

A Real-to-Sim-to-Real Approach to Robotic Manipulation with VLM-Generated Iterative Keypoint Rewards

Shivansh Patel^{1*}, Xincheng Yin^{1*}, Wenlong Huang², Shubham Garg³, Hooshang Nayyeri³, Li Fei-Fei², Svetlana Lazebnik¹, Yunzhu Li⁴

Abstract—Task specification for robotic manipulation in open-world environments is challenging, requiring flexible and adaptive objectives that align with human intentions and can evolve through iterative feedback. We introduce Iterative Keypoint Reward (IKER), a visually grounded, Python-based reward function that serves as a dynamic task specification. Our framework leverages VLMs to generate and refine these reward functions for multi-step manipulation tasks. Given RGB-D observations and free-form language instructions, we sample keypoints in the scene and generate a reward function conditioned on these keypoints. IKER operates on the spatial relationships between keypoints, leveraging commonsense priors about the desired behaviors, and enabling precise SE(3) control. We reconstruct real-world scenes in simulation and use the generated rewards to train reinforcement learning (RL) policies, which are then deployed into the real world—forming a real-to-sim-to-real loop. Our approach demonstrates notable capabilities across diverse scenarios, including both prehensile and non-prehensile tasks, showcasing multi-step task execution, spontaneous error recovery, and on-the-fly strategy adjustments. The results highlight IKER’s effectiveness in enabling robots to perform multi-step tasks in dynamic environments through iterative reward shaping. Project Page: <https://iker-robot.github.io/>

I. INTRODUCTION

Suppose that a robot is tasked with placing a pair of shoes on a rack, but a shoe box is occupying the rack, leaving insufficient space for both shoes (Figure 1, top right). The robot must first push the box aside to create space and then proceed to place the shoes. This example highlights the importance of task specification for robots in unstructured, real-world environments, where tasks can often involve multiple implicit steps. In such cases, rigid predefined instructions fail to capture the complexities of interaction required to accomplish the goal. To be effective, task specifications must incorporate commonsense priors—expectations about how the robot should behave. For instance, rather than attempting to squeeze the shoes in awkwardly, the robot should realize that it must first clear space.

Recent vision-language models (VLMs) show promise for freeform robotic task specification due to their rapidly advancing ability to encode rich world knowledge by pre-training on vast and diverse datasets [1–8]. VLMs excel in interpreting natural language descriptions and complex instructions. Their broad knowledge bridges human expectations and robot behavior, capturing human-like priors and problem-solving strategies. However, previous works that

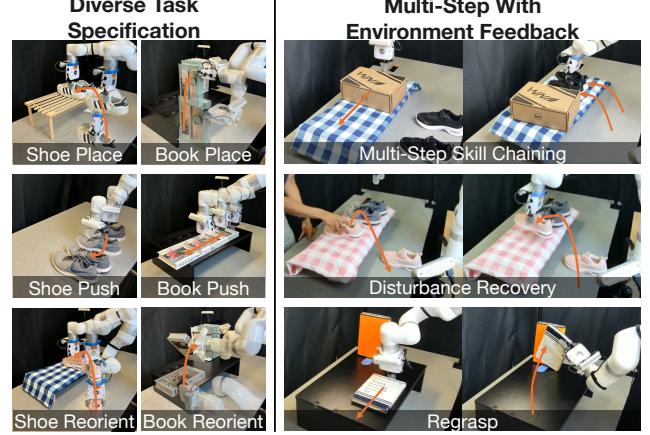


Fig. 1: **Capabilities of Our Framework.** IKER is designed to handle a wide range of real-world tasks. It can be seamlessly chained to execute multi-step tasks. It exhibits robustness to disturbances and demonstrates the ability to solve problems flexibly.

leverage VLMs in robotics face two major limitations: (1) they lack the capability to specify precise target locations in 3D, and (2) they are often unable to adapt to the environment changes as the task progresses.

In this work, we introduce **Iterative Keypoint Reward (IKER)**, a visually grounded reward function for robotic manipulation that addresses these limitations. Inspired by recent work [9], we draw the observation that both object positions and orientations can be encoded using keypoints. Hence, IKER allows for fine-grained manipulation in 3D, facilitating complex tasks that require accurate location and orientation control. Additionally, IKER incorporates an iterative refinement mechanism, where the VLM updates the task specification based on feedback from the robot’s interactions with the environment. This mechanism enables dynamically-adjusting strategies and intermediate steps, such as repositioning objects for a better grasp.

While VLMs excel in processing real-world visual data, training policies directly in the real world is often infeasible due to safety, scalability, and efficiency constraints. To address this, we first generate IKER using real-world observations, then transfer the scene and the reward to simulation for training, and finally, deploy the optimized policy back into the real world. Thus, our system operates in a real-to-sim-to-real loop.

We demonstrate the efficacy of our real-to-sim-to-real framework with IKER across diverse scenarios involving

*indicates equal contribution. ¹University of Illinois at Urbana-Champaign, ²Stanford University, ³Amazon, ⁴Columbia University

everyday objects like shoes and books. These include prehensile tasks, such as placing shoes on racks, and non-prehensile tasks, like sliding books to target locations. We conduct both quantitative and qualitative evaluations to assess the system’s ability to perform complex, long-horizon tasks autonomously. The results showcase human-like capabilities, including multi-step action sequencing, spontaneous error recovery, and the ability to update strategies in response to changes in the environment.

II. RELATED WORK

VLMs in Robotics. VLMs have become a prominent tool in robotics [10–38]. Existing works utilizing VLMs in robotics primarily focus on two areas: task specification [10–12, 15, 16, 21] and low-level control [11, 13, 14, 39]. Our work aligns with the former, with an emphasis on flexibility and adaptability in complex, real-world environments.

For task specification, many works employ VLMs to break down complex tasks into manageable subtasks, demonstrating their utility in bridging high-level instructions and robotic actions. Huang *et al.* [40] demonstrate the use of LLMs as zero-shot planners, enabling task decomposition into actionable steps. Similarly, Ahn *et al.* [10] leverage VLMs to parse long-horizon tasks and sequence them into executable steps for robots. Belkhale *et al.* [41] introduce “language motions” that serve as intermediaries between high-level instructions and specific robotic actions, allowing policies to capture reusable, low-level behaviors. Unlike these works, our approach focuses on flexible interpretation of tasks in the context of a dynamically changing environment.

