Appendix

A Table of Contents

B Ethics, Impacts and Limitations

B.1 Ethical Considerations

There exist potential ethical concerns from the use of large-scale language models trained on internet-scale data. These models have been trained on vast corpi that may contain harmful content and implicit or even explicit biases expressed by internet users and may be capable of generating such content when queried. However, these issues are not specific to our work, rather they are inherent to LLMs trained at scale and other works that use LLMs face a similar ethical concern. Furthermore, we note that our research only makes use of LLMs to guide the behavior of a robot at a coarse level - specifying where a robot should go and how to leave the area. Our LLM prompting scheme ensures that this is all that is outputted from the LLM. Such outputs leave little scope for abuse, the LLM is not capable of performing the low-level control itself, which is learned through a task reward independently.

B.2 Broader Impacts

Our research on guiding RL agents to solve long-horizon tasks using LLMs has potential for both positive and negative impacts. PSL draws connections between work on language modeling, motion planning and reinforcement learning for low-level control, which could lead to advancements in learning for robotics. PSL reduces the engineering burden on the human, instead of manually specifying/pre-training a library of behaviors, only a reward function and task description need be specified. More broadly, enabling robots to autonomously solve challenging robotics tasks increase the likelihood of robots one day being able to complete labor intensive work in dangerous situations. However, with increased automation, there are risks of potential job loss. Furthermore, with increased robot capabilities, there is a risk of misuse by bad actors, for which appropriate safeguards should be designed.

B.3 Limitations

There are several limitations of PSL which leave scope for future work. 1) We impose a specific structure on the language plans and task solution (go to location X, interact there, so on). While this assumption covers a broad set of tasks as well illustrate in our experimental evaluation, tasks that involve interacting with multiple objects simultaneously or continuous switching between interaction and movement in a fluid manner may not be directly applicable. Future work can explore integrating a more expressive plan structure with the Sequencing Module. 2) Use of motion-planning makes application to dynamic tasks challenging. To that end, research on motion-planner distillation, such as Motion Policy Networks Fishman et al. (2022) could enable much faster, reactive behavior. 3) Although the RL agent is capable of adapting pose estimation errors, in the current formulation, there is not much the Learning Module can do if the high-level plan itself is entirely incorrect, or if the Sequencing module misinterprets the language instruction and moves the robot to the wrong object. One extension to address this limitation would be to fine-tune the Plan and Seq Modules online using RL as well, to adapt the large models to the specific environment and reward function.

C Additional Experiments

We perform additional analyses of PSL in this section.

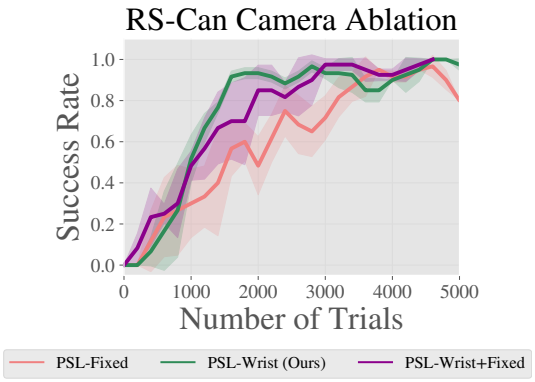


Figure C.1: **Camera View Learning Performance Ablation.** wrist camera views clearly accelerate learning performance, converging to near 100% performance **4x** faster than using fixed-view and **3x** faster than using wrist+fixed-view observations.

Effect of camera view on policy learning performance: As discussed in Sec. 3, for PSL we use local observations to improve learning performance and generalization to new poses. We validate this claim on the Robosuite Can task, in which we hypothesize that the local wrist camera view will accelerate policy learning performance. This is because the image of the can will be independent of the can's position in general since the Sequencing Module will initialize the RL agent as close to the can as possible. As observed in Fig. C.1, this is indeed the case - PSL learns **4x** faster than using a fixed view camera in terms of the number of trials. We additionally test if combining wrist and fixed view inputs (a common paradigm in robot learning) can alleviate the issue, however PSL with wrist cam is still **3x** faster at solving the task.

Effect of camera view on chaining pre-trained policies: In this ablation, we illustrate another important effect of using local views, such as wrist cameras: ease of chaining pre-trained policies. Since we leverage motion planning to sequence between policy executions, chaining pre-trained policies is relatively straightforward: simply execute the Sequencing Module to reach the first region of interest, execute the first pre-trained policy till its stage termination condition is triggered, then call the Sequencing Module on the next region, and so on. However, to do so, it is also crucial that the observations do not change significantly, so that the inputs to the pre-trained policies are not out of distribution (OOD). If we use a fixed, global view of the scene, the overall scene will change as multiple policies are executed, resulting in future policy executions failing due to OOD inputs. In Table C.1, we observe this exact phenomenon, in which any version of PSL that is provided a fixed-view input fails to chain pre-trained policies effectively, while PSL with local (wrist) views only is able to chain pre-trained policies on every task, up to 5 stages.

