PARTINSTRUCT: PART-LEVEL INSTRUCTION FOLLOWING FOR FINE-GRAINED ROBOT MANIPULATION

Yifan Yin[1]*, **Zhengtao Han**[1]*, **Shivam Aarya**[1], **Shuhang Xu**[1], **Jianxin Wang**[1], **Jiawei Peng**[1], **Angtian Wang**[1], **Alan Yuille**[1], **Tianmin Shu**[1]
[1] Johns Hopkins University
{yyin34, zhan47, saarya1, sxu79, jwang616, jpeng30, angtianwang ayuille1, tianmin.shu}@jhu.edu

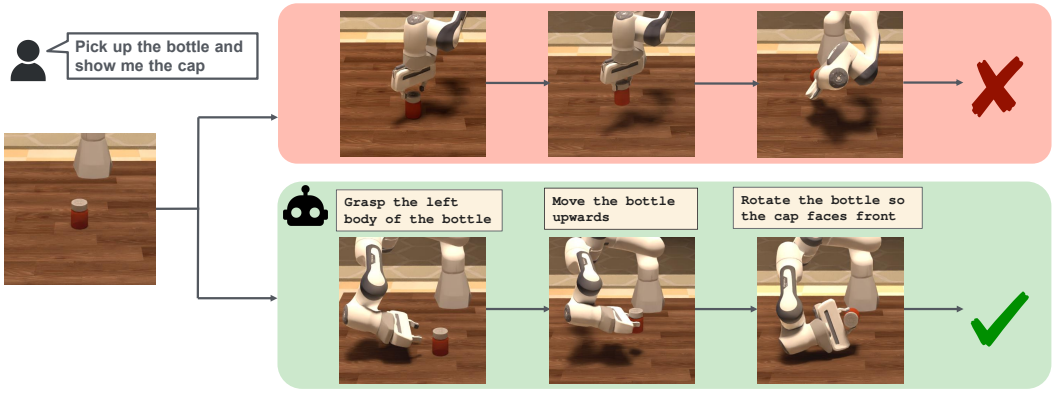

Figure 1: An example fine-grained robot manipulation task in **PartInstruct**. To successfully perform the task described in the instruction (e.g., showing the cap without occluding it), the robot needs to reason about what object parts are relevant, ground the parts to its 3D visual perception, and plan for a sequence of part-level manipulation skills (e.g., the bottom sequence).

ABSTRACT

Fine-grained robot manipulation, such as lifting and rotating a bottle to display the label on the cap, requires robust reasoning about object parts and their relationships with intended tasks. Despite recent advances in training general-purpose robot manipulation policies guided by language instructions, there is a notable lack of large-scale datasets for fine-grained manipulation tasks with part-level instructions and diverse 3D object instances annotated with part-level labels. In this work, we introduce PartInstruct, the first large-scale benchmark for both training and evaluating fine-grained robot manipulation models using part-level instructions. PartInstruct comprises 513 object instances across 14 categories, each annotated with part-level information, and 1302 fine-grained manipulation tasks organized into 16 task classes. Our training set consists of over 10,000 expert demonstrations synthesized in a 3D simulator, where each demonstration is paired with a high-level task instruction, a chain of basic part-based skill instructions, and ground-truth 3D information about the object and its parts. Additionally, we designed a comprehensive test suite to evaluate the generalizability of learned policies across new states, objects, and tasks. We evaluated several state-of-the-art robot manipulation approaches including end-to-end vision-language policy learning and bi-level planning models for robot manipulation on our benchmark. The experimental results reveal that current models struggle to robustly ground part concepts and predict actions in 3D space, and face challenges when manipulating object parts in long-horizon tasks.

*Equal contribution. **Project website:** https://partinstruct.github.io

1 INTRODUCTION

There has been an increasing interest in training general-purpose vision-language policies for robot manipulation guided by language instructions (Mees et al., 2022; Jiang et al., 2023; Zhang et al., 2023; James et al., 2020; Rahmatizadeh et al., 2018; Zhang et al., 2018), particularly with the recent advances in large generative models (Brohan et al., 2022; Team et al., 2024b). Prior works on language-guided robot manipulation have been mainly focused on high-level manipulation tasks involving simple objects (such as rearranging blocks). However, in the real world, robots often need to perform fine-grained manipulation of diverse everyday objects, in which the robots need to not only identify the target object but also understand and interact with specific parts of that object to perform the intended task as instructed. For instance, to successfully perform the manipulation task defined in the instruction as shown in Figure 1, the robot needs to identify crucial parts of the object relevant to the task (such as the label on the cap of the bottle) and reason about a chain of basic part-based skills that would lead to the desired goal state implied by the instruction, which is to display the label clearly to the human user without occlusion.

Despite the importance of part-level perception and reasoning for robot manipulation, existing robot manipulation benchmarks on instruction following lack comprehensive integration of part-level semantics in both task instructions and object ground-truth annotations (e.g., James et al., 2020; Jiang et al., 2023; Mees et al., 2022; Xiang et al., 2020b; Zhang et al., 2023). These benchmarks focus on object instance-level manipulation tasks but do not include fine-grained, part-level manipulation tasks like the example in Figure 1. There have been recent benchmarks that evaluate fine-grained, part-level manipulation tasks, but they either lack language instructions (e.g., Mu et al., 2021; Geng et al., 2023a) or do not provide training data for policy learning (e.g., Ding et al., 2024).

We introduce PartInstruct, the first large-scale benchmark for vision-language policy learning in fine-grained robot manipulation with part-level semantics. Our core idea is to develop part-level skills that enable robots to perform complex, fine-grained object manipulation tasks, including those requiring long-horizon motion plans. To support this, we built *PartGym*, a robot manipulation simulator for part-level instruction following. Built on PartGym, PartInstruct provides a rich set of 3D object assets with detailed part annotations, along with a large-scale dataset of expert demonstrations. Additionally, we developed a comprehensive evaluation suite consisting of five test sets, each corresponding to a different type of generalization test. Together, these tests assess how well a learned policy performs in unseen scenarios, including new states, objects, and tasks.

We evaluated multiple state-of-the-art vision-language policy learning methods designed for language-guided robot manipulation. We also combined recent learning-based low-level action policy planning models and VLM-based high-level task planners to create strong bi-level planning baselines for fine-grained manipulation tasks, which explicitly reasons object parts relevant to a task and how to interact with them to achieve the final goal. Our experimental results demonstrate that state-of-the-art methods still struggle with complex fine-grained manipulation tasks.

In summary, our main contribution includes (1) the first part-level instruction following benchmark for both training and evaluating fine-grained robot manipulation models' capacity for part-level grounding, reasoning, and planning; (2) a large training dataset with diverse assets and detailed annotations; (3) a comprehensive evaluation of state-of-the-art vision-language policy learning and bi-level planning baselines, revealing limitations of current robot manipulation models.

2 RELATED WORK

2.1 INSTRUCTION FOLLOWING BENCHMARKS FOR TABLE-TOP ROBOT MANIPULATION

Early benchmarks like CALVIN (Mees et al., 2022), RLbench (James et al., 2020), and VIMAbench (Jiang et al., 2023) focused on object-level manipulation but lacked part-level semantics. ManiSkill (Mu et al., 2021), PartManip (Geng et al., 2023a), and Open6DOR (Ding et al., 2024) introduced finer object part interactions, with Open6DOR incorporating spatial semantic instructions but relying on an oracle planner. Recent methods like Composable Part-based Manipulation (CPM) (Liu et al., 2024), RoboPoint (Yuan et al., 2024), and SAGE (Geng et al., 2023b) emphasize precise part interactions and semantic grasping, highlighting the need for detailed part-level understanding in manipulation tasks.

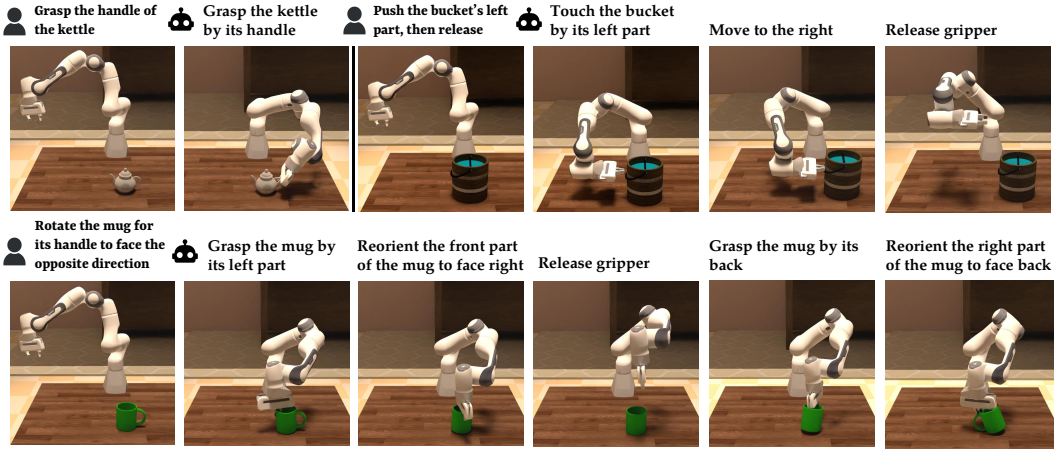

Figure 2: Example tasks and expert demonstrations in the dataset. Each task is defined by a task instruction. Each demonstration is annotated with a chain of base skills and the corresponding skill instructions (the instructions following the task instructions). Specifically, in this figure, the demonstrations for the three tasks have 1, 3, and 5 annotated skill instructions respectively.