Beyond task decomposition, VLMs have been used to generate affordances and value maps that guide robotic actions. Huang *et al.* [12] employs VLMs to generate 3D affordance maps, providing robots with spatial knowledge of which parts of the environment are suitable for interaction. Liu *et al.* [15] use VLMs to predict point-based affordances, enabling zero-shot manipulation tasks. Zhao *et al.* [42] incorporate VLMs into model predictive control, where the models predict the outcomes of candidate actions to guide optimal decision-making. These works demonstrate the potential of VLMs to bridge high-level task understanding with spatial and functional knowledge needed for robotic control. Similar to our work, Huang *et al.* [9] use keypoints and define relations and constraints between them to execute manipulation tasks, but their approach follows an open-loop strategy. In contrast, we employ a closed-loop approach, enabling dynamic plan adjustments. Additionally, our approach also supports non-prehensile manipulations, such as pushing.

Some works have also explored VLMs for reward function generation [43–46]. However, most of these approaches have limited real-world applicability. Some lack demonstrations on real robots [44], are restricted to a single real-world scenario [46], or focus on highly constrained tasks like a robot dog walking on a ball [45]. In contrast, our work demonstrates the versatility and robustness of VLM-generated rewards on multiple real-world manipulation tasks.

Algorithm 1 IKER Execution Framework

```

1: Given: Language instruction  $I$ 
2: Initialize:  $\text{done} \leftarrow \text{false}$ ,  $\text{execution.history} \leftarrow []$ ,  $i \leftarrow 1$ 
3: Generate 3D models of objects
4: while true do
5:    $(\{k_j^{(i)}\}_{j=1}^{K_i}, O_i) \leftarrow \text{GetKeypoints}(3D \text{ models})$ 
6:    $\text{code} \leftarrow \text{QueryVLM}(O_i, \text{execution.history})$ 
7:    $(\text{done}, \{k_j^{\text{target}(i)}\}_{j=1}^{K_i}) \leftarrow \text{Execute}(\text{code})$ 
8:   if  $\text{done}$  is true then
9:     break
10:  end if
11:   $s_i \leftarrow \text{TransferSceneToSimulation}(3D \text{ models})$ 
12:   $\pi_i \leftarrow \text{LearnPolicy}(s_i, \{k_j^{\text{target}(i)}\}_{j=1}^{K_i})$ 
13:   $\text{ExecutePolicyInRealWorld}(\pi_i)$ 
14:  Append  $(O_i, \text{code})$  to  $\text{execution.history}$ 
15:   $i \leftarrow i + 1$ 
16: end while

```

Real-to-Sim and Sim-to-Real. Real-to-sim has gained significant attention for its ability to facilitate agent training. Once a scene is transferred to simulation, it can be used for a wide range of tasks, including RL. Several approaches focus on reconstructing rigid bodies for use in simulation [47–53]. For instance, Kappler *et al.* [47] introduce a method for reconstructing rigid objects to facilitate grasping. Some works rather focus on reconstructing articulated objects [54–61]. Huang *et al.* [62] present methods for reconstructing the occluded shapes of articulated objects. Jiang *et al.* [55] introduce a framework, DITTO, to generate digital twins of articulated objects from real-world interactions. In our work, we utilize the fast state-of-the-art BundleSDF method [63] to generate object meshes that are transferred to the simulation.

Sim-to-real transfer has shown great performance in a variety of skills, including tabletop manipulation [64, 65], mobile manipulation [66, 67], dynamic manipulation [68], dexterous manipulation [69–72], and locomotion [73, 74]. However, directly deploying learned policies to physical robots cannot guarantee successful performance due to the sim-to-real gap. To bridge this gap, researchers have developed many techniques, such as system identification [75–77], domain adaptation [78–82], and domain randomization [73, 83, 84, 84–88]. In our work, we use domain randomization as it does not require any interaction data from the real world during training. It relies entirely on simulation and makes policies robust by exposing them to a wide variety of randomized conditions. Recently, Torne *et al.* [89] proposed RialTo, a complete real-to-sim-to-real loop system that focuses on leveraging simulation to robustify imitation learning policies trained using real-world collected demonstrations. In contrast, we focus on executing long-horizon tasks by training only in simulation, bypassing the need for demonstrations.

III. METHOD

In this section we formally define Iterative Keypoint Reward (IKER) and discuss how it is automatically synthesized and refined by VLMs by continuously taking in environmental feedback. Then, we discuss our overall framework, which uses IKER in a real-to-sim-to-real loop. Our method overview is illustrated in Figure 2, with detailed steps provided in Algorithm 1.

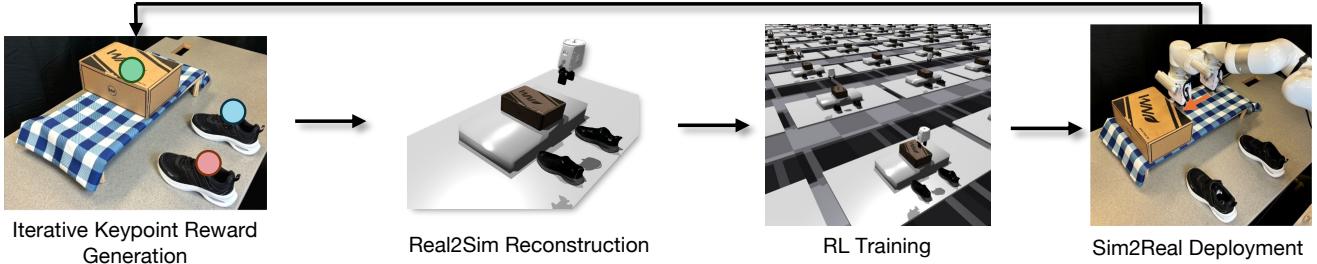


Fig. 2: **Framework Overview.** Iterative Keypoint Reward (IKER) is a visually grounded reward generated by Vision-Language Models (VLMs) as task specification. The framework reconstructs the real-world scene in simulation, and the generated reward is used to train RL policies, which are subsequently deployed in the real-world.

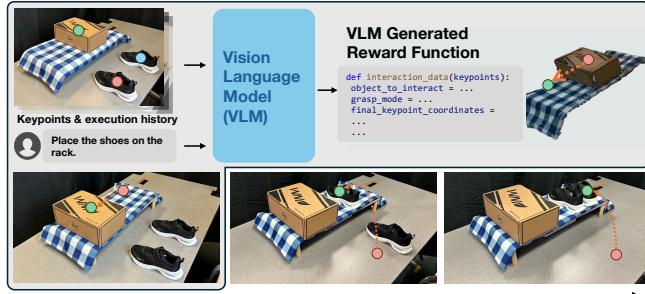


Fig. 3: **Iterative Keypoint Reward Generation.** This corresponds to the first step in Figure 2. We first obtain keypoints in the scene. These keypoints, combined with a human command and execution history, are processed by a VLM to generate code that maps keypoints to the reward function. A more detailed illustration of the keypoints and generated code is provided in Figure 7.