Effect of incorrect plans on training policies using PSL: As noted in the Limitations Section (Sec. B), we acknowledge that as defined, if the Plan Module or Sequence Module fail catastrophically (incorrect plan or moving to the wrong region in space), there is currently no concrete mechanism for the Learning Module to adapt. However, we run an experiment in which we train the agent using PSL using an incorrect high-level plan on two stage tasks (MW-Assembly, MW-Bin-Picking, MW-Hammer) and find that in some cases, the agent can still learn to solve the task, achieving performance close to E2E (Fig. C.4. Intuitively, this is possible because in PSL, the high-level plan is not expressed as a hard constraint, but rather as a series of regions for the agent to visit and a set of exit conditions for those regions. In the end, however, only the task reward is used to train the RL policy so if the plan is wrong, the Learn Module must learn to solve the entire task end-to-end from sub-optimal initial states.

Ablating Camera View choice for Baselines: We evaluate if including O^{local} in addition to O^{global} improves performance across four tasks (`RS-Lift`, `RS-Door`, `RS-Can`, `RS-NutRound`) and include the results in Fig. C.5. In general there is little to no performance improvement for RAPS or

E2E across the board. The additional local view marginally improves sample efficiency but it does not resolve the fundamental exploration problem for these tasks.

	K-Single-Task	K-MS-3	K-MS-4	K-MS-5
PSL-Wrist	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
PSL-Fixed	1.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
PSL-Wrist+Fixed	1.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table C.1: **Chaining Pre-trained Policies Ablation.** PSL can leverage local views (wrist cameras) to chain together multiple pre-trained policies via motion-planning using the Sequencing Module. While PSL with each camera input is able to produce a capable single-task policy, chaining only works with wrist camera observations as the observations are kept in-distribution.

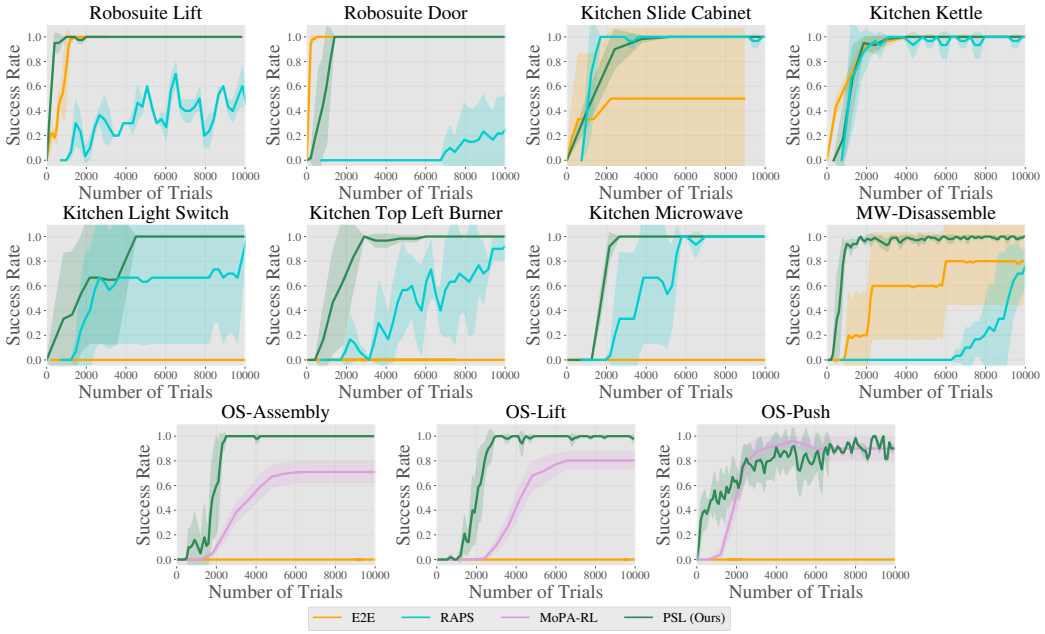


Figure C.2: **Single Stage Results.** We plot task success rate as a function of the number of trials. PSL improves on the efficiency of the baselines across single-stage tasks (*plan length of 1*) in Robosuite, Kitchen, Meta-World, and Obstructed Suite, **achieving an asymptotic success rate of 100% on all 11 tasks**.

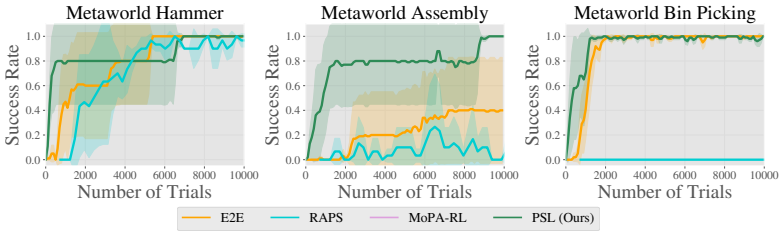


Figure C.3: **Meta-World Two Stage Learning Curves.** We plot task success rate as a function of the number of trials. PSL learns faster than the baselines by employing high-level planning to accelerate RL performance.