2.2 Vision-Language Policies for Robot Manipulation

Vision-language integration in robot manipulation has led to generalist policies like RT-1 (Brohan et al., 2022), OpenVLA (Kim et al., 2024), and Octo (Team et al., 2024b), which leverage large-scale vision-language models for versatile instruction execution. Key-pose-based methods, including PerAct (Shridhar et al., 2023), Act3D (Gervet et al., 2023), and RVT (Goyal et al., 2023; 2024), focus on identifying crucial poses to simplify manipulation. Diffusion-based frameworks such as DP (Chi et al., 2023) and DP3 (Ze et al., 2024) employ DDPMs to model multimodal action distributions, generating flexible and expressive robot actions.

2.3 Robot Planning with LLMs and VLMs.

The integration of LLMs and Vision-Language Models has enhanced robotic planning by improving understanding, reasoning, and task execution. TaPA (Wu et al., 2023) and LLM-Planner (Song et al., 2023) decompose high-level instructions into actionable sub-tasks, while SayCan (Ahn et al., 2022) grounds linguistic instructions in physical affordances to ensure feasible actions. These methods enable robots to interpret and execute multi-step tasks through structured action planning.

Table 1: Comparison of PartInstruct with existing tabletop robot manipulation benchmarks based on: the number of distinctive part-level instructions, the number of part labels, the number of fine-grained part-level tasks, availability of training demonstrations, and whether these demonstrations include part-level annotations such as 2D and 3D segmentation masks.

Name	# Part Instruct	# Parts	# Part Tasks	Demo	2D Part Mask	3D Part Mask
CALVIN	6	-	6	✓	✗	✗
RLbench	136	-	64	✗	✗	✗
VIMAbench	-	-	-	✓	✗	✗
LoHoRavens	-	-	-	✓	✗	✗
ManiSkill (SAPIEN)	-	14,068	-	✓	✗	✗
PartManip	-	8,489	1,432	✗	✗	✗
Open6DOR	2,447	-	1,419	✗	✗	✗
PartInstruct (ours)	4,043	4,653	1,302	✓	✓	✓

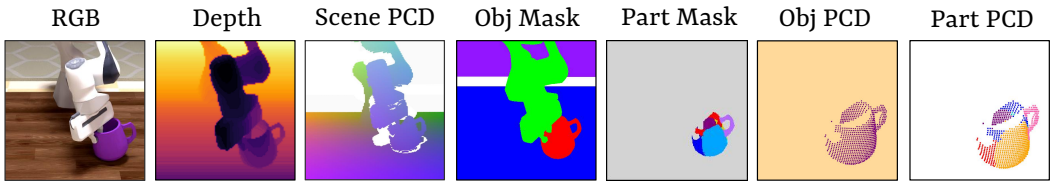

Figure 3: PartGym supports multimodal observations, including RGB images, depth maps, and scene point clouds (PCDs). It also provides object and part annotations, including object segmentations, 2D part segmentation for each object part (part mask), 3D object instance segmentation (obj PCDs), and 3D part segmentations on point clouds (part PCDs) for each object.

Table 2: Definitions of base skills.

Skill	Description
grasp_obj(obj, part)	Robot grasps *obj* at *part*.
move_gripper(dir, dis=UNIT, grasping=false)	Robot moves gripper along *dir dis*.
rotate_obj(obj, part, dir)	Robot rotates *obj*, such that *part* is facing *dir*.
touch_obj(obj, part)	Robot touches *obj* at *part*.
release_gripper(obj)	Robot releases the gripper and moves away from *obj*.

3 PartInstruct Benchmark

3.1 Problem Setup

As shown in Figure 1, a natural language instruction I_{task} describes a part-level instruction following task if it requires that a robot perform a fine-grained manipulation where the robot must interact with a list of object parts in a certain manner to achieve the intended goal g. Critically, the relevant object parts and how the robot needs to interact with them are often not explicitly described in the instructions. Thus the robot must learn to reason about relevant parts and plan how to manipulate them to perform the task successfully. To define g, we first establish a set of goal predicates that specify the states of the object, its parts, the robot's end effector, and their relationships. For example, **ON (obj, part, surface)** represents physical contact between an object part and a given surface; **FACING (obj, part, dir)** indicates the orientation of an object part from a third-person perspective; and **GRASPING (obj, part)** denotes a "grasp" interaction between the object part and the robot's end effector. Given these goal predicates, each task goal is defined by a set of goal predicates. Examples of tasks are presented in Table 3. In the task illustrated in Figure 1, the goal is represented by the predicate set {**GRASPING (bottle, ∼cap)**, **FACING (bottle, cap, front)**, **AT_POSITION (bottle, INIT_POS+VEC(UP))**}, where **∼cap** is any part other than the cap. Note that some tasks consist of multiple phases, where the next phase can only begin after completing the previous one, as the order of interactions is crucial for these tasks. For full task definitions, refer to Appendix A.1.3.

To develop an embodied agent capable of executing tasks defined by g, we hypothesize that it would be beneficial to start with a set of base skills that can be combined to handle a wide range of fine-grained manipulation tasks. In particular, we consider five types of base skills: **grasp_part**, **touch_part**, **rotate_obj**, **move_gripper**, and **release_gripper**. As detailed in Table 2 and Appendix A.1.2, each skill is parameterized by (1) the object part it interacts with and the type of interaction (e.g., touching or grasping), (2) the degree of rotation required for the part, and (3) the distance and direction in which the gripper or object should be moved. This information is summarized in a skill instruction I_{skill} associated with that skill. As illustrated in Figure 2, a task given by an overall task instruction can be decomposed into a sequence of base skill executions. We hypothesize that structuring fine-grained manipulation tasks into sequences of base skills can facilitate the training of hierarchical planning models to compose complex plans with base skills for long-horizon tasks that a end-to-end vision-language policy would struggle with.

Table 3: Example task instructions and goal states. Row A corresponds to the task illustrated in Figure 1, while rows B to D correspond to the three tasks shown in Figure 2.

	Task Instruction	Goal States
A	Rotate the *part* of the *object* to face *direction* while lifting it	`GRASPING(obj)`, `FACING(part, dir)`, `AT_POSITION(obj, POS_INIT_OBJ+VEC(UP))`
B	Grasp the *object* by the *part*	`GRASPING(gripper, part)`, `ON(obj, table)`
C	Move the *object* to *direction* by pushing it at the *part*, then free it	*Phase1*: `TOUCHING(part)`, `AT_POSITION(obj, POS_INIT_OBJ+VEC(dir))` *Phase2*: `MIN_DISTANCE(gripper, obj)`, `GRIPPER_OPEN()`
D	Rotate the *part* of the *object* to face the opposite direction	`FACING(part, ~DIR_INIT(part))`, `ON(obj, table)`

3.2 Simulation Environment

To train and evaluate language-guided part-level manipulation models, we introduce PartGym, a realistic robot simulator for fine-grained manipulation tasks requiring part-level understanding. PartGym provides (1) rich 3D assets of everyday objects, (2) part-level 3D ground-truth annotations, and (3) a large task set for fine-grained robot manipulation with natural language instructions. We used Pybullet (Coumans & Bai, 2016–2021) as the backbone physics engine to simulate the physical interactions between a robot arm and different objects and their parts. Specifically, the environment includes a 7-DoF Franka Emika Panda robot with a two-finger parallel gripper. Built upon the PartNet Mobility dataset (Xiang et al., 2020a; Mo et al., 2019; Chang et al., 2015), PartGym can simulate manipulation tasks for 14 types of table-top everyday objects annotated with different part labels. In total, there are 513 object instances and 4,653 part labels.

Observations. As shown in Figure 3, we provide multimodal observations for a robot, including RGB images, depth maps, and scene point clouds. Additionally, we provide object and part annotations. Lastly, proprioception robot states like joint states and end-effector poses are also available as part of the observations.

Action Space. The Panda robot takes a 7D action vector at each step. The first 6 dimensions represent the end-effector's absolute Cartesian pose, parameterized by a 3D coordinate as well as the roll, pitch, and yaw angles. The final dimension controls the gripper's position.

We provide more details about PartGym in Appendix A.2.

3.3 Dataset

As mentioned in Section 3.1, each fine-grained manipulation task in PartInstruct is accompanied by a natural language description of the overall task, referred to as the **task instruction** I_{task}. Additionally, we provide a sequence of **skill instructions** I_{skill} that specify the part-level manipulation **subgoals** sg to complete the task. It is important to note that skill instructions are provided only during training. For evaluation, models receive only the task instruction as the language input.

3.3.1 Task Categories

PartInstruct has 16 task categories, —10 seen and 6 unseen—each requiring specific part-level interactions. Some tasks involve direct part manipulation (e.g."Hold the part of the object and shift it in a certain direction"), requiring the agent to ground the part in visual data and predict necessary actions. Other task categories require the agent to change the state of a part (e.g. "Rotate the object such that a given part is facing a certain direction"), demanding inference of the final state of the object part and indirect manipulation when needed.