A. Iterative Keypoint Reward (IKER)

Given an RGB-D observation of the environment and an open-vocabulary language instruction I for a multi-step task, our goal is to obtain a sequence of policies, $\pi_{i=1}^N$, that complete the task. Crucially, the number of policies N is not predefined, allowing for flexibility in how the robot approaches the task. For example, in the scenario of Fig. 3, the first policy, π_1 , moves the shoe box to create space, while subsequent policies handle the placement of each shoe.

For each step i , we denote the RGB observation as O_i . We assume a set of K_i keypoints $\{k_j^{(i)}\}_{j=1}^{K_i}$ is given (discussed later in Sec. III-B), each specifying a 3D position in the task space. Using these keypoints, our objective is to automatically generate a reward function, termed IKER, that maps the keypoint positions to a scalar reward $f^{(i)} : \mathbb{R}^{K_i \times 3} \rightarrow \mathbb{R}$.

To generate the reward function $f^{(i)}$, we use a VLM (GPT-4o [1] in our case), which is provided with the context comprising (1) the human instruction I describing the task, (2) the current RGB observation O_i with keypoints overlaid with numerical markers, and (3) the sequence of previous observations and reward functions up to step $i-1$, i.e. $\{O_1, f^{(1)}, \dots, O_{i-1}, f^{(i-1)}\}$.

Additionally, the VLM is guided by a prompt that instructs it to generate a Python function for the reward $f^{(i)}$. The prompt directs the VLM to break down the task into executable steps, predict which object to interact with, specify the movement of objects by indicating where their keypoints should be placed relative to other keypoints, and perform arithmetic calculations on these keypoints to predict their final locations. We do not explicitly specify which keypoint

belongs to which object, allowing the VLM to infer this information. The prompt also instructs the VLM to present all outputs in a prescribed code format and set the flag $done = True$ if the task is completed. By predicting code, the VLM can perform arbitrary and precise calculations using the current keypoint locations, which would not be possible if limited to raw text. Please refer to Appendix C for the complete prompt, and Figure 7 for a step-by-step walkthrough of RGB observations and generated reward functions for the example of Figure 3.

Upon receiving the final keypoint locations by executing the generated code, we compute a scalar reward to evaluate the policy's performance. The reward function, $f^{(i)}$, facilitates learning by combining the following terms:

- Gripper-to-object Distance Reward (r_{dist}): Encourages the robot to approach the object of interest by penalizing large distances between them.
- Direction Reward (r_{dir}): Guides the robot to move the keypoints in the direction of the target locations.
- Alignment Reward (r_{align}): Drives the robot to position the keypoints close to their target locations.
- Success Bonus (r_{bonus}): Provides an additional reward when the average distance between the keypoints and their target positions remains within a specified threshold for a certain number of timesteps, indicating successful task completion.
- Penalty Term (r_{penalty}): Applies penalties for undesirable actions such as excessive movements, dropping the object, or applying excessive force.

$$f^{(i)} = \alpha_{\text{dist}} r_{\text{dist}} + \alpha_{\text{dir}} r_{\text{dir}} + \alpha_{\text{align}} r_{\text{align}} + \alpha_{\text{bonus}} r_{\text{bonus}} + \alpha_{\text{penalty}} r_{\text{penalty}}$$

B. Transferring real-world scene to simulation

To transfer the real-world scene within the workspace boundary to simulation, we first generate 3D meshes of manipulable objects, such as the shoe box and shoes shown in Figure 3, by capturing video footage of each object as it is moved to ensure the camera captures all sides. These videos allow for accurate 3D mesh reconstruction using BundleSDF [63], and multiple objects can be processed in parallel to speed up the scanning phase. Once a mesh is created for an object, it can be reused in different settings, eliminating the need to recreate it for each new scenario. With the meshes prepared, we use FoundationPose [90] to

estimate the objects' poses, enabling precise placement of the corresponding meshes in the simulated environment. For static elements, like the workspace table and shoe rack in Figure 3, we capture a point cloud to create their meshes for use in the simulation.

The generated meshes are further used to identify candidate keypoints. For manipulable objects like shoes or books, keypoints are placed at the object's extremities along its axes, defined with respect to the object's center, independent of the human instruction. For static objects like shoe racks, which are part of the environment, keypoints are uniformly distributed across their surfaces. Numerical labels assigned to keypoints are grouped by objects. For example, as shown in Figure 7, keypoints 1–4 correspond to the box, 5–8 to the left shoe, 9–12 to the right shoe, and the remaining keypoints to the rack. Keypoints that are too close together in the image projection are removed. Specifically, background keypoints (e.g., like rack) near object keypoints are removed first. Among overlapping object keypoints, only the one with the lower numerical label is retained. Note, however, that the VLM is not explicitly told this information but has to infer the association between the keypoints and the objects based on the input image.

C. Policy Training in Simulation

We control the robot in the end-effector space, which has six degrees of freedom: three prismatic joints for movement along the x, y, and z axes, and three revolute joints for rotation. The gripper fingers remain closed by default, opening only when grasping objects. Refer to Appendix A for a detailed discussion on grasping.

State Space: The state space for our policy captures the essential information to execute the task. The input is a vector s_t consisting of the gripper's end-effector pose $(\mathbf{p}_e, \mathbf{q}_e) \in \mathbb{R}^7$, the pose of object currently being manipulated $(\mathbf{p}_o, \mathbf{q}_o) \in \mathbb{R}^7$, a set of object keypoints $\mathcal{K}_o = \{k_j^{(i)}\}_{j=1}^{K_i} \in \mathbb{R}^{K_i \times 3}$, and their corresponding target positions $\mathcal{K}_t = \{k_{t_j}^{(i)}\}_{j=1}^{K_i} \in \mathbb{R}^{K_i \times 3}$. \mathcal{K}_o is calculated by applying rigid body transformations to keypoints defined in the object's local coordinate frame, mapping them to their corresponding positions in the world frame. \mathcal{K}_t is derived from the reward function $f^{(i)}$ generated by the VLM. Rotations \mathbf{q}_e and \mathbf{q}_o are represented as quaternions. This state space $s_t = (\mathbf{p}_e, \mathbf{q}_e, \mathbf{p}_o, \mathbf{q}_o, \mathcal{K}_o, \mathcal{K}_t)$ captures essential information on objects of interest as well as the goal of the policy. Instead of incorporating raw RGBD data directly into the state space, object poses and keypoints are extracted from RGBD inputs using a vision-based pose estimation method, as detailed in Section III-D. This preprocessing step removes the necessity of including raw RGBD data in the policy.