	MW-BinPick	MW-Assembly	MW-Hammer
E2E	1.0 ± 0.0	0.4 ± 0.5	0.0 ± 1.0
RAPS	0.0 ± 0.0	$0.3 \pm .25$	1.0 ± 0.0
TAMP	1.0 ± 0.0	1.0 ± 0.0	0.0 ± 0.0
SayCan	1.0 ± 0.0	$0.5 \pm .08$	1.0 ± 0.0
PSL	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table C.2: **Metaworld Two Stage Results.** While the baselines perform well on most of the tasks, only PSL is able to consistently solve every task. This is because the LLM planning and Sequencing modules ease the learning burden for the RL policy, enabling it to learn contact-rich, long-horizon behaviors.

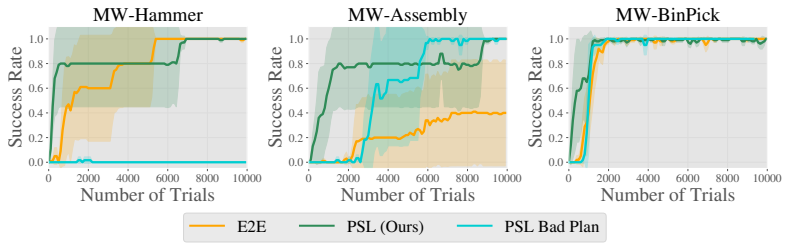


Figure C.4: **PSL Bad Plans.** We plot task success rate as a function of the number of trials. Even when given the wrong high-level plan, PSL is able to learn to solve the task, albeit at a slower rate.

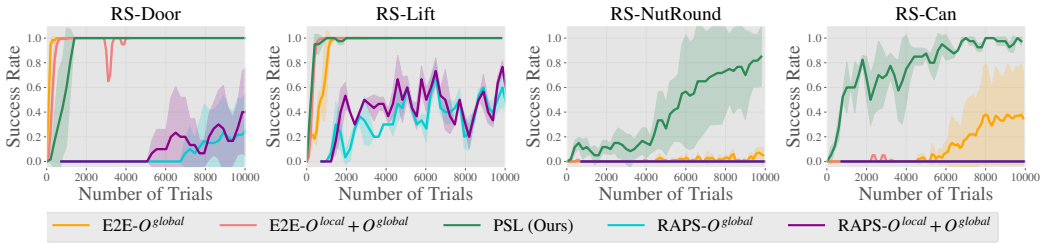


Figure C.5: **Ablating Camera Views for Baselines.** We plot task success rate as a function of the number of trials. As seen in the figure above, including O^{local} does not improve the performance of E2E or RAPS.

D PSL IMPLEMENTATION DETAILS

Algorithm 2 PSL Implementation

Require: LLM, task description g_l, Motion Planner MP, low-level horizon H_l, segmentation model $\mathcal{S}$, RGB-D global cameras, RGB wrist camera, Camera Matrix K^{global}

initialize RL: π_θ, replay buffer $\mathcal{R}$
Planning Module
High-level plan $\mathcal{P} \leftarrow$ Prompt(LLM, g_l)
for episode $1...N$ **do**
  for $p \in \mathcal{P}$ **do**
Sequencing Module
    target region (t), termination condition $\leftarrow p$
    PC^{global} = Projection(O_1^{global}, O_2^{global}, K^{global})
    M_{robot}, M_{obj} = Segmentation(O_1^{global}, O_2^{global}, robot, object)
    PC^{robot} , PC^{object} = $M_{robot} * PC^{global}$, $M_{obj} * PC^{scene}$
    $PC^{scene} = PC^{global} - PC^{robot}$
    ee_{target} = mean(PC^{obj})
    q_{target} = IK(ee_{target})
    MotionPlan(MP, q_{target}, PC^{scene})
Learning Module
    for $i = 1, ..., h$ low-level steps **do**
      Get action $a_t \sim \pi_\theta(O_t^{local})$
      Get next state $O_{t+1}^{local} \sim p(|s_t, a_t)$.
      Store $(O_t^{local}, a_t, O_{t+1}^{local}, r)$ into $\mathcal{R}$
      Sample $(O_k^{local}, a_t, O_{k+1}^{local}, r) \sim \mathcal{R}$ ▷ k = random index
      update π_θ using RL
      if post-condition **then**
        break
      end if
    end for
  end for
end for

D.1 PLANNING MODULE

Given a task description g_l, we prompt an LLM using the format described in Sec. 3.3 to produce a language plan. We experimented with a variety of publicly available and closed-source LLMs including LLAMA Touvron et al. (2023a), LLAMA-2 Touvron et al. (2023b), GPT-3 Brown et al. (2020), Chat-GPT, and GPT-4 OpenAI (2023). In initial experiments, we found that GPT-based models performed best, and GPT-4 in particularly most closely adhered to the prompt and produced the most accurate plans. As a result, in our experiments, we use GPT-4 as the LLM planner for all tasks. We sample from the model with temperature 0 for determinism. Sometimes, the LLM hallucinates non-existent stage termination conditions or objects. As a result, we add a pre-processing step in which we delete components of the plan that contain such hallucinations.