In the 6 test task categories, we have also designed more challenging part-level manipulation tasks. One focus is on long-horizon tasks that require the manipulation of multiple parts in sequence. For instance, "Push the object toward [**direction**] while touching [**part**], lift the object by holding [**part**], then rotate [**part**] to face [**direction**]." Another focus is on tasks that demand more complex reasoning about parts, the environment, and their spatial relationships. For example, consider the

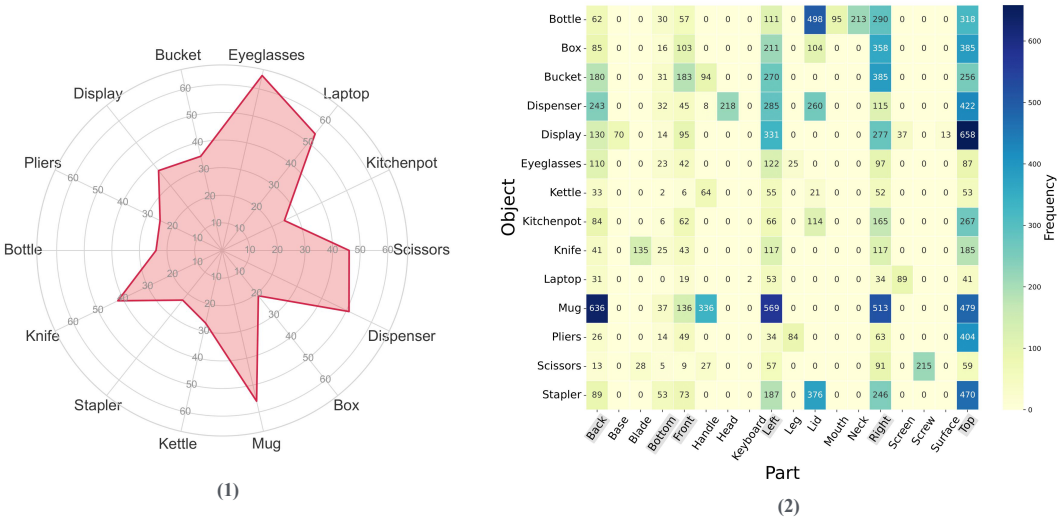

Figure 4: (1) Number of object instances in each object category; (2) Annotated parts grouped by object categories. The horizontal axis stands for different part names and the vertical axis gives different object categories. The value in the heatmap indicates the frequency of each part for an object category in PartInstruct. A darker color shows a higher frequency. Spatial part names are highlighted in light gray to distinguish them from semantic part names.

Table 4: Summary of the five test sets and the type of generalization each one addresses.

Test Set	**Type of Generalization**
Test 1 (OS)	Novel object positions and rotations
Test 2 (OI)	Novel object instances within the same category
Test 3 (TP)	Novel part combinations within the same task categories
Test 4 (TC)	Novel part-level manipulation task categories
Test 5 (OC)	Novel object categories

task, "Rotate a part of the object on the table so that it points to the opposite direction." Here, instead of explicitly naming the final state (e.g., a specific direction), the task requires the robot to have additional knowledge about the current direction of a certain part, identify its opposite direction, and manipulate the object so that the part points in that direction.

3.3.2 Trajectory Generation

We leverage a motion planner (use Breyer et al. (2021) for grasping point detection) with oracles to generate a large-scale dataset of demonstrations for vision-language imitation learning. Each demo contains an observation set with different modalities, an expert action trajectory, an overall task instruction that describes the part-level manipulation task in natural language, as well as a chain of skill instructions. See Figure 2 for several example episodes in PartInstruct. Each skill level instruction contains zero or one part that the robot is manipulating with. PartInstruct includes 10,000 demonstrations for training and over 1,800 annotated episodes for testing. Object instance distribution across object categories and the distribution of annotated parts for each object category is shown in Figure 4.

3.3.3 Evaluation Protocol

Each task defined in Section 3.1 has a binary success criterion. A task is considered complete when every action predicate in its defined predicate set has been satisfied. To systematically evaluate the performance of the learned policy, we designed a five-level evaluation protocol (see Table 4). Each test set evaluates a policy in one type of generalization condition. Specifically, they focus on

generalizability over object initial states (OS), novel object instances (OI), novel part combinations in the same task type (TP), novel task categories (TC), and novel object categories (OC). Detailed visualization can be viewed in Appendix A.2.4.

4 EXPERIMENTS

To achieve general-purpose robot manipulation, there have been two common types of approaches: (1) end-to-end policy learning that directly maps observation and instruction to actions (e.g., Zare et al., 2024; Florence et al., 2019; Mandlekar et al., 2020; Rahmatizadeh et al., 2018; Team et al., 2024b; Gervet et al., 2023; Goyal et al., 2024; Chi et al., 2023; Ze et al., 2024) and (2) bi-level planning that first generates high-level plans (typically subgoals), then compute and execute the low-level action plans to achieve the subgoals (e.g., Wu et al., 2023; Song et al., 2023; Ahn et al., 2022; Geng et al., 2023b; Wong et al., 2023). In our benchmark, we evaluate both types of approaches.

4.1 END-TO-END POLICY LEARNING

4.1.1 BASELINES

We evaluate the following state-of-the-art end-to-end robot manipulation policy learning methods:

Octo (Team et al., 2024b) is a transformer-based generalist robot policy pretrained in diverse large-scale robotic episodes.

Act3D (Gervet et al., 2023) is a 3D feature field transformer for multi-task 6-DoF robotic manipulation. Unlike Octo, it employs a key-frame-based approach to complete tasks. During a rollout, the Act3D policy is used to predict the next key pose, which is then executed by commanding the robot to the target key pose using a motion planner.

RVT2 (Goyal et al., 2024) is a multi-task transformer-based 3D manipulation model. Similar to Act3D, it also applies key-frame based manipulation.

Diffusion Policy (DP) (Chi et al., 2023) represents a visuomotor policy as a conditional denoising diffusion process in the action space, which allows it to effectively handle multimodal action distributions and high-dimensional action sequences.

3D Diffusion Policy (DP3) (Ze et al., 2024) combines 3D visual representations with diffusion-based policies, leveraging compact 3D point cloud data for efficient visuomotor policy learning.

Note that the original DP and DP3 models do not support language instruction inputs. To fit the setup of PartInstruct, we modify them to incorporate language inputs. Specifically, we use a pre-trained T5 language encoder to get the language embedding (Raffel et al., 2020). The embedding is then concatenated with other features and used as the observation condition for the denoising process.

We trained the baselines DP, DP3, Act3D, and RVT2 from scratch and fine-tuned the pretrained baseline Octo on our training data. Our hypothesis is that fine-tuning Octo will improve its performance on our benchmark by leveraging its large-scale pretraining on Open X-Embodiment (et al., 2024). The implementation details can be found in Appendix A.4.

4.1.2 RESULTS

To evaluate each learned policy, we follow the common practice outlined in recent works (Jiang et al., 2023; Chi et al., 2023; Ze et al., 2024). Specifically, we select the top two checkpoints for each baseline and conduct approximately 20 rollouts per object class across all test splits, resulting in over 1,000 rollouts per baseline. We report the *Success Rate* (SR, %) for all end-to-end policy baselines in the left part of Figure 6 and in the top block of Table 5. The low success rate across all baselines suggests that it remains challenging to train an end-to-end generalist policy for fine-grained object manipulation tasks given part-level instructions. They particularly struggle with long-horizon tasks (Test 4) and generalizing to unseen object types (Test 5).

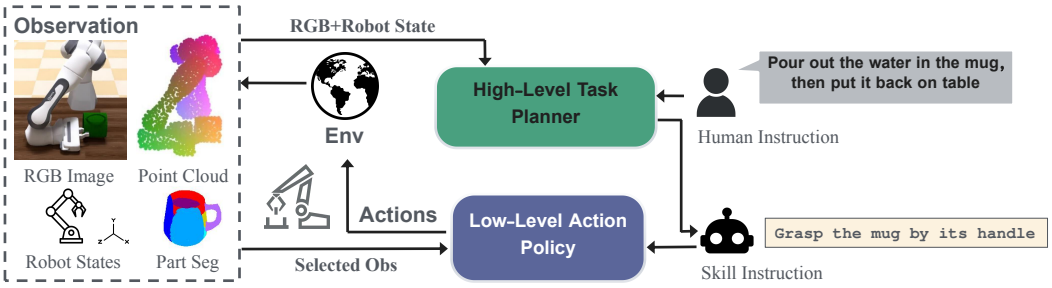

Figure 5: Overview of the bi-level planning framework. The High-Level Task Planner generates a skill instruction as a subgoal for the low-level action policy based on the task instruction and the current observation. Given the subgoal described in the skill instruction, the low-level action policy then generates actions for achieving that subgoal. The high-level task planner updates the skill instruction once every n steps, while the low-level action policy updates the action at every step.

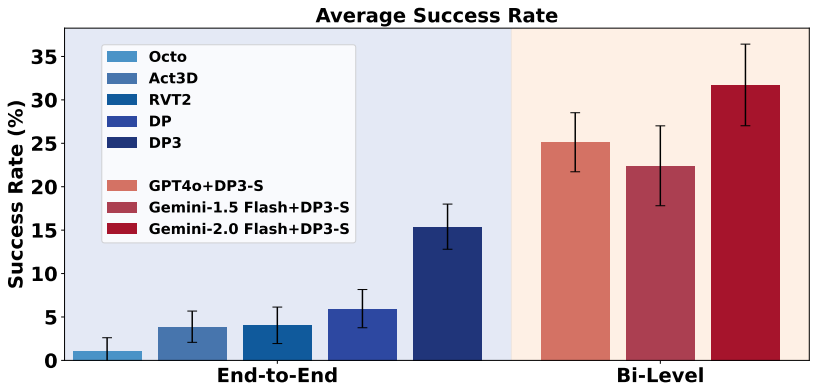

Figure 6: Success Rates of all baselines. The left group represents end-to-end learning policies, while the right group corresponds to bi-level planning models. Error bars denote the standard errors calculated across all evaluation rollouts.