Action Space: The action space is defined relative to the gripper's current position and orientation. The policy outputs actions $a_t = (\Delta \mathbf{p}_e, \Delta \mathbf{r}_e)$, where $\Delta \mathbf{p}_e \in \mathbb{R}^3$ and $\Delta \mathbf{r}_e \in \mathbb{R}^3$ specifies the changes in translation and rotation respectively.

Training Algorithm & Architecture: We train our policies using IsaacGym [91] simulator with the PPO [92] algorithm. We use an actor-critic architecture [93] with a shared

backbone. The network is a multi-layer perceptron (MLP) consisting of hidden layers with 256, 128, and 64 units, each followed by ELU [94] activation. Currently, it takes about 5 minutes to train per task, which can be prohibitive for certain applications. However, this training time can be reduced by increasing the number of parallel environments and utilizing more powerful GPUs.

Domain Randomization (DR): Recognizing the challenges inherent in transferring policies between the simulation and the real world, we employ DR to bridge the real-to-sim-to-real gaps. DR is applied to object properties like friction, mass, restitution, compliance, and geometry. We further randomize the object position, the gripper location, and the grasp within a range. We found these to be especially crucial for non-prehensile tasks like pushing. The specific parameter ranges are detailed in Appendix B, and the effectiveness of DR is evaluated in Section IV-E.

D. Deployment of Trained Policy

The trained RL policy π_i is deployed directly in the real world. Since the policy outputs the end-effector pose, we employ inverse kinematics to compute the joint angles at each timestep. The RL policy operates at 10Hz, producing action commands that are then clipped to ensure the end effector remains within the workspace limits. For keypoint tracking, we utilize FoundationPose [90] to estimate the object's pose. These pose estimates are subsequently used to compute the keypoint locations that are defined relative to the objects. When VLM predicts to grasp objects, we use AnyGrasp [95] to detect grasps in the real-world.

IV. EXPERIMENTS AND ANALYSIS

We aim to investigate whether Iterative Keypoint Reward can effectively represent reward functions for diverse manipulation skills within our IKER for real-to-sim-to-real pipeline. We also want to see whether our pipeline can perform multi-step tasks in dynamic environments by leveraging Iterative Keypoint Reward as feedback for replanning.

A. Experimental Setup, Metrics and Baselines

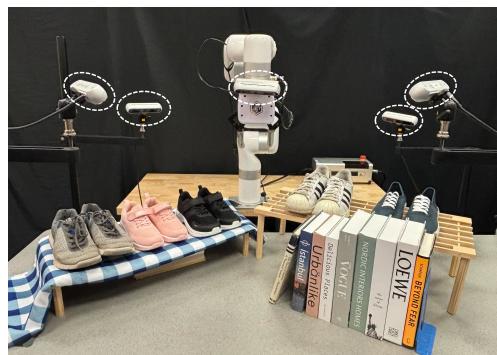


Fig. 4: **Setup and experiment objects.** We use XArm7 to conduct all our experiments. Our setup includes 4 stationary and 1 wrist-mounted camera. We experiment with 5 shoe pairs and 2 shoe racks for tasks involving shoe scenarios. Additionally, we experiment with 9 different books for stowing tasks.

Task	Annotated (Human labeled reward)				Automatic (VLM-generated reward)			
	Simulation		Real-World		Simulation		Real-World	
	IKER (Ours)	Pose	IKER (Ours)	Pose	IKER (Ours)	Pose	IKER (Ours)	Pose
Shoe Place	0.945	0.938	0.8	0.9	0.778	0.353	0.7	0.3
Shoe Push	0.871	0.850	0.7	0.7	0.716	0.289	0.6	0.2
Stowing Push	0.901	0.914	0.8	0.7	0.679	0.374	0.6	0.3
Stowing Reorient	0.848	0.859	0.8	0.7	0.858	0.265	0.7	0.2

TABLE I: **Performance of IKER in simulation and real-world.** IKER, which makes use of visual keypoints, significantly outperforms the conventional pose-based approach, especially when using VLMs to automatically generate reward functions.

We conduct experiments on XArm7 with four stationary RealSense cameras. Figure 4 shows the setup, along with the objects used. These cameras capture the point clouds, which are used to construct the simulation environment and to provide data for AnyGrasp to predict grasp. Additionally, a wrist-mounted camera is used to capture images that are used to query the VLM.

As a baseline, we use an *annotated* variant of IKER with human-labeled reward functions, allowing evaluation without VLM influence. We also compare our keypoint-based method with another baseline that uses object pose to construct reward function, which is more conventional in RL training [71, 96–99]. In this pose-based method, the VLM generates a function f that maps the initial object poses (represented by xyz coordinates for position and RPY angles for orientation) to their final poses. The prompt for this baseline is discussed in Sec. C.

We evaluate our approach across four scenarios, illustrated in Figure 1 (left): Shoe Place, Shoe Push, Book Push, and Book Reorient. In Shoe Place, the robot picks up a shoe from the ground and places it on a rack. In Shoe Push, it pushes a shoe towards other shoe to form a matching pair. In Book Push, it pushes a book to align with other book, or push the book towards table edge, and in Book Reorient, it repositions a book on a shelf. Each scenario has 10 start/end configurations. In simulation, success rates are averaged over 128 randomized environments generated for 10 start/end configuration, making a total of 1280 trials per scenario. In the real-world, success is evaluated directly on the 10 start/end configurations. A trial is considered successful in both cases if the average keypoint distance to the target is within 5 cm.

B. Policy Training with IKER for Single-Step Tasks

We conduct experiments comparing RL training with keypoints and object pose in reward functions. Our experiments span four representative tasks, and are summarized in Table I.

In the *annotated* method, success rates for shoe placement using IKER and object pose are 0.945 and 0.938, respectively. A similar trend is observed in the shoe push, stowing push, and reorient tasks, where performance differences are minimal. These results demonstrate that, when targets are specified through human annotations, both keypoints and object poses effectively capture the target locations and serve as viable approaches for RL policy training.