D.2 SEQUENCING MODULE

The input to the Sequencing Module is O^{global}. In our experiments, we use two camera views, O_1^{global} and O_2^{global}, which are RGB-D calibrated camera views of the scene, to obtain unoccluded views of the scene. We additionally provide the current robot configuration, which is joint angles for robot arms: q_{joint} and the target region label around which the RL policy must perform environment interaction. From this information, the module must solve for a collision free path to a region near the target. This problem can be addressed by classical motion planning. We take advantage of sampling-based motion planning due to its minimal setup requirements (only collision-checking) and favorable performance on planning. In order to run the motion planner, we require a collision checker, which we implement using point-clouds. To compute the target object position, we use predicted segmentation along with calibrated depth, as opposed to a dedicated pose estimation

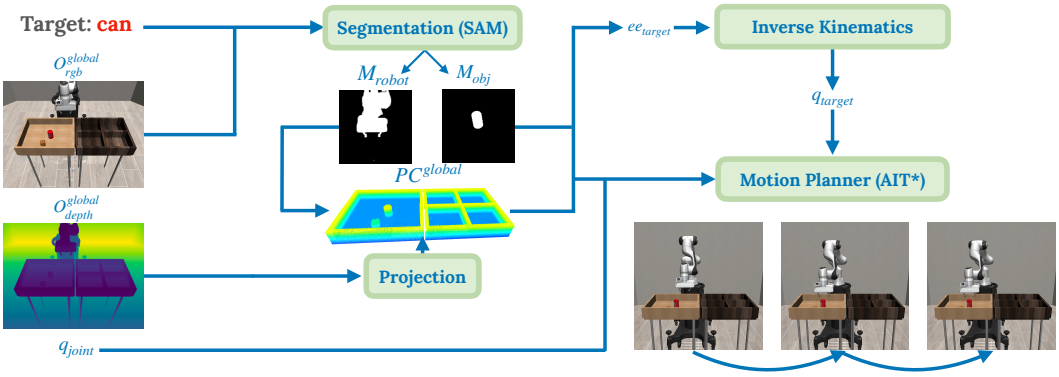


Figure D.1: **Sequencing Module.** Inputs to the Sequencing Module are two calibrated RGB-D fixed views, O^{global}, the proprioception q_{joint} and the target object. It performs visual motion planning to the target object by computing a scene point-cloud (PC^{global}), segmenting the target object (M_{obj}) to estimate its pose (q_{target}), segmenting the robot (M_{robot}) to remove it from PC^{global} and motion planning using AIT*.

network, primarily because state of the art segmentation models (Kirillov et al., 2023; Zhou et al., 2022) have significant zero-shot capabilities across objects.

Projection: In this step, we project the depth map from each global view of the scene, O^{global}_1 and O^{global}_2 into a point-cloud PC^{global} using their associated camera matrices K^{global}_1 and K^{global}_2. We perform the following processing steps to clean up PC^{global}: 1) cropping to remove all points outside the workspace 2) voxel down-sampling with a size of 0.005 m^3 to reduce the overall size of PC^{global} 3) outlier removal, which prunes points that are farther from their 20 neighboring points than the average in the point-cloud as shown in Fig. D.1.

Segmentation: We compute masks for the robot (M_{robot}) and the target object (M_{obj}) by using a segmentation model (SAM Kirillov et al. (2023)) $\mathcal{S}$ which segments the scene based on RGB input. We reduce noise in the masks by filling holes, computing contiguous mask clusters and selecting the largest mask. We use M_{robot} to remove the robot from PC^{global}, in order to perform collision checking of the robot against the scene. Additionally, we use M_{obj} along with PC^{global} to compute the object point-cloud PC^{obj}, which we average to obtain an estimate of object position, which is the target position for the motion planner. For the manipulation tasks we consider in the paper, this is the target end-effector pose of the robot, ee_{target}.

Visual Motion Planning: Given the target end-effector pose ee_{target}, we use inverse kinematics (IK) to compute q_{target} and pass $q_{joint}, q_{target}, PC^{global}$ into a joint-space motion planner. To that end, we use a sampling-based motion planner, AIT* Strub & Gammell (2020), to perform motion planning. In order to implement collision checking from vision, for a sampled joint-configuration q_{sample}, we compute the corresponding position of the robot mesh and compute the occupancy of each point in the scene point-cloud against the robot mesh. If the object is detected as grasped, then we additionally remove the object from the scene pointcloud, compute its convex hull and use the signed distance function of the joint robot-object mesh for collision checking. As a result, the Sequencing Module operates entirely over visual input, and achieves a pose near the region of interest before handing off control to the local RL policy. We emphasize that the Sequencing Module *does not need to be perfect*, it merely needs to produce a reasonable initialization for the Learning Module.