Table 5: Success Rates (%) of baselines across all test sets. The top block includes end-to-end policy learning baselines, and the bottom block includes bi-level planning baselines. The best results are highlighted in blue.

Baselines	**OS**	**OI**	**TP**	**TC**	**OC**	***All***
End-to-End Policy Learning						
Octo	1.82±1.3	0.0	0.91±0.1	0.0	3.33±3.2	1.11±1.5
Act3D	6.25±1.8	5.68±1.7	4.55±1.6	0.0	2.08±2.1	3.88±1.8
RVT2	4.55±2.0	4.55±2.0	6.36±2.3	0.91±0.9	3.33±3.3	4.04±2.1
DP	7.27±1.8	8.64±1.9	8.18±1.8	3.75±2.1	6.67±3.2	5.96±2.2
DP3	23.18±2.8	23.18±2.8	18.18±2.6	7.73±1.8	6.67±3.2	15.40±2.6
Bi-Level Planning						
GPT4o+DP3	33.64±3.2	32.73±3.2	25.91±3.0	10.00±2.0	23.33±5.5	25.12±3.4
Gemini-1.5 Flash+DP3	30.48±4.5	25.45±4.2	27.62±4.4	1.82±1.8	26.67±8.1	22.41±4.6
Gemini-2.0 Flash+DP3	40.58±4.2	34.56±4.1	33.33±4.1	11.90±2.9	38.24±8.3	31.72±4.7

4.2 BI-LEVEL PLANNING

4.2.1 BASELINES

We hypothesize that it would be easier to train action policies with skill instruction annotations compared to directly training a policy for the whole task. Such low-level action policies can then be combined with a high-level planner that generates skill instructions given a task instruction to solve the manipulation task intended by the user. To evaluate the efficacy of bi-level planning on our benchmark, we extend common bi-level planning frameworks (e.g., Geng et al. (2023b)) as shown in Figure 5. Specifically, the bi-level planner consists of two modules: (1) a high-level task planner and (2) a low-level action policy. We describe each module below.

High-level Task Planner. We leverage a VLM for high-level task planning. At step t, we prompt the VLM with the task instruction I_{task} to generate the skill instruction for the current step as the subgoal sg_t, i.e., $\pi_{\text{VLM}}(sg_t|o_t, I_{\text{task}})$, where o_t is the observation at step t. We constrain the skill instructions to the space of base skills defined in Table 2 and Appendix A.1.2, which is also specified in the prompt for the VLM. To facilitate decision-making, we also provide additional observations when prompting the VLM, such as RGB images, robot states, etc. See Appendix A.4.3 for the detailed prompt. sg_t will be passed to the low-level action policy for execution and will be updated every n step. Here, n is estimated by the typical length of a skill execution in the training set.

Low-level Action Policy. The low-level action policy is a vision-language policy that generates low-level manipulation actions based on a subgoal and the current observation, i.e., $\pi(a_t|o_t, sg_t)$, where a_t is the action at step t. We can train such policies using the skill instructions annotated for training demonstrations in our dataset. We trained the end-to-end policy learning models evaluated in Section 4.1 on skill instructions as low-level action policies.

Additionally, we hypothesize that an explicit visual understanding of object parts can facilitate part-level instruction grounding. A general vision presentation containing all parts is difficult to obtain since object parts can be overlapped with each other. However, given our benchmark setup as described in Section 3.3.2, the robot interacts with at most one part for the subgoal sg_t defined in each skill instruction, making it possible to give additional vision inputs about the target object part to the low-level action policies. We select the best-performing end-to-end policy learning baselines, DP and DP3, to train the low-level action policies with object part segmentation as part of the input.

For DP, we provide a part segmentation mask as an extra vision input. Given the advanced capability of current general-purpose segmentation models like Segment Anything Model 2 (SAM 2) (Ravi et al., 2024a) in segmenting and tracking object parts, we adopt the approach of Grounded-SAM-2 (Ren et al., 2024). Specifically, given an RGB image and language input, we first utilize a VLM, e.g. Florence-2 (Ravi et al., 2024b) to ground the language onto the target part, then prompt SAM 2 to generate segmentation masks and track the object part in real-time. At each step, we add the obtained part segmentation mask as an extra channel on top of the original RGB, make the input a 4-channel image. We refer to this model as *DP-S*.

For DP3, we use a part point cloud as an additional vision input. Since there has not been a general-purpose object part segmentation model on 3D point cloud (Sun et al., 2024; Sarker et al., 2024), we obtain the 3D part segmentation using a lift-to-3D method. In detail, we first apply the same method in DP to obtain a 2D part mask tracked using SAM2. We then lift the 2D mask into 3D with the depth map and camera intrinsics. To represent a 3D part mask, we append a binary mask channel to the original point cloud observation. We refer to this action policy as *DP3-S*.

The implementation details of bi-level planning baselines can be found in Appendix A.4.3.

4.2.2 RESULTS

We adopt the same evaluation protocol described in Section 4.1.2 for bi-level planning baselines. To evaluate different low-level action policies without considering the effect of high-level task planners, we first pair each low-level action policy with ground-truth skill instructions. As shown in Table 6, *DP3-S* has the highest success rate across all test sets.

Given this result, we then adopt DP3-S as the low-level action policy and pair it with different high-level planners to create bi-level planning baselines. The results are reported in the right part of

Table 6: Performance of low-level action policies when paired with ground-truth high-level plans.

Baselines	OS	OI	TP	TC	OC	*All*
Octo	3.64	5.50	5.90	0.00	6.67	4.34
Act3D	0.45	2.80	3.33	0.00	0.00	1.32
RVT2	1.82	3.64	1.82	0.00	3.33	1.91
DP	6.47	11.42	16.53	0.06	6.94	8.28
DP3	19.34	13.61	17.89	0.00	12.08	12.58
DP-S	20.00	16.36	25.45	0.00	6.67	13.70
DP3-S	23.64	29.09	23.64	1.82	26.67	20.97

Figure 6 and the bottom block of Table 5. We can see from the results that the bi-level planning baselines outperform the end-to-end learning in every test set by a large margin. This demonstrates the effectiveness of training a separate low-level action policy for base skills and using VLM as high-level task planner. Among all high-level planning baselines, Gemini-2.0 Flash paired with DP3-S performs the best. However, bi-level planning still struggles with many tasks, particularly when the tasks require longer chains of base skills (e.g., Test 4). In these longer-horizon tasks, there is a higher chance for the high-level task planner to make mistakes. Errors from the low-level action policy are also more likely to be accumulated.

4.3 Ablation Studies

In Section 4.2, we demonstrate that bi-level planning models with low-level action policies informed by part segmentation perform significantly better than state-of-the-art end-to-end policies. To evaluate the effect of each component of the high-level planning models, we conduct the following ablation studies.

Table 7: Impact of high-level task planners on bi-level planning models. We pair each high-level task planner with an oracle motion planner to execute the skill instructions.

Baselines	OS	OI	TP	TC	OC	*All*
GPT4o	33.12	32.42	34.14	14.42	42.13	31.25
Gemini-1.5 Flash	20.41	19.07	19.36	0.15	29.24	17.65
Gemini-2.0 Flash	27.73	25.94	26.75	0.00	32.70	22.62

4.3.1 Effects of High-level Planners

To evaluate different VLMs as high-level planners, we build bi-level planners by combining each VLM with an oracle motion planner—the same one used for training demonstrations—to execute the generated skill instructions. Unlike the full bi-level planning baselines where skill instructions update at fixed intervals, here the oracle planner determines when a subgoal is reached, prompting the high-level planner to issue the next instruction only upon subgoal completion.

Table 7 shows that the bi-level planner using GPT-4o as the high-level task planner improves performance compared to Table 5 when paired with an oracle low-level planner. In contrast, Gemini-based planners perform worse with an oracle, likely because the oracle completes an incorrect skill instruction before Gemini can update it. In such cases, Gemini-based high-level task planners struggle with recovering mistakes made in the previous steps. In contrast, an incorrect skill instruction often leads to the robot arm being stuck in a pose when a learned low-level action policy is trying to perform it.

Table 8: Impact of various vision inputs on low-level action policies. We pair low-level action policies using different vision inputs with ground-truth high-level plans.