In the *automatic* method, IKER significantly outperforms object pose representations. For example, in shoe placement,

IKER achieves a 0.7 success rate, while object poses reach only 0.3. Similar results are seen across other tasks. Object pose success is limited to simpler scenarios with no orientation changes, as VLMs struggle with rotations in $SO(3)$ space. In contrast, keypoints simplify the challenge by requiring VLMs to reason only in Cartesian space, eliminating the need to handle object poses in $SE(3)$ space.

As shown in Table I, there is a slight reduction in success rate from simulation to the real world. For shoe placement, IKER achieves success of 0.945 in simulation and 0.8 in the real world. For shoe push, the success rate drops from 0.871 to 0.850. These results suggest that domain randomization described in Section III-C helps the model generalize to real-world conditions, but factors like inaccuracies in environment reconstruction, real-world perception errors, and the inability to simulate extreme object dynamics still affect performance.

Most of the failures in our framework stem from discrepancies between the heuristic grasps used in simulation and the grasps generated by AnyGrasp in the real world, as well as incorrect VLM predictions. For incorrect VLM predictions, the model sometimes selects the wrong keypoints or fails to use all available keypoints on an object when determining its relationship to another object. For instance, if an object has four keypoints, the VLM may only use one of them, leading to suboptimal alignment and placement. These issues can be mitigated by providing more in-context examples while querying the VLMs. These challenges may become less pronounced with the incorporation of advancements such as [100], which enhance the spatial reasoning capabilities of VLMs. Additionally, some failures are caused by physical dynamics when pushing objects. These issues can be partially mitigated by explicitly estimating dynamic parameters during real-to-sim transfer.

C. Iterative Replanning for Multi-Step Tasks

We demonstrate the robot’s iterative chaining ability with a task of three sequential actions: first pushing a shoe box to create space, then placing a pair of shoes on a rack. Failure in one task leads to failure in the next. We evaluate this process using 10 different start and end configurations, iterating through each to assess overall performance.

We compare our method with VoxPoser [12], which employs LLMs to generate code that produces potential fields for motion planning. VoxPoser serves as an ideal baseline because it synthesizes motion plans for diverse manipulation tasks from free-form language instructions. Notably, VoxPoser plans are open-loop and lack feedback to

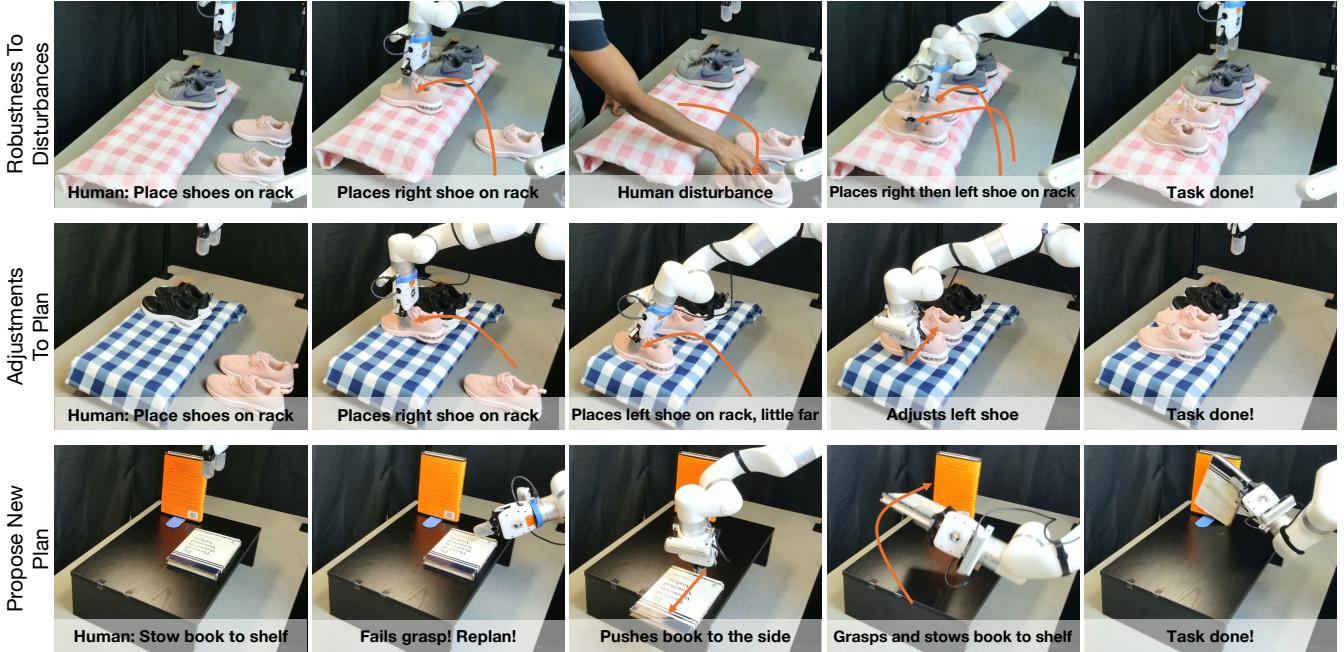


Fig. 5: **Scenarios demonstrating capabilities of our framework.** The framework is robust to disturbances and can adapt in response to unexpected events. Additionally, it can propose new plans when the original ones become infeasible.



Fig. 6: **Multi-Step Task Chaining Comparison with VoxPoser.** Our proposed framework consistently demonstrates superior performance compared to VoxPoser at every step of the task sequence.

refine specifications at each step. To adapt it to our tasks, we enhanced VoxPoser with two major modifications: (1) VoxPoser used OWL-ViT [101] to find object bounding boxes, but it struggled to distinguish between left and right shoes, so we provided ground-truth object locations. (2) We gave VoxPoser the entire plan, as the original planner struggled with multi-step tasks. This gave VoxPoser an advantage over our method due to access to privileged information.

Figure 6 shows the iterative chaining results. Across the three tasks, our method consistently outperformed VoxPoser. In the first task, we succeeded 8 out of 10 times compared to VoxPoser’s 5 successes. For the second task, we had 5 successes while VoxPoser had 1. In the final task, our method succeeded 4 times, whereas VoxPoser failed in all attempts. VoxPoser’s failures can be attributed to several factors, such as pushing the shoe box either too far or not far enough. Additionally, its grasping strategy relies on a simple heuristic that positions the robot’s end effector around the object center before closing the gripper, often resulting in failed grasps. It also struggles with collisions during object manipulation,

as it does not account for the environment to avoid obstacles. Furthermore, improper placement of shoes—such as stacking both shoes on top of each other, causing them to fall—further highlights its limitations.