D.3 Learning Module

D.3.1 Stage Termination Details

As described in Section 3, we use stage termination conditions to determine when the Learning Module should hand control back to the Sequencing Module to continue to the next stage in the plan. For most of the tasks we consider, these stage termination conditions amount to checking for a grasp or placement for the target object in the stage. For example, for `RS-NutRound`, the plan for the first stage is (grasp, nut) and the plan for the second stage is (place, peg). Placements are straightforward

to check: simply evaluate if the object being manipulated is within a small region near the target object. This can be computed using the estimated pose of the two objects (current and target). Grasps are more challenging to estimate and we employ a two stage pipeline to detecting a grasp. First, we estimate the object pose and then evaluate if the z value has increased from when the stage began. Second, in order to ensure the object is not simply tossed in the air, we check if the robot's gripper is tightly caging the object. We do so by collision checking the object point-cloud against the gripper mesh. We use the same collision checking procedure as outlined in Sec 3 for checking collision between the scene point-cloud and robot mesh.

To estimate the turned condition, we compute the mask of the burner knob, evaluate its principal axis and measure its angle from vertical. If it is greater than X radians then the stage condition triggers. Checking the pushed condition is straightforward, the object needs to have moved by X distance forward from the start pose in xy. Finally, for checking the opened condition, we estimate the handle pose relative to the hinge and compute the angle of the door. If it is greater than X radians we consider the door opened. For the slide cabinet the handle pose itself can be used to check opening. Closing is likewise estimated as the inverse of opening. For all conditions, we take the threshold X from the environment success condition or reward function.

D.3.2 Training Details

For all tasks, we use the reward function defined by the environment, which may be dense or sparse depending on the task. We find that for PSL, it is crucial to use an action-repeat of 1, in general we found that increasing this harmed performance, in contrast to the E2E baseline which performs best with an action repeat of 2. For training policies using DRQ-v2, we use the default hyper-parameters from the paper, held constant across all tasks. We train policies using 84x84 images. We use the "medium" difficult exploration schedule defined in Yarats et al. (2021), which anneals the exploration σ from 1.0 to 0.1 over the course of 500K environment steps. Due to memory concerns, instead of using a replay buffer size of 1M as done in Yarats et al. (2021), ours is of size 750K across each task. Finally, for path length, we use the standard benchmark path length for E2E and MoPA-RL, 5 per stage for RAPS following Dalal et al. (2021), and 25 per stage for PSL.

E Baseline Implementation Details

E.1 RAPS

For this baseline, we simply take the results from the RAPS Dalal et al. (2021) paper as is, which use Dreamer Hafner et al. (2019) and sparse rewards. In initial experiments, we attempted to combine RAPS with DRQ-v2 Yarats et al. (2021) and found that Dreamer performed better, which is consistent with RAPS+Dreamer having the best results in Dalal et al. (2021). We additionally tried to run RAPS with dense rewards, but found that the method performed significantly worse. One potential reason for this is that it is not clear exactly how to aggregate the dense rewards across primitive executions - we tried simply taking the dense reward after executing a primitive as well as simply summing the rewards of intermediate primitive executions. Both performed worse than training RAPS with sparse rewards. Note that PSL outperforms RAPS even when both methods have only access to sparse rewards, e.g. the Kitchen environments. We observe clear benefits over RAPS on the single-stage (Fig. C.2) and multi-stage (Table 2) tasks.

E.2 MoPA-RL

As described in the main paper, we take the results from MoPA-RL Yamada et al. (2021) as is on the Obstructed Suite of tasks. Those results were run from state-based input and leveraged the simulator for collision checking. We do so as we were unable to successfully combine MoPA-RL with DRQ-v2 based on the publicly released implementations of both methods.

E.3 TAMP

We use PDDLStream Garrett et al. (2020a) as the TAMP algorithm of choice as it has been shown to have strong planning performance on long-horizon manipulation tasks in Robosuite (Dalal et al., 2023; Mandlekar et al., 2023). The PDDLStream planning framework models the TAMP domain and uses the adaptive algorithm, a sampling based algorithm, to plan. This TAMP method uses samplers for grasp generation, placement sampling, inverse kinematics, and motion planning, making performance stochastic. Hence we average performance across 50 evaluations to reduce variance. We adapt the authors TAMP implementation (from (Dalal et al., 2023; Mandlekar et al., 2023)) for our tasks. Note this method uses privileged access to the simulator, leveraging knowledge about the task (which must be explicitly specified in a problem file), the scene (from the domain file and access to collision checking) and 3D geometry of the environment objects.