Baselines	OS	OI	TP	TC	OC	*All*
DP	6.47	11.42	16.53	0.06	6.94	8.28
DP-S GT	15.45	20.91	26.36	0.91	13.33	15.39
DP-S SAM2	20.00	16.36	25.45	0.00	6.67	13.70
DP3	19.34	13.61	17.89	0.00	12.08	12.98
DP3-S GT	45.45	36.36	36.36	1.82	40.00	32.00
DP3-S SAM2	23.64	29.09	23.64	1.82	26.67	20.97

4.3.2 EFFECTS OF DIFFERENT VISUAL INPUTS

To examine the impact of different visual representations, particularly 2D and 3D part masks, on policy learning, we conduct another ablation study, where we evaluate the low-level action policies with various visual inputs. Specifically, in addition to *DP-S SAM2* and *DP3-S SAM2*, we also trained low-level action policies using ground-truth mask information, *DP-S GT* and *DP3-S GT*, as well as the vanilla models without any part-level mask, *DP* and *DP3*. The results are summarized in Table 8. With part segmentations, either 2D or 3D, the low-level action policies can achieve significantly better performance. The performance gap between the policies trained with ground-truth part segmentation and SAM2-based part segmentation also suggests that there improvement in both the VLM's ability to ground fine-grained parts and in the capacity of state-of-the-art segmentation methods to accurately segment object parts.

4.4 ANALYSIS

4.5 POLICY PERFORMANCE ON PART-LEVEL TASKS

Current vision-language policies perform well on object-level tasks and can follow simple commands such as "grasp" or "touch". However, they struggle with precise part-level instructions like "touch the left part", which require fine-grained spatial reasoning. Zero-shot inference using pretrained generalist policies fails to achieve success (see Appendix A.4.2), likely because these models have not been trained with detailed part-level data. Our dataset and PartGym simulator are valuable because they provide the detailed part annotations and fine-grained tasks for effective training.

4.6 CHALLENGES IN PART-LEVEL INSTRUCTION FOLLOWING

The PartInstruct benchmark shows that part-level instruction following is particularly challenging for state-of-the-art vision-language policies. These models must recognize and track object parts despite variations in appearance—for example, a "lid" may look different on a bottle, pot, stapler, or mug. Moreover, task instructions often do not specify which parts to interact with, forcing the policy to infer the target, while fine-grained tasks demand higher precision and detailed spatial awareness.

Bi-level planning helps address these challenges by decomposing complex tasks into simpler subgoals that each focus on a single object part. This approach simplifies the training of low-level action policies by reducing the need to track dynamic part-level details and enables the use of pretrained vision-language models for high-level reasoning and planning. Robust visual representations are also crucial for fine-grained manipulation. Our ablation study reveals that 3D representations, such as point clouds, are more effective than 2D images because they provide precise shape and location information. Additionally, explicit object part segmentation—particularly in 3D—significantly boosts performance, with *DP3-S* outperforming *DP3* by approximately 20% (see Table 8).

5 CONCLUSION

In this work, we introduced PartInstruct, a large-scale benchmark designed to advance fine-grained robot manipulation using part-level instructions. By curating a diverse set of objects, tasks, and expert demonstrations, PartInstruct provides a foundation for training and evaluating robot manipulation models that require reasoning about object parts and their relationships with tasks. Our evaluations of state-of-the-art models highlight critical challenges in grounding part concepts and executing long-horizon tasks. With comprehensive experiments, our work provides key insights for future research, highlighting the need for further innovation in perception, reasoning, and planning to enable robots to effectively perform fine-grained, part-aware manipulation.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Michel Breyer, Jen Jen Chung, Lionel Ott, Roland Siegwart, and Juan Nieto. Volumetric grasping network: Real-time 6 dof grasp detection in clutter. In *Conference on Robot Learning*, pp. 1602–1611. PMLR, 2021.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

Yufei Ding, Haoran Geng, Chaoyi Xu, Xiaomeng Fang, Jiazhao Zhang, Songlin Wei, Qiyu Dai, Zhizheng Zhang, and He Wang. Open6dor: Benchmarking open-instruction 6-dof object rearrangement and a vlm-based approach. In *First Vision and Language for Autonomous Driving and Robotics Workshop*, 2024.

Embodiment Collaboration et al. Open x-embodiment: Robotic learning datasets and rt-x models, 2024. URL `https://arxiv.org/abs/2310.08864`.

Peter Florence, Lucas Manuelli, and Russ Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 5(2):492–499, 2019.

Haoran Geng, Ziming Li, Yiran Geng, Jiayi Chen, Hao Dong, and He Wang. Partmanip: Learning cross-category generalizable part manipulation policy from point cloud observations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2978–2988, 2023a.

Haoran Geng, Songlin Wei, Congyue Deng, Bokui Shen, He Wang, and Leonidas Guibas. Sage: Bridging semantic and actionable parts for generalizable articulated-object manipulation under language instructions. *arXiv preprint arXiv:2312.01307*, 2023b.

Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. Act3d: 3d feature field transformers for multi-task robotic manipulation. In *7th Annual Conference on Robot Learning*, 2023.

Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. Rvt: Robotic view transformer for 3d object manipulation. In *Conference on Robot Learning*, pp. 694–710. PMLR, 2023.

Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, and Dieter Fox. Rvt-2: Learning precise manipulation from few demonstrations. *arXiv preprint arXiv:2406.08545*, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Raisa Islam and Owana Marzia Moushi. Gpt-4o: The cutting-edge advancement in multimodal llm. *Authorea Preprints*, 2024.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: Robot manipulation with multimodal prompts. 2023.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Weiyu Liu, Jiayuan Mao, Joy Hsu, Tucker Hermans, Animesh Garg, and Jiajun Wu. Composable part-based manipulation. *arXiv preprint arXiv:2405.05876*, 2024.

Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. *arXiv preprint arXiv:2003.06085*, 2020.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. *arXiv preprint arXiv:2107.14483*, 2021.

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017. URL https://arxiv.org/abs/1706.02413.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3758–3765. IEEE, 2018.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024a. URL https://arxiv.org/abs/2408.00714.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos, 2024b. URL https://arxiv.org/abs/2408.00714.

Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024. URL https://arxiv.org/abs/2401.14159.

Sushmita Sarker, Prithul Sarker, Gunner Stone, Ryan Gorman, Alireza Tavakkoli, George Bebis, and Javad Sattarvand. A comprehensive overview of deep learning techniques for 3d point cloud classification and semantic segmentation. *Machine Vision and Applications*, 35(4):67, 2024.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pp. 785–799. PMLR, 2023.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2998–3009, 2023.

Yuliang Sun, Xudong Zhang, and Yongwei Miao. A review of point cloud segmentation for understanding 3d indoor scenes. *Visual Intelligence*, 2(1):14, 2024.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024a.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024b.

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S Siegel, Jiahai Feng, Noa Korneev, Joshua B Tenenbaum, and Jacob Andreas. Learning adaptive planning representations with natural language guidance. *arXiv preprint arXiv:2312.08566*, 2023.

Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied task planning with large language models. *arXiv preprint arXiv:2307.01848*, 2023.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020a.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11097–11107, 2020b.

Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. Robopoint: A vision-language model for spatial affordance prediction for robotics. *arXiv preprint arXiv:2406.10721*, 2024.

Maryam Zare, Parham M Kebria, Abbas Khosravi, and Saeid Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges. *IEEE Transactions on Cybernetics*, 2024.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy. *arXiv preprint arXiv:2403.03954*, 2024.

Shengqiang Zhang, Philipp Wicke, Lütfi Kerem Şenel, Luis Figueredo, Abdeldjallil Naceri, Sami Haddadin, Barbara Plank, and Hinrich Schütze. Lohoravens: A long-horizon language-conditioned benchmark for robotic tabletop manipulation. *arXiv preprint arXiv:2310.12020*, 2023.

Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 5628–5635. IEEE, 2018.

A APPENDIX

A.1 PDDL DEFINITIONS

A.1.1 PREDICATE DEFINITIONS

This subsection gives the definition of the basic predicates utilized by the motion planner.

Table 9: Definition of Predicates

Predicate	Description
`ON(obj, part, contact)`	Whether `obj` is on the `contact`.
`TOUCHING(obj, part)`	Whether the gripper is in contact with `obj` at `part`; `part` can be empty to indicate a general touch on any parts.
`GRASPING(obj, part)`	Whether the gripper is carrying `obj` at `part`; `part` can be empty to indicate a general grasp with any parts.
`FACING(obj, part, dir)`	Whether `part` of `obj` is facing or pointing `dir`.
`AT_POSITION(obj, pos)`	Whether `obj` is at position `pos` = [x,y,z].

A.1.2 SKILL DEFINITIONS

This subsection shows the detailed definition of the five skills.

Table 10: Definition of Base Skills

Skill	Description	Preconditions	Effects
`grasp_obj(obj, part)`	Robot grasps *obj* at *part*.	`ON(table, obj)`; `~GRASPING(obj)`; `~TOUCHING(obj)`	`GRASPING(obj, part)`
`move_gripper (dir,dis=UNIT, grasping=false)`	Robot moves gripper along *dir* *dis*.	**If *grasping*==True:** `GRASPING(obj)`	`AT_POSITION( gripper, last_gripper_pos + vec(dir)×dis)`; **If *grasping*==True:** `GRASPING(obj)`
`rotate_obj(obj, part, dir)`	Robot rotates *obj*, such that *part* is facing *dir*.	`GRASPING(obj)`	`GRASPING(obj)`; `FACING(part, dir)`
`touch_obj(obj, part)`	Robot touches *obj* at *part*.	`ON(table, obj)`; `~GRASPING(obj)`; `~TOUCHING(obj)`	`TOUCHING(obj, part)`
`release_gripper (obj)`	Robot releases the gripper and moves away from *obj*.	`ON(table, obj)`; `GRASPING(obj)` *or* `TOUCHING(obj)`	`ON(table, obj)`; `~GRASPING(obj)`; `~TOUCHING(obj)`

A.1.3 TASK DEFINITIONS

This subsection shows the detailed definition of different task types in PartInstruct.