D. Robustness, Adjusting Plans, and Re-Planning

Unlike previous works that rely on open-loop plans, our approach leverages closed-loop plans, enabling adjustments during execution. This feature gives rise to several capabilities, as demonstrated in Figure 5.

In the first scenario, a human interrupts the robot while it is in the process of placing shoes on the ground. The framework demonstrates resilience by recovering from the interruption. The robot re-grasps the shoe and successfully completes the task by placing both shoes on the rack.

In the second scenario, when the robot attempts to place the left shoe, it detects that the shoe is not positioned close enough to the right shoe. To address this, the VLM predicts a corrective action, suggesting that the robot push the left shoe closer to the right shoe to form a proper pair.

In the third scenario, the robot is tasked with stowing a book on a shelf. However, the initial grasp attempt fails because the book is too large to be grasped. In response, the VLM predicts an alternative strategy to complete the task, adjusting the approach to ensure successful placement.

E. Effect of Domain Randomization

We present the results of our framework with and without DR for shoe place and push. The performance is averaged over 10 runs. In the simulation, the performance without DR is 0.964, while with DR, it is slightly lower at 0.945, suggesting that without DR, the policy performs better in a single, controlled setting. However, in the real world, the performance without DR drops to 0.6, whereas with DR, it is

Task	Without DR		With DR	
	Simulation	Real-World	Simulation	Real-World
Place	0.964	0.5	0.945	0.8
Push	0.923	0.2	0.871	0.7

TABLE II: **Performance of Shoe Place and Push with and without Domain Randomization (DR).** DR slightly reduces simulation performance but significantly improves real-world task performance across different scenarios.

0.8, highlighting the effectiveness of DR. For the place task without DR, we observe that the policy is less robust to sim-to-real gap, frequently colliding with the shoe rack during transport. Additionally, immediately after picking up the object, the policy sometimes fails, likely due to discrepancies in the pose estimation.

For push, these issues are more pronounced: success is 0.2 without DR but improves to 0.7 with DR. Without DR, the policy often crushes the shoe or causes it to slip out of alignment during pushing. These findings demonstrate the importance of DR for reliable real-world performance.

V. CONCLUSION AND LIMITATIONS

In this work, we introduced Iterative Keypoint Reward (IKER), a framework that leverages VLMs to generate visually grounded reward functions for robotic manipulation in open-world environments. By using keypoints from RGB-D observations, our approach enables precise SE(3) control and integrates priors from VLMs without relying on rigid instructions. IKER bridges simulation and real-world execution through a real-to-sim-to-real loop, training policies in simulation and deploying them in physical environments. Experiments across diverse tasks demonstrate the framework’s ability to handle complex, long-horizon challenges with adaptive strategies and error recovery. This work represents a step toward more intelligent and flexible robots capable of operating effectively in dynamic, real-world settings.

Despite these advancements, our approach has certain limitations. We need to capture objects from all views to obtain object meshes. In the future, this may be simplified by using methods [52] that can generate meshes from a single image. Additionally, our real-to-sim transfer does not account for dynamics parameters, which could be modeled more accurately through system identification techniques. Also, while our framework reconstructs multiple objects in the environment, we do not account for tasks involving complicated multi-object interactions, limiting our evaluation primarily to single-object manipulation at each stage.

VI. ACKNOWLEDGEMENTS

We thank Aditya Prakash, Arjun Gupta, Binghao Huang, Hanxiao Jiang, Kaifeng Zhang, and Unnat Jain for fruitful discussions. This work is partially supported by the Amazon AICE Award and the Sony Group Corporation. This work does not relate to the positions of Shubham Garg and Hooshang Nayyeri at Amazon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.

REFERENCES

- [1] OpenAI, “Gpt-4 technical report,” *arXiv*, 2023.
- [2] A. Zeng, A. Wong, S. Welker, K. Choromanski, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke *et al.*, “Socratic models: Composing zero-shot multimodal reasoning with language,” *arXiv preprint arXiv:2204.00598*, 2022.
- [3] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, “Learning transferable visual models from natural language supervision,” 2021. [Online]. Available: <https://arxiv.org/abs/2103.00020>
- [4] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. V. Le, Y. Sung, Z. Li, and T. Duerig, “Scaling up visual and vision-language representation learning with noisy text supervision,” 2021. [Online]. Available: <https://arxiv.org/abs/2102.05918>
- [5] J. Li, D. Li, C. Xiong, and S. Hoi, “Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation,” in *International conference on machine learning*. PMLR, 2022, pp. 12 888–12 900.
- [6] J. Li, D. Li, S. Savarese, and S. Hoi, “Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models,” 2023. [Online]. Available: <https://arxiv.org/abs/2301.12597>
- [7] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan, “Flamingo: a visual language model for few-shot learning,” 2022. [Online]. Available: <https://arxiv.org/abs/2204.14198>
- [8] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, “Coca: Contrastive captioners are image-text foundation models,” 2022. [Online]. Available: <https://arxiv.org/abs/2205.01917>
- [9] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, “Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation,” *arXiv preprint arXiv:2409.01652*, 2024.
- [10] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, “Do as i can, not as i say: Grounding language in robotic affordances,” *arXiv preprint arXiv:2204.01691*, 2022.
- [11] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, “Code as policies: Language model programs for embodied control,” in *arXiv preprint arXiv:2209.07753*, 2022.
- [12] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, “Voxposer: Composable 3d value maps for robotic manipulation with language models,” *arXiv preprint arXiv:2307.05973*, 2023.
- [13] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu *et al.*, “Rt-1: Robotics transformer for real-world control at scale,” *arXiv preprint arXiv:2212.06817*, 2022.
- [14] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn *et al.*, “Rt-2: Vision-language-action models transfer web knowledge to robotic control,” *arXiv preprint arXiv:2307.15818*, 2023.
- [15] F. Liu, K. Fang, P. Abbeel, and S. Levine, “Moka: Open-vocabulary robotic manipulation through mark-based visual prompting,” *arXiv preprint arXiv:2403.03174*, 2024.
- [16] H. Huang, F. Lin, Y. Hu, S. Wang, and Y. Gao, “Copa: General robotic manipulation through spatial constraints of parts with foundation models,” *arXiv preprint arXiv:2403.08248*, 2024.
- [17] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu *et al.*, “Octo: An open-source generalist robot policy,” *arXiv preprint arXiv:2405.12213*, 2024.
- [18] S. Huang, Z. Jiang, H. Dong, Y. Qiao, P. Gao, and H. Li, “Instruct2act: Mapping multi-modality instructions to robotic actions with large language model,” *arXiv preprint arXiv:2305.11176*, 2023.
- [19] M. Xu, P. Huang, W. Yu, S. Liu, X. Zhang, Y. Niu, T. Zhang, F. Xia, J. Tan, and D. Zhao, “Creative robot tool use with large language models,” *arXiv preprint arXiv:2310.13065*, 2023.
- [20] H. Zhou, M. Ding, W. Peng, M. Tomizuka, L. Shao, and C. Gan, “Generalizable long-horizon manipulations with large language models,” *arXiv preprint arXiv:2310.02264*, 2023.
- [21] S. Nasiriany, F. Xia, W. Yu, T. Xiao, J. Liang, I. Dasgupta, A. Xie, D. Driess, A. Wahid, Z. Xu *et al.*, “Pivot: Iterative visual prompting elicits actionable knowledge for vlm,” *arXiv preprint arXiv:2402.07872*, 2024.