E.4 SayCan

As described in the main paper, we re-implement SayCan Ahn et al. (2022) using GPT-4 (the same LLM we use in our methdo) and manually engineered pick/place skills that use pose-estimation and motion-planning. Following our Sequencing module: 1) we build a 3D scene point-cloud using camera calibration and depth images 2) we perform vision-based pose estimation using segmentation along with the scene point cloud and 3) we run motion planning using collision queries from the 3D point-cloud, which is used for collision queries. Finally, we use heuristically engineered pick and place primitives to perform interaction behavior which we describe as follows. We note that for our tasks of interest, the pick motion can be represented as a top-grasp. Once we position the robot near the object; we then simply lower the robot arm till the end-effector (not the grippers) come in contact with the object. We then close the gripper to execute the grasp. For place, we follow the implementation of Ahn et al. (2022) and lower the held object until contact with a surface, then release (open the gripper) and lift the robot arm. We set the affordance function for both skills to 1, following the design in Ahn et al. (2022) for motion planned skills.

For LLM planning, we specify the following prompt:

> Given the following library of robot skills: ... Task description: ... Make sure to take into account object geometry. Formatting of output: a list of robot skills. Don't output anything else.

This prompt is the same as our prompt except we specify the robot skill library in terms of object centric behaviors, instead of stage termination conditions.

> Given the following library of robot skills: ... Task description: ... Give me a simple plan to solve the task using only the provided skill library. Make sure the plan follows the formatting specified below and make sure to take into account object geometry. Formatting of output: a list of robot skills. Don't output anything else.

`Robosuite`

> **Skill Library:** pick can, pick milk, pick cereal, pick bread slice, pick silver nut, pick gold nut, put can on/in X, put milk on/in X, put cereal on/in X, put bread slide on/in X, put silver nut on/in X, put gold nut on/in X, grasp door handle, turn door handle, pick cube

`Kitchen`

> **Skill Library:** grasp vertical door handle for slide cabinet, move left, move right, grasp hinge cabinet, grasp top left burner with red tip, rotate top left burner with red tip 90 degree clockwise, rotate top left burner with red tip 90 degrees counterclockwise, push light switch knob left, push light switch knob right, grasp kettle, lift kettle, place kettle on/in X, grasp microwave handle, pull microwave handle