Table 11: Seen Task Instructions and Goal States

Seen (10)		
Order	**Example Task Instruction**	**Goal States**
1	Grasp the *object* by the *part*	`GRASPING(gripper, part),ON(obj, table)`
2	Touch the *object* at the *part*	`TOUCHING(part),ON(obj, table)`
3	Hold the *part* of the *object* and move it to *direction*	`GRASPING(part), AT_POSITION(obj, POS_INIT_OBJ+VEC(dir))`
4	Push the *object* towards *direction* by touching *part*	`TOUCHING(part), AT_POSITION(obj, POS_INIT_OBJ +VEC(dir))`
5	Slide the *object* on the table towards *direction* while keeping hold of *part*, then release it	*Phase1*: `GRASPING(part), AT_POSITION(obj, POS_INIT_OBJ+VEC(dir))` *Phase2*: `GRIPPER_OPEN, MIN_DISTANCE(gripper, obj)`
6	Move the *object* to *direction* by pushing it at *part*, then free it	*Phase1*: `TOUCHING(part), AT_POSITION(obj, POS_INIT_OBJ+VEC(dir))` *Phase2*: `GRIPPER_OPEN, MIN_DISTANCE(gripper, obj)`
7	While keeping hold of *part*, move the *object* towards *direction* in the air	`GRASPING(part), AT_POSITION(obj, POS_INIT_OBJ +VEC(dir)+VEC(UP))`
8	Rotate *part* of the *object* to face *direction* while lifting it	`GRASPING(obj),FACING(part, dir), AT_POSITION(obj, POS_INIT_OBJ+VEC(UP))`
9	Move the *object* towards *direction* after raising it, while keeping hold of *part*, then put it down	*Phase1*: `GRASPING(part), AT_POSITION(obj, POS_INIT_OBJ+VEC(dir)+VEC(UP))` *Phase2*: `GRASPING(part), AT_POSITION(obj, POS_INIT_OBJ+VEC(dir))`
10	Move the *object* towards *direction1* in the air, then rotate *part* to point towards *direction2*	`GRASPING(obj), AT_POSITION(obj, POS_INIT_OBJ+VEC(UP)+VEC(dir1)), FACING(part, dir2)`

Table 12: Unseen Task Instructions and Goal States

Unseen (6)		
Order	**Example Task Instruction**	**Goal States**
11	Rotate *part* in the air so it points towards *direction*, then put it down	*Phase1*: `GRASPING(obj), FACING(part, dir), AT_POSITION(obj, POS_INIT_OBJ+VEC(UP))` *Phase2*: `GRASPING(obj), FACING(part, dir), AT_POSITION(obj, POS_INIT_OBJ)`
12	Shift the *object* towards *direction1* in the air while grasping *part1*, turn *part2* to *direction2*, then set it down	*Phase1*: `GRASPING(part1), FACING(part, dir), AT_POSITION(obj,POS_INIT_OBJ+VEC(dir1) +VEC(UP))` *Phase2*: `GRASPING(part1), FACING(part, dir), AT_POSITION(obj, POS_INIT_OBJ+VEC(dir1))`
13	Turn *part* of the *object* to point to *direction1* while keeping it on the table, then push it towards *direction2*	*Phase1*: `ON(obj, table),FACING(part, dir1);` *Phase2*: `ON(obj, table), AT_POSITION(obj, POS_INIT_OBJ+VEC(dir2))`
14	While keeping it on the table, push the *object* towards *direction1* while touching *part1*, then rotate *part2* to face *direction2*	*Phase1*: `ON(obj, table), TOUCHING(part1), AT_POSITION(obj, POS_INIT_OBJ+VEC(dir1))` *Phase2*: `ON(obj, table),FACING(part2, dir2)`
15	Rotate *part* of the *object* to face the opposite direction	`FACING(part, ∼DIR_INIT(part)), ON(obj, table)`
16	Push the *object* to *direction1* and rotate *part* to point towards *direction2* in the air, finally place it down	*Phase1*: `FACING(part, dir2), AT_POSITION(obj,POS_INIT_OBJ+ VEC(dir1)+ VEC(UP))` *Phase2*: `FACING(part, dir2), AT_POSITION(obj,POS_INIT_OBJ+VEC(dir1))`

A.2 BENCHMARK

A.2.1 OBSERVATION AND ACTION SPACE

Table 13 shows the observation and action space available in *PartGym*.

Table 13: Observation and Action Space details.

Observation Space	
Static View - RGB	$300 \times 300 \times 3$
Static View - Depth	300×300
Static View - PCD	3×1024
Static View - Semantic	300×300
Static View - Traget Part PCD	3×1024
Static View - Traget Part Mask	300×300
Wrist View - RGB	$300 \times 300 \times 3$
Wrist View - Depth	300×300
Wrist View - PCD	3×1024
Wrist View - Semantic	300×300
Wrist View - Traget Part Mask	300×300
Wrist View - Traget Part PCD	3×1024
Proprioceptive state	EE position (3) EE orientation (3) Joint positions (7) Gripper action (1)
Action Space	
Absolute cartesian pose (w.r.t. world frame)	EE position (3) EE orientation (3) Gripper action (1)

A.2.2 TABLE-TOP OBJECTS IN PARTGYM

PartGym provides in total 513 object instances across 14 categories. Some objects and their textures are shown in Figure 7.

Figure 7: Different object instances in PartGym.

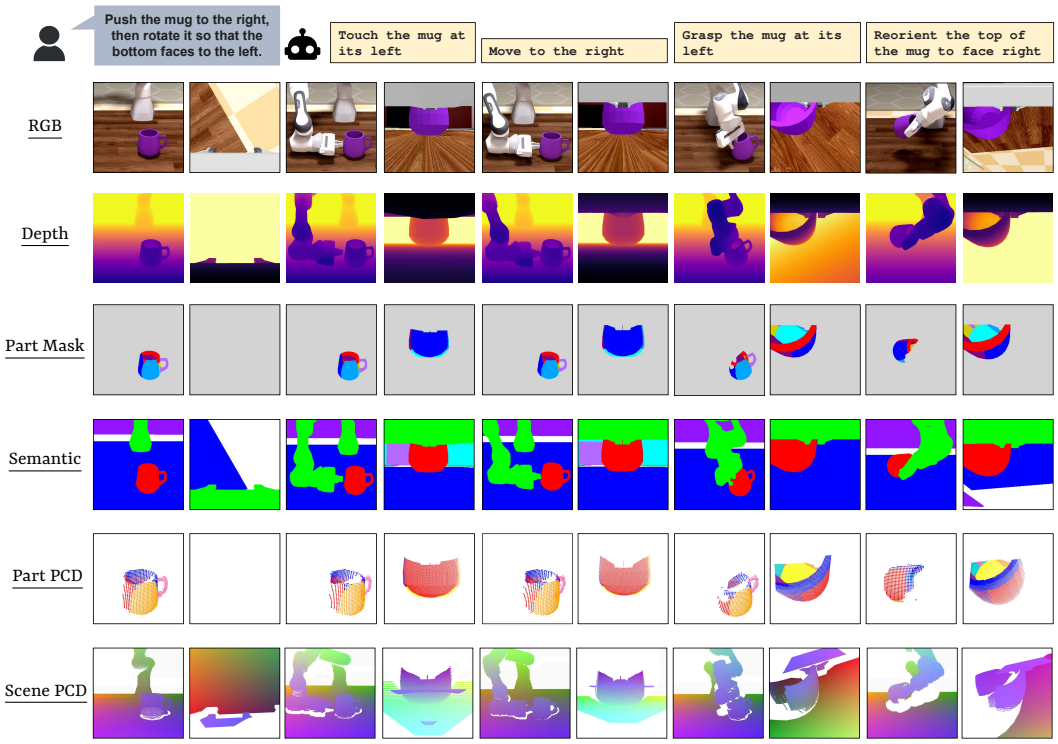

Figure 8: Different visual modalities in PartGym.

A.2.3 Details of PartGym

The aim of PartGym is to boost embodied AI research related to interaction with table-top object parts. PartGym support real-time rendering of different visual modalities (see Figure 8). In additional to the typical modalities like RGB, depth and object segmentation, PartGym also provides part-related visions like part masks and part point cloud, including spatial parts and semantic parts of an object. These part-related vision modalities are renderred by *PyBullet* (Coumans & Bai, 2016–2021) simulation engine using the ground-truth part assets given by *PartNet Mobility* (Xiang et al., 2020a; Mo et al., 2019; Chang et al., 2015).

Additionally, PartGym provides a framework to implement bi-level planning models for part-level manipulation tasks in simulation environments. It provides a template skill instruction generator, an oracle skill execution checker, as well as a systematical way to render part-related modalities shown in any skill instruction.

A.2.4 VISUALIZATION OF DIFFERENT TEST SPLITS

We provide the visualization of all 5 test sets in this section.

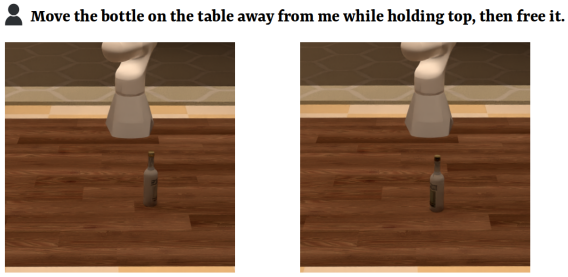

Figure 9: Left: Training set. Right: Test 1(OS).