[22] N. Di Palo and E. Johns, “Keypoint action tokens enable in-context imitation learning in robotics,” *arXiv preprint arXiv:2403.19578*, 2024.

[23] F. Zeng, W. Gan, Y. Wang, N. Liu, and P. S. Yu, “Large language models for robotics: A survey,” *arXiv preprint arXiv:2311.07226*, 2023.

[24] L. Zha, Y. Cui, L.-H. Lin, M. Kwon, M. G. Arenas, A. Zeng, F. Xia, and D. Sadigh, “Distilling and retrieving generalizable knowledge for robot manipulation via language corrections,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 15 172–15 179.

[25] M. G. Arenas, T. Xiao, S. Singh, V. Jain, A. Ren, Q. Vuong, J. Varley, A. Herzog, I. Leal, S. Kirmani *et al.*, “How to prompt your robot: A promptbook for manipulation skills with code as policies,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 4340–4348.

[26] K. Mahadevan, J. Chien, N. Brown, Z. Xu, C. Parada, F. Xia, A. Zeng, L. Takayama, and D. Sadigh, “Generative expressive robot behaviors using large language models,” in *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, 2024, pp. 482–491.

[27] J. Liang, F. Xia, W. Yu, A. Zeng, M. G. Arenas, M. Attarian, M. Bauza, M. Bennice, A. Bewley, A. Dostrmohamed *et al.*, “Learning to learn faster from human feedback with language model predictive control,” *arXiv preprint arXiv:2402.11450*, 2024.

[28] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman *et al.*, “Grounded decoding: Guiding text generation with grounded models for embodied agents,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[29] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley *et al.*, “Robots that ask for help: Uncertainty alignment for large language model planners,” *arXiv preprint arXiv:2307.01928*, 2023.

[30] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, “Vima: General robot manipulation with multimodal prompts,” *arXiv preprint arXiv:2210.03094*, vol. 2, no. 3, p. 6, 2022.

[31] Z. Yang, C. Garrett, D. Fox, T. Lozano-Pérez, and L. P. Kaelbling, “Guiding long-horizon task and motion planning with vision language models,” *arXiv preprint arXiv:2410.02193*, 2024.

[32] J. Duan, W. Pumacay, N. Kumar, Y. R. Wang, S. Tian, W. Yuan, R. Krishna, D. Fox, A. Mandlekar, and Y. Guo, “Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation,” *arXiv preprint arXiv:2410.00371*, 2024.

[33] J. Duan, W. Yuan, W. Pumacay, Y. R. Wang, K. Ehsani, D. Fox, and R. Krishna, “Manipulate-anything: Automating real-world robots using vision-language models,” *arXiv preprint arXiv:2406.18915*, 2024.

[34] W. Yuan, J. Duan, V. Blukis, W. Pumacay, R. Krishna, A. Murali, A. Mousavian, and D. Fox, “Robopoint: A vision-language model for spatial affordance prediction for robotics,” *arXiv preprint arXiv:2406.10721*, 2024.

[35] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, “Progrompt: program generation for situated robot task planning using large language models,” *Autonomous Robots*, vol. 47, no. 8, pp. 999–1012, 2023.

[36] G. Tang, S. Rajkumar, Y. Zhou, H. R. Walke, S. Levine, and K. Fang, “Kalie: Fine-tuning vision-language models for open-world manipulation without robot data,” *arXiv preprint arXiv:2409.14066*, 2024.

[37] W. Liang, S. Wang, H.-J. Wang, O. Bastani, D. Jayaraman, and Y. J. Ma, “Eurekaverse: Environment curriculum generation via large language models,” *arXiv preprint arXiv:2411.01775*, 2024.

[38] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine, “Robotic control via embodied chain-of-thought reasoning,” *arXiv preprint arXiv:2407.08693*, 2024.

[39] A. O’Neill, A. Rehman, A. Gupta, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar *et al.*, “Open x-embodiment: Robotic learning datasets and rt-x models,” *arXiv preprint arXiv:2310.08864*, 2023.

[40] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, “Language models as zero-shot planners: Extracting actionable knowledge for embodied agents,” in *International Conference on Machine Learning*. PMLR, 2022.

[41] S. Belkhale, T. Ding, T. Xiao, P. Sermanet, Q. Vuong, J. Tompson, Y. Chebotar, D. Dwibedi, and D. Sadigh, “Rt-h: Action hierarchies using language,” in <https://arxiv.org/abs/2403.01823>, 2024.

[42] W. Zhao, J. Chen, Z. Meng, D. Mao, R. Song, and W. Zhang, “Vlmpc: Vision-language model predictive control for robotic manipulation,” in *Robotics: Science and Systems*, 2024.

[43] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik *et al.*, “Language to rewards for robotic skill synthesis,” *arXiv preprint arXiv:2306.08647*, 2023.

[44] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, “Eureka: Human-level reward design via coding large language models,” *arXiv preprint arXiv: Arxiv-2310.12931*, 2023.

[45] Y. J. Ma, W. Liang, H.-J. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman, “Dreureka: Language model guided sim-to-real transfer,” 2024. [Online]. Available: <https://arxiv.org/abs/2406.01967>

[46] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu, “TextReward: Automated dense reward function generation for reinforcement learning,” *arXiv preprint arXiv:2309.11489*, 2023.

[47] D. Kappler, F. Meier, J. Issac, J. Mainprice, C. G. Cifuentes, M. Wüthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg, “Real-time perception meets reactive motion generation,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1864–1871, 2018.

[48] B. Wen, W. Lian, K. Bekris, and S. Schaal, “You only demonstrate once: Category-level manipulation from single visual demonstration,” *arXiv preprint arXiv:2201.12716*, 2022.