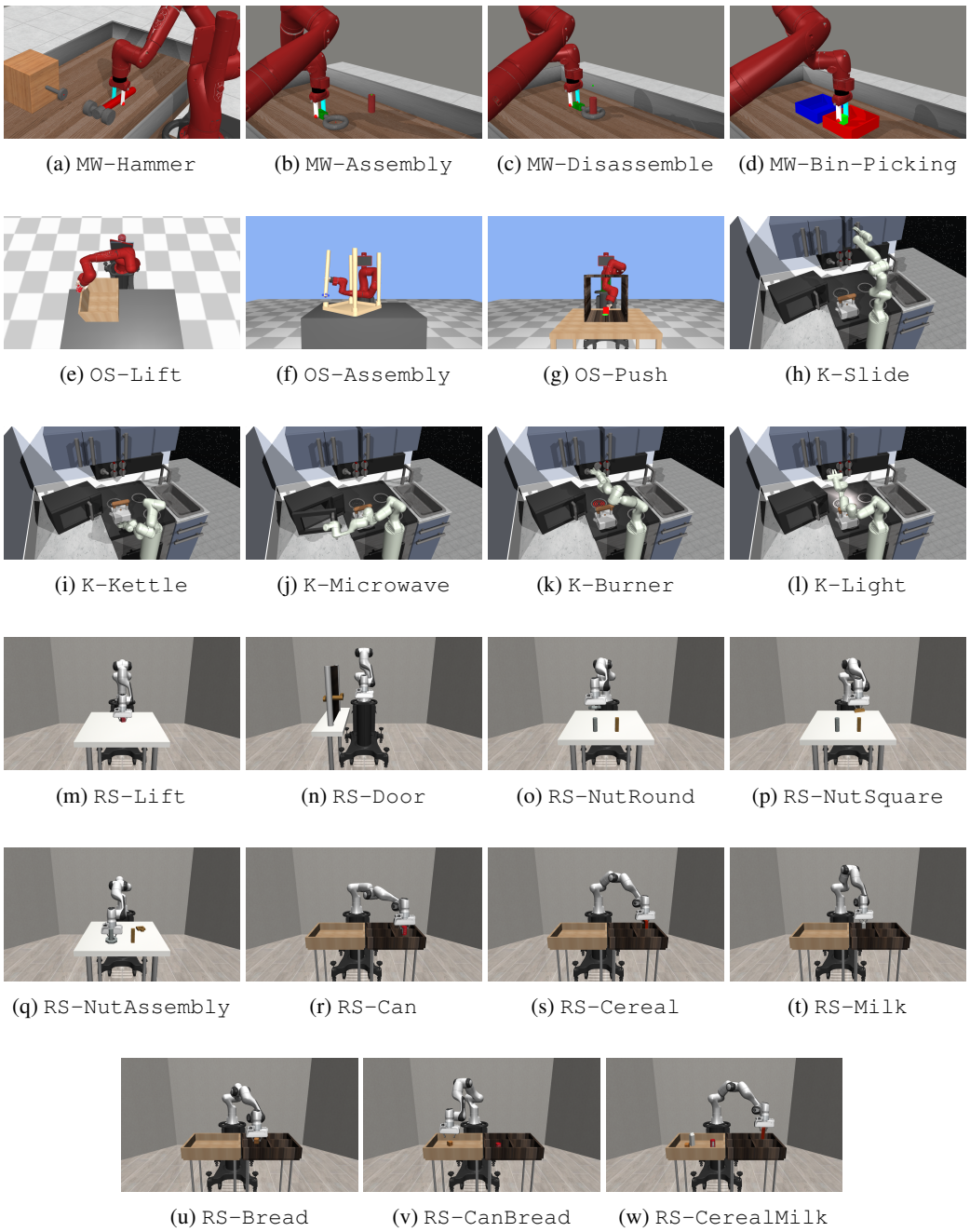
`Metaworld:`

> **Skill Library:** grasp cube, place cube on/in X, grasp hammer, place hammer, hit nail with hammer, grasp wrench, lift wrench

`Obstructed-Suite`

> **Skill Library:** grasp can, place can in bin, insert table leg in X, move table leg, grasp cube, place cube on table, push cube

F TASKS



(a) MW-Hammer (b) MW-Assembly (c) MW-Disassemble (d) MW-Bin-Picking

(e) OS-Lift (f) OS-Assembly (g) OS-Push (h) K-Slide

(i) K-Kettle (j) K-Microwave (k) K-Burner (l) K-Light

(m) RS-Lift (n) RS-Door (o) RS-NutRound (p) RS-NutSquare

(q) RS-NutAssembly (r) RS-Can (s) RS-Cereal (t) RS-Milk

(u) RS-Bread (v) RS-CanBread (w) RS-CerealMilk

Figure F.1: **Task Visualizations**. PSL is able to solve all tasks with at least 80% success rate from purely visual input.

We discuss each of the environment suites that we evaluate using PSL. All environments are simulated using the MuJoCo simulator Todorov et al. (2012).

1. **Meta-World** (Row 1 of Fig. F.1). Meta-World, introduced by Yu et al. (2020), aims to offer a standardized suite for multi-task and meta-learning methods. The benchmark consists of 50 separate manipulation tasks with a Sawyer robot, well-shaped reward functions, involve manipulating a single object to a randomized goal position, or multiple objects to a deterministic goal position. We evaluate on the single-task, multi-goal, v2 variants of the Meta-World environments. All environments use end-effector position control - a 3DOF arm action space along with gripper control - orientation is fixed. In our evaluation we use the default environment task rewards, a fixed camera view for the baselines and a wrist camera for our local policies. We refer the reader to the Meta-World paper for additional details regarding the environment suite.
2. **Obstructed Suite** (Rows 1-2 of Fig. F.1). The Obstructed Suite of tasks introduced by Yamada et al. (2021) are a challenging set of tasks requiring a Sawyer arm to perform obstacle avoidance while solving the task. The `OS-Lift` task requires the agent to pick up a can that is inside a tall box, requiring it to reach over the walls to grab the object and then lift it without making contact with the edges of the bin. The `OS-Push` environment tasks the agent with push a block to the goal in the present of a bin that forces the agent to adjust its motion in order to avoid being blocked by its upper joints. Finally, the `OS-Assembly` task involves moving the robot arm to a precise placement location while avoiding obstacles, then performing the table leg placement. Note that we evaluate our method on these environments from visual input, a more challenging setting than the one considered by Yamada et al. (2021).
3. **Kitchen** (Rows 2-3 of Fig. F.1). The Kitchen manipulation suite introduced in the Relay Policy Learning paper Gupta et al. (2019) and maintained in D4RL Fu et al. (2020) is a set of challenging, sparse reward, joint-controlled manipulation tasks in a single kitchen. We modify the benchmark to use end-effector control as we find that this significantly improves learning performance. The tasks require the ability to explore efficiently whilst also being able to chain skills across long temporal horizons, to achieve behaviors such as opening the microwave, moving the kettle, flicking the light switch, turning the top left burner, and finally sliding the cabinet door (`K-MS-5`). Aside from the single-stage tasks described in Section 4, we evaluate on three multi-stage tasks which require chaining the single-stage tasks in a particular order. `K-MS-3` involves moving the kettle, flicking the light switch and turning the top left burner, `K-MS-7` involves doing the same tasks as `K-MS-5` then turning the bottom right burner and opening the hinge cabinet and `K-MS-10` involves performing the tasks in `K-MS-7` and then opening the top right burner, opening the bottom left burner and then closing the microwave.
4. **Robosuite** (Rows 3-6 of Fig. F.1). The Robosuite benchmark from Zhu et al. (2020) contains challenging, long-horizon manipulation tasks involving pick-place and nut assembly, as well as simpler tasks that involve lifting a cube and opening a door. The rewards are coarsely defined in terms of distances to targets as well as grasp/placement conditions, which, in fact, are straightforward to implement in the real world as well using pose estimation. This stands in contrast to Meta-World which spends considerable engineering effort defining well-shaped dense rewards often by taking advantage of object geometry. As a result, learning-based methods struggle to make any progress on Robosuite tasks that involve more than a single-stage - optimizing the reward function tends to leave the agent a local minima. The suite also contains a well-tuned, realistic Operation Space Control Khatib (1987) implementation that we leverage to train policies in end-effector space.