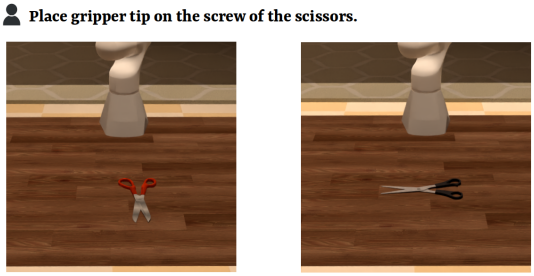

Figure 10: Left: Training set. Right: Test 2(OI).

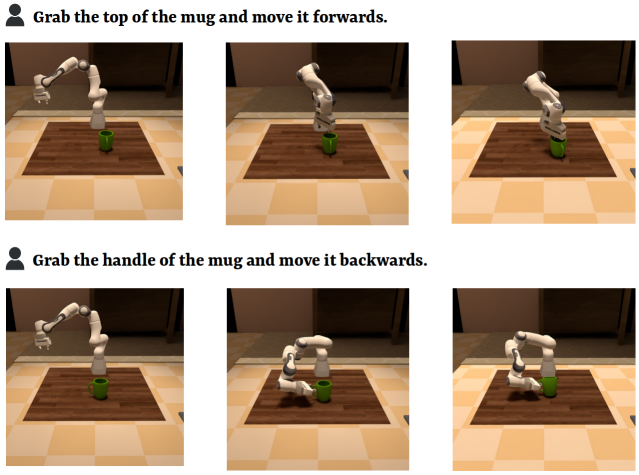

Figure 11: Above: Training set. Below: Test 3(TP).

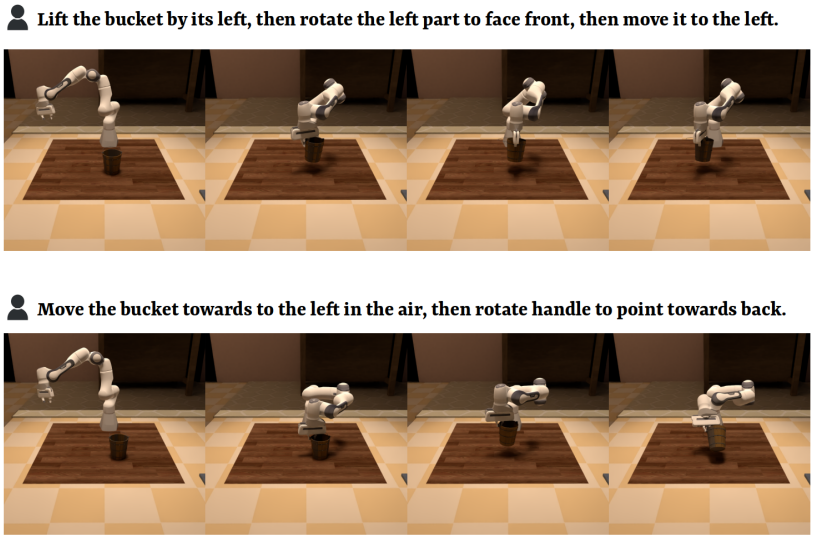

Figure 12: Above: Training set. Below: Test 4(TC).

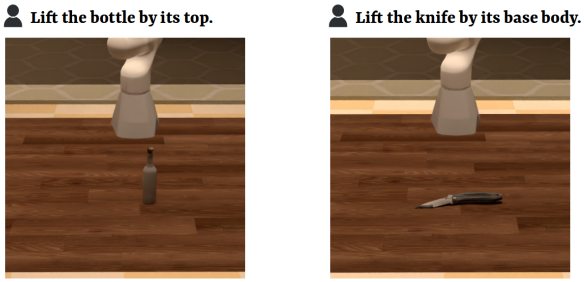

Figure 13: Left: Training set. Right: Test 5(OC).

A.2.5 STATISTICS OF PARTINSTRUCT EPISODES

We provided detailed statistics about parts within each object type.

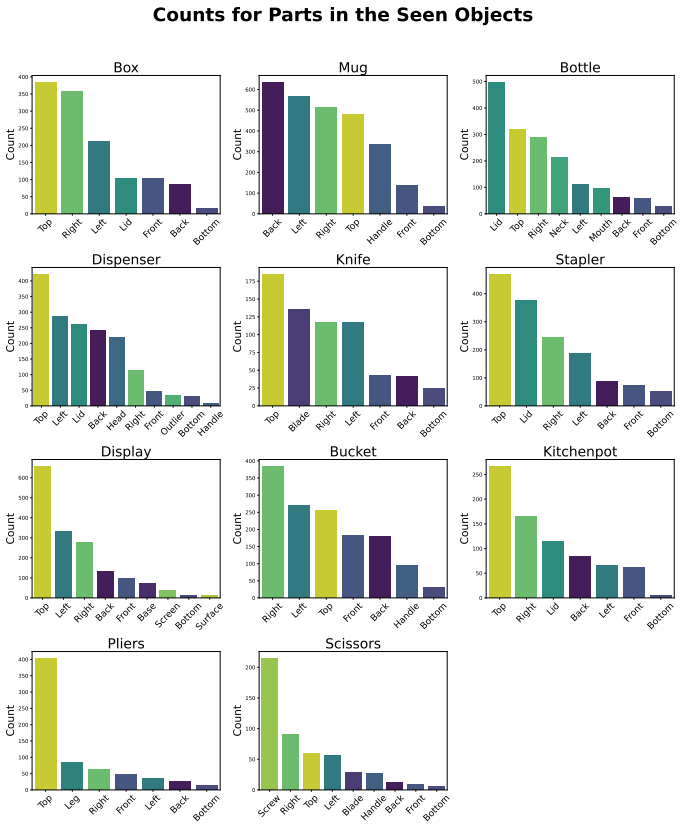

Figure 14: Parts in PartInstruct episodes, grouped by seen object types.

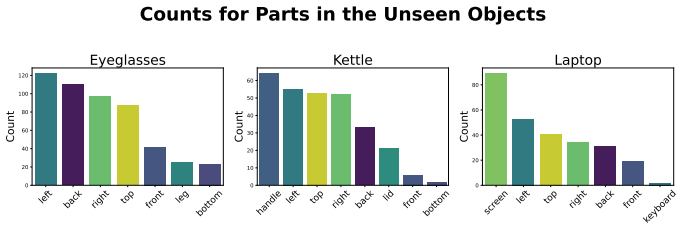

Figure 15: Parts in PartInstruct episodes, grouped by unseen object types.

A.3 Skill and Object Part Impact Study

Here, we selected the rollout logs of the best-performing policy and analyzed the impact of different skill types and object parts. Specifically, we evaluated the success rate and failure causes for each skill and part.

The Success Rate was calculated by dividing the number of successful executions of each skill or part by the number of times it appeared in the skill chain. The Failure Cause was calculated by dividing the number of times a skill chain failed because of a specific skill or part by the total number of skill chain failures.

Table 14: Average success rate and failure cause for the three part-level skills

Skill	**Grasp Object**	**Rotate Object**	**Touch Object**
Success Rate (%)	53.55	18.18	54.55
Failure Cause (%)	43.51	6.11	16.79

Table 15: Average success rate and failure cause for selected parts.

Part	**Blade**	**Left**	**Neck**	**Top**	**Screen**	**Mouth**	**Bottom**
Success Rate (%)	46.67	45.10	66.67	52.78	60.00	66.67	30.00
Failure Cause (%)	4.60	13.79	1.15	24.14	2.30	0.00	0.00

Part	**Handle**	**Leg**	**Lid**	**Front**	**Right**	**Back**	**Screw**	**Head**
Success Rate (%)	64.29	33.33	63.64	29.03	36.49	41.03	0.00	16.67
Failure Cause (%)	4.60	2.30	4.60	5.75	21.84	9.20	1.15	3.45

A.4 Implementation Details

A.4.1 Training Details in End-to-End Policy Learning

We trained the baseline models, including Diffusion Policy (DP) (Chi et al., 2023), 3D Diffusion Policy (DP3) (Ze et al., 2024), and Act3D (Gervet et al., 2023), from scratch. For RVT2 (Goyal et al., 2023) and Octo (Team et al., 2024b), we implemented both fine-tuning of the pretrained models and training from scratch on our dataset. All trained models are using vision modalities from a static-view camera put with the same extrinsics in the workspace, as well as the real-time robot states information. Experiments were conducted on cluster nodes of A100 or H100 using Distributed Data Parallel (DDP). Training from scratch generally took about two days, while fine-tuning required one day.

Diffusion Policy (DP)

We train CNN-based DP from scratch on our dataset. The action prediction horizon is set to 16 steps, with an observation horizon of 2 steps and action steps of 8. The input RGB images are cropped to a size of 76×76. For language instructions, we use a pre-trained T5-small language encoder to obtain a language embedding of 512 dimensions. This language embedding is then concatenated with other features to form the final feature representation.

