on the way to perform a different task. Intermittent contact is also quite common in all of our play data. However, the smoothing on top of the interaction signal tends to clean many of these signals. See the Contact Ablation Experiments in Appendix D.1 for more analysis.

Furthermore, we pose this question: what defines "perfect" segmentation? In the framing of our method, any interaction with the environment, even accidental ones, are still valid for the affordance space to learn. If we accidentally brush the top of the cabinet door on our way to push an object, and the door opens slightly, this can be seen as a successful cabinet slight-open task. Since the start and goal object state are unique for this task, in theory it should not at all affect the affordance learning for a different start and goal object state. However, accidental interactions will start to affect learning if these interactions are common and bias the policy towards unsafe regions of the state (for example, if repeatedly brushing the top of the cabinet door on the way to push the block biases the policy away from pushing the block properly).

E Additional Limitations

We will now present a longer discussion of limitations presented in Sec. 5, as well as some additional limitations, to help guide future work.

Multi-Object Scenarios: As noted in Sec. 5, multi-object interaction scenarios like tool-use represent a key challenge for future work. While we show PLATO operating in multi-object environments in this work, we do not extend PLATO to multi-object *interactions* involving dynamic objects, for example hitting a puck with a hockey stick. In these settings, it might be difficult to obtain signals for interaction between the dynamic objects. We will give a couple of ideas of how future work might tackle this problem in the hopes of opening up a broader discussion. Before these ideas, one general point of clarification: PLATO introduces detecting "interaction" as a more general concept, which applies even when no contact occurs between the robot and the desired object (e.g., tool use). We used contact as our interaction signal since it was the most readily available for single-object interactions. However for multi-object interactions, the notion of interaction still exists and our method still applies if we can detect these interactions somehow. Since PLATO only requires detecting interaction during training, one option is to have a human label interaction segments in their training data manually. If this is too time intensive, we can potentially leverage *learned* binary signals for interaction using limited supervised interaction labels. Consider the tool-use task of hitting a hockey puck with a hockey stick. Even though we cannot directly observe a contact signal between the stick and the puck, future work could learn to predict the interaction signal from visual (e.g. observing the stick hit the puck) and maybe even haptic information (force feedback of the hockey stick on the end-effector when hitting), using supervised labels. In addition, recent approaches like ComPILE [26] have learned to segment skills without any labels, and could be used to isolate an interaction signal for dynamic multi-object interactions in a self-supervised fashion. While designing such a system for detecting interaction might require some effort, we believe that PLATO's results suggest that such effort can result in serious performance gains for policy learning from play.

With this general notion of interaction in mind, we would like to present two potential ideas for future work to expand PLATO to multi-object scenarios:

1. One idea would be to learn an *embodiment-specific* affordance space. Then, each tool can be viewed as a different embodiment of the robot, and with knowledge of what tool is currently being used, we can learn affordances specific to that tool. At test time if we know what tool we picked up, the policy can leverage the affordance space of this particular tool in a manner similar to PLATO.
2. As another related idea, we could attempt to learn multiple *degrees* of interaction: e.g. similar to how PLATO learns the relationship between an object affordance (hockey stick moves) and robot skill (grasp and swing a stick), we might also learn the relationship between a second order object affordance (hockey puck moves) and a first order object affordance (hockey stick strikes). Then at test time the policy could reason backwards in time, first inferring the correct second order affordance (hockey puck moves), then the first order affordance (hockey stick swings), then the robot action to take (grasp and swing the stick).

We see our work as a first step towards exploring some of these interesting paradigms for interaction and multi-level skill reasoning.

Regularization Weight: As one might expect, both Play-LMP and our algorithm PLATO share many of the challenges of variational auto-encoders [27]. Recent replications of this work have shown that the regularization weight β has a sizeable effect on the final policy reconstruction error [28]. High values of β can yield posterior collapse of the latent space, where the plan posterior outputs distributions for differing trajectories that do not reflect these differences; low values of β can yield low reconstruction losses, but conversely cause the posterior plans to encode information that is hard to predict from just the start and end states. As a result, the learned prior may not match plans encoded by the posterior, thus hurting the policy. Future work might look into more expressive learned priors, such as mixture models, in order to better match the posterior and thereby reduce the sensitivity to β. Another direction could be finding alternate ways to specify tasks at test time, for example giving some notion of *how* a goal should be reached.

Human Sub-Optimality: Additionally, when collecting play data, certain challenging primitives attempted by humans might fail often. We noticed that humans often fail at Side-Rotate in Block2D, for example, and the resulting demonstration might look like a sub-optimal Push from the perspective of the posterior and the prior. This introduces even more plan variability into the latent space for the Push task, and thus hurts test time performance. Another interesting direction would be to better understand how sub-optimality affects the learned latent space and potentially develop a notion of trajectory "quality" to bias this latent space.

Use of Object State Estimation: As mentioned in Sec. 5, PLATO makes use of object state estimates when learning object-centric affordances. A natural question is how this work can be adapted to work with pure image inputs. Critically, the only obstacle to extending our method to image inputs is our learned prior network, which utilizes just ground-truth object state information rather than the full proprioceptive state and object state. Note that our PLATO-R ablation can be extended to learn from images without any additional modifications, however this ablated method lacks the benefits of object-focused affordance learning (see Section 4). While we did not explore learning from images, we believe our method can readily scale to images with a few modifications. One method would be to mask out only the object(s) of interest from the start and goal images before passing them into the prior to encourage a robot agnostic latent space. Another method would be to learn an object representation directly from images that is independent of the robot state (e.g. with contrastive learning on negative examples of different robot poses, but identical environment states).

Regardless, we showed that learning object centric representations actually improves the robustness of policies at test time, and thus justifies the extra effort of designing these representations. Furthermore, there is a sizable research field devoted to improving object state estimation using learning and filtering techniques (even involving state estimation under clutter), so we believe that object state estimation methods will get even more practical in the coming years.