G LLM PROMPTS AND PLANS

In this section, we list the LLM prompts per task.
Overall prompt structure:

> Stage termination conditions: (grasp, place, push, open, close, turn). Task description: ... Give me a simple plan to solve the task using only the stage termination conditions. Make sure the plan follows the formatting specified below and make sure to take into account object geometry. Formatting of output: a list in which each element looks like: (<object/region>, <operator>). Don't output anything else.

G.1 ROBOSUITE

`RS-PickPlaceCan`:

> **Task Description** can goes into bin 1.
> **Plan:** [("can", "grasp"), ("bin 1", "place")])

`RS-PickPlaceCereal`:

> **Task Description:** cereal goes into bin 3.
> **Plan:** [("cereal", "grasp"), ("bin 3", "place")])

`RS-PickPlaceMilk`:

> **Task Description**: milk goes into bin 2.
> **Plan:** [("milk", "grasp"), ("bin 2", "place")])

`RS-PickPlaceBread`:

> **Task Description:** bread slice goes into bin 4.
> **Plan:** [("bread slice", "grasp"), ("bin 4", "place")])

`RS-PickPlaceCanBread`:

> **Task Description:** can goes into bin 1, bread slice in bin 4.
> **Plan:** [("can", "grasp"), ("bin 1", "place"), ("bread slice", "grasp"), ("bin 4", "place")])

`RS-PickPlaceCerealMilk`:

> **Task Description:** milk goes into in bin 2, cereal in bin 3.
> **Plan:** [("cereal", "grasp"), ("bin 3", "place"), ("milk", "grasp"), ("bin 2", "place")])

`RS-NutAssembly:`

> **Task Description:** The silver nut goes on the silver peg and the gold nut goes on the gold peg.
> **Plan:** [("silver nut", "grasp"), ("silver peg", "place"),("gold nut", "grasp"), ("gold peg", "place")]

RS-NutAssemblySquare:

Task Description: The gold nut goes on the gold peg.
Plan: [(“gold nut”, “grasp”), (“gold peg”, “place”)]

RS-NutAssemblyRound:

Task Description: The silver nut goes on the silver peg.
Plan: [(“silver nut”, “grasp”), (“silver peg”, “place”)]

RS-Lift:

Task Description: lift the red cube.
Plan: [(“red cube”, ”grasp”)]

RS-Door:

Task Description: open the door.
Plan: [(“door handle”, “grasp”)]

G.2 META-WORLD

MW-Assembly:

Task Description: put the green wrench on the maroon peg.
Plan: [(“green wrench”, “grasp”), (“maroon peg”, “place”)]

MW-Disassemble:

Task Description: remove the green wrench from the peg.
Plan: [(“green wrench”, “grasp”)]

MW-Hammer:

Task Description: use the red hammer to push in the nail.
Plan: [(“red hammer”, “grasp”), (“nail”, “push”)]

MW-Bin-Picking:

Task Description: move the cube in the red bin into the blue bin.
Plan: [(“cube in red bin”, “grasp”), (“blue bin”, “place”)]

G.3 KITCHEN

Kitchen-Microwave:

Task Description: pull the microwave door open.
Plan: [(“microwave door handle”, “open”)]

`Kitchen-Slide`

Task Description: use the rightmost vertical bar to slide the door.
Plan: [("rightmost vertical bar", "open")]

`Kitchen-Light`

Task Description: use the round knob to flick the light switch.
Plan: [("knob", "turn")]

`Kitchen-Burner`

Task Description: rotate the top left burner with the red tip.
Plan: [("top left burner with the red tip", "turn")]

`Kitchen-Kettle`

Task Description: move the kettle forward.
Plan: [("kettle", "push")]

G.4 OBSTRUCTED SUITE

`OS-Lift:`

Task Description: lift red can from wooden bin.
Plan: [("red can', "grasp")]

`OS-Assembly:`

Task Description: move the table leg, which is already in your hand, into the empty hole.
Plan: [("empty hole', "place")]

`OS-Push:`

Task Description: push the red block onto the green circle.
Plan: [("red block", "grasp")]