3D Diffusion Policy (DP3)

The DP3 model is trained under a similar setup as DP, with an action prediction horizon of 16 steps, an observation horizon of 2 steps, and action steps of 8. For the point cloud observations, we use an input size of 1024 points, which are downsampled from the original point cloud using the Iterative Farthest Point Sampling algorithm (Qi et al., 2017). The language instructions are processed in DP3 following the same approach as in DP.

ACT3D

Act3D takes an image input size of 256×256. The action prediction horizon is set to 6 steps, and the observation horizon is 1 step. Following the raw work (Gervet et al., 2023), we use ResNet50(He et al., 2016) as the vision encoder, and use CLIP (Radford et al., 2021) embeddings for vision-language alignment. For 3D action map generation, the number of "ghost" points is set to be 10,000, with a number of sampling level of 3.

OCTO

For fine-tuning, we use the released checkpoint of the `octo-base-1.5` model and fine-tune its output head for 20,000 iterations. We use both the static view camera and the wrist view camera. The input image sizes are 256×256 for the static view and 128×128 for the wrist view. The window size is set to 2 steps, and the action horizon is set to 16 steps.

RVT2

To adapt RVT2 in our benchmark settings, we first convert the depth map from the static camera view into a point cloud in the camera coordinates, then apply camera extrinsic to transfer the point cloud into the world coordinates, where the action heat maps will be generated and apply supervision in. The action prediction horizon is chosen to be 6 steps, and the observation horizon is set to be 1 step.

A.4.2 RESULTS OF PRE-TRAINED GENERALIST POLICIES

We selected several popular generalist policy, including RT-1, Octo and OpenVLA, and evaluated their zero-shot performance on our test sets. For RT-1, we followed the implementation of Open X-Embodiment project and used the released `rt_1_x_tf_trained_for_002272480_step` checkpoint for inference. For Octo, we used `octo-base-1.5` model, following the same setup as described in A.4.1. For OpenVLA, we used the pretrained model `openvla-7b`. We followed the same evaluation protocol as other baselines, and our results shows that these generalist policies fail to achieve any success on our test sets.

A.4.3 IMPLEMENTATION DETAILS IN BI-LEVEL PLANNING

IMPLEMENTATION OF THE HIGH-LEVEL TASK PLANNER

The high-level task planner features a skill inference mechanism that leverages comprehensive contextual information—including user task instructions, previously executed skill chains, and real-time state data such as vision and pose information—to determine the next appropriate action. Recall that the high-level task planner updates the skill instruction once every n steps. Here n is determined by the average number of steps typically required for each skill in the training dataset. Specifically, we use 130 for **`grasp_obj`**, 30 for **`move_gripper`**, 68 for **`touch_obj`**, 40 for **`release_obj`**, and 22 for **`rotate_obj`**. Once the execution counter reaches these average values, the VLM is prompted to infer the subsequent skill based on the current state. This design ensures that the decisions are grounded in both historical and real-time data. In addition, the planner incorporates an exception handling measure to maintain output consistency and reliability. Any unacceptable terms generated by the VLM—such as directional indicators that out of our definition, will be normalized to their prescribed equivalents. Also, despite embedding all acceptable part names within the prompt, a dedicated mechanism cross-references any inferred part names against a stored list of valid part names. For VLM baselines, we use `gemini-1.5-flash-002` and `gemini-2.0-flash-exp` for Gemini models Team et al. (2024a), and `GPT4o` for the OpenAI model Islam & Moushi (2024).

TRAINING OF THE LOW-LEVEL ACTION POLICIES

We trained the low-level action policy using skill instructions retrieved from our training data, assuming the presence of an oracle planner that decomposes the overall task. Apart from the skill instructions, the training setup remains identical to the end-to-end learning approach described in A.4.1

PART GROUNDING AND TRACKING

We selected `sam2_hiera_small` as our mask generation and tracking model due to its fastest tracking time among all configurations of SAM 2. For language grounding, we chose `Florence-2-large` as our Vision-Language Model. To evaluate the performance, we used the rollout logs of *DP-S SAM2*.

The performance was assessed using two key metrics: Grounding Success, Intersection over Union (IoU). Grounding Success is calculated as the ratio of successfully grounded parts to the total number of parts during a task. A grounding is considered successful if: 1) after language grounding, the prompt points given by the VLM consist of one positive and one negative point (to prompt SAM 2), and 2) the IoU of the generated mask is greater than zero. If either of these conditions is not met, the grounding is deemed a failure. The IoU measures the overlap between the predicted mask genrated by SAM 2 and the ground-truth mask retrieved from PartGym environment, it is defined as the area of intersection divided by the area of union of the predicted and true regions. The results across different test sets are summarized in Table 16.

Table 16: Performance of part grounding and tracking across different test sets.

Metric	**Test1**	**Test2**	**Test3**	**Test4**	***All***
Grounding Success (%)	25.26	35.73	36.87	15.06	27.58
IoU	0.15	0.18	0.19	0.25	0.20

VLM PROMPTS

We provide implementation details on high-level task planner in this section. Below is the VLM prompts.

Prompt Example

You are an expert at planning manipulation tasks. You will be given one task instruction for each manipulation task. Each task instruction can be divided into a chain of skill instructions. Your job is to infer the next skill instruction (you only need to output one immediate next skill instruction each time, even if the entire task requires multiple skills) for the robot to execute, based on the following information and the attached image (current rgb frame) without outputting any intermediate inference and explanation:
- **Task Instruction:** {user_input}
- **Executed Skill Instructions:** {executed_skill_instructions}
- **Gripper State:** gripper_state (The gripper is open when the value is around 0.04 and it is closed when the value is less or around 0.018.)
- **First and Last TCP Pose:** This is the tcp pose of the first frame: first_tcp_pose and this is the tcp pose of the last frame: last_tcp_pose. (The tcp format is np.r_[self.rotation.as_quat(), self.translation] where the first four elements are the rotation quaternion and the last three are the translation element. You can use the tcp poses difference to infer if the move_gripper skill is complete (the Euclidean distance of two tcp positions should be at least 0.05). Please analyze the images and determine whether the task is completed between the first and last frames. Focus on the position of the object relative to the gripper. A small gap between the gripper and the object can be allowed.)

Here is more relevant information about the task.

Skill Descriptions:
1. **grasp_obj:**
- *Description:* This skill grasps an object by a specific part.
- *Parameters:*
  part_grasp: The exact part of the object to be grasped. Must match the user's input (e.g., 'blade', 'lid').
- *Format:* Grasp the {obj_class} at its {part_grasp}
2. **move_gripper:**
- *Description:* This skill moves the gripper in a specified direction while optionally keeping an object grasped.
- *Parameters:*
  dir_move: Direction to move the gripper. Can only be 'top', 'bottom', 'left', 'right', 'front', or 'back'.
- *Format:* Move {dir_str}, where 'dir_str' is mapped from 'dir_move' by: - 'front' → 'forwards' - 'back' → 'backwards' - 'top' → 'upwards' - 'bottom' → 'downwards' - 'left' → 'to the left' - 'right' → 'to the right'
3. **rotate_obj:**
- *Description:* This skill rotates an object in a specific direction based on a given part.
- *Parameters:*
  dir_rotate: Direction to rotate the object. Must be one of 'top', 'bottom', 'left', 'right', 'front', 'back'.
  part_rotate: The part of the object that should be rotated.
- *Format:* Reorient the {part_rotate} of the {obj_class} to face {dir_str}, where 'dir_str' is mapped from 'dir_rotate'.
4. **touch_obj:**
- *Description:* This skill touches a part of an object.
- *Parameters:*
  part_touch: The part of the object to be touched.
- *Format:* Touch the {obj_class} at its {part_touch}
5. **release_obj:**
- *Description:* This skill releases an object from the gripper.
- *Parameters:* None.
- *Format:* Release

Prompt Example (Continued)

Part Names:
- **Scissors:** blade, handle, screw, left, right, top, bottom, front, back
- **Kitchen Pot:** base body, lid, left, right, top, bottom, front, back
- **Laptop:** base frame, screen, touchpad, keyboard, screen frame, left, right, top, bottom, front, back
- **Eyeglasses:** base body, leg, left, right, top, bottom, front, back
- **Bucket:** handle, base body, left, right, top, bottom, front, back
- **Display:** base support, surface, frame, screen, left, right, top, bottom, front, back
- **Pliers:** base body, leg, outlier, left, right, top, bottom, front, back
- **Bottle:** mouth, lid, body, neck, left, right, top, bottom, front, back
- **Knife:** base body, translation blade, rotation blade, left, right, top, bottom, front, back
- **Stapler:** base body, lid, body, left, right, top, bottom, front, back
- **Kettle:** handle, base body, lid, left, right, top, bottom, front, back
- **Mug:** handle, body, containing things, left, right, top, bottom, front, back
- **Box:** rotation lid, base body, left, right, top, bottom, front, back
- **Dispenser:** base body, pressing lid, head, handle, outlier, left, right, top, bottom, front, back

Task Splitting Example:
Break down the task: Split the task instruction into individual steps. Example: "Move the box in the air towards the right while keeping touch of right, then put it down." Steps: (1) Grasp the box at its right, (2) Move upwards, (3) Move to the right, (4) Move downwards. *Return only the next skill instruction in the specified format.*

Notes:
- Do not modify or assume alternate names for object parts.
- The task sequence should follow the user's input as strictly as possible.
- Do not replace object parts with similar or inferred names.
- Only bucket, mug, and scissors have a part called handle. Do not infer the handle part name for other objects.