[49] M. Liu, R. Shi, L. Chen, Z. Zhang, C. Xu, X. Wei, H. Chen, C. Zeng, J. Gu, and H. Su, “One-2-3-45++: Fast single image to 3d objects with consistent multi-view generation and 3d diffusion,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 10 072–10 083.

[50] C. Xu, A. Li, L. Chen, Y. Liu, R. Shi, H. Su, and M. Liu, “Sparp: Fast 3d object reconstruction and pose estimation from sparse views,” in *European Conference on Computer Vision*. Springer, 2025, pp. 143–163.

[51] R. Shi, H. Chen, Z. Zhang, M. Liu, C. Xu, X. Wei, L. Chen, C. Zeng, and H. Su, “Zero123++: a single image to consistent multi-view diffusion base model,” *arXiv preprint arXiv:2310.15110*, 2023.

[52] R. Liu, R. Wu, B. Van Hoorick, P. Tokmakov, S. Zakharov, and C. Vondrick, “Zero-1-to-3: Zero-shot one image to 3d object,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 9298–9309.

[53] J. Gao, T. Shen, Z. Wang, W. Chen, K. Yin, D. Li, O. Litany, Z. Gojcic, and S. Fidler, “Get3d: A generative model of high quality 3d textured shapes learned from images,” *Advances In Neural Information Processing Systems*, vol. 35, pp. 31 841–31 854, 2022.

[54] J. Mu, W. Qiu, A. Kortylewski, A. Yuille, N. Vasconcelos, and X. Wang, “A-sdf: Learning disentangled signed distance functions for articulated shape representation,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 001–13 011.

[55] Z. Jiang, C.-C. Hsu, and Y. Zhu, “Ditto: Building digital twins of articulated objects from interaction,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5616–5626.

[56] N. Nie, S. Y. Gadre, K. Ehsani, and S. Song, “Structure from action: Learning interactions for articulated object 3d structure discovery,” *arXiv preprint arXiv:2207.08997*, 2022.

[57] Z. Chen, A. Walsman, M. Memmel, K. Mo, A. Fang, K. Vemuri, A. Wu, D. Fox, and A. Gupta, “Urdformer: A pipeline for constructing articulated simulation environments from real-world images,” *arXiv preprint arXiv:2405.11656*, 2024.

[58] Z. Mandi, Y. Weng, D. Bauer, and S. Song, “Real2code: Reconstruct articulated objects via code generation,” *arXiv preprint arXiv:2406.08474*, 2024.

[59] J. Liu, A. Mahdavi-Amiri, and M. Savva, “Paris: Part-level reconstruction and motion analysis for articulated objects,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 352–363.

[60] J. Liu, H. I. I. Tam, A. Mahdavi-Amiri, and M. Savva, “Cage: Controllable articulation generation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 17 880–17 889.

[61] J. Liu, D. Iliash, A. X. Chang, M. Savva, and A. Mahdavi-Amiri, “Singapo: Single image controlled generation of articulated parts in object,” *arXiv preprint arXiv:2410.16499*, 2024.

[62] X. Huang, I. Walker, and S. Birchfield, “Occlusion-aware reconstruction and manipulation of 3d articulated objects,” in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 1365–1371.

[63] B. Wen, J. Tremblay, V. Blukis, S. Tyree, T. Muller, A. Evans, D. Fox, J. Kautz, and S. Birchfield, “Bundlesdf: Neural 6-dof tracking and 3d reconstruction of unknown objects,” *CVPR*, 2023.

[64] M. Shridhar, L. Manuelli, and D. Fox, “Cliprot: What and where pathways for robotic manipulation,” *arXiv preprint arXiv: Arxiv-2109.12098*, 2021.

[65] Y. Jiang, C. Wang, R. Zhang, J. Wu, and L. Fei-Fei, “Transic: Sim-to-real policy transfer by learning from online correction,” 2024. [Online]. Available: <https://arxiv.org/abs/2405.10315>

[66] J. Gu, D. S. Chaplot, H. Su, and J. Malik, “Multi-skill mobile manipulation for object rearrangement,” *arXiv preprint arXiv: Arxiv-2209.02778*, 2022.

[67] S. Yenamandra, A. Ramachandran, K. Yadav, A. Wang, M. Khanna, T. Gervet, T.-Y. Yang, V. Jain, A. W. Clegg, J. Turner, Z. Kira, M. Savva, A. Chang, D. S. Chaplot, D. Batra, R. Mottaghi, Y. Bisk, and C. Paxton, “Homerobot: Open-vocabulary mobile manipulation,” *arXiv preprint arXiv: Arxiv-2306.11565*, 2023.

[68] B. Huang, Y. Chen, T. Wang, Y. Qin, Y. Yang, N. Atanasov, and X. Wang, “Dynamic handover: Throw and catch with bimanual hands,” *arXiv preprint arXiv:2309.05655*, 2023.

[69] Y. Chen, C. Wang, L. Fei-Fei, and C. K. Liu, “Sequential dexterity: Chaining dexterous policies for long-horizon manipulation,” *arXiv preprint arXiv:2309.00987*, 2023.

[70] Y. Qin, B. Huang, Z.-H. Yin, H. Su, and X. Wang, “Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation,” 2022. [Online]. Available: <https://arxiv.org/abs/2211.09423>

[71] H. Qi, A. Kumar, R. Calandra, Y. Ma, and J. Malik, “In-hand object rotation via rapid motor adaptation,” in *Conference on Robot Learning*. PMLR, 2023, pp. 1722–1732.

[72] Z.-H. Yin, B. Huang, Y. Qin, Q. Chen, and X. Wang, “Rotating without seeing: Towards in-hand dexterity through touch,” *arXiv preprint arXiv:2303.10880*, 2023.

[73] A. Kumar, Z. Fu, D. Pathak, and J. Malik, “RMA: rapid motor adaptation for legged robots,” in *Robotics: Science and Systems XVII, Virtual Event, July 12-16, 2021*, D. A. Shell, M. Toussaint, and M. A. Hsieh, Eds., 2021. [Online]. Available: <https://doi.org/10.15607/RSS.2021.XVII.011>

[74] T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi, “Agile but safe: Learning collision-free high-speed legged locomotion,” 2024. [Online]. Available: <https://arxiv.org/abs/2401.17583>

[75] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, “Sim-to-real: Learning agile locomotion for quadruped robots,” *arXiv preprint arXiv: Arxiv-1804.10332*, 2018.

[76] P. Chang and T. Padir, “Sim2real2sim: Bridging the gap between simulation and real-world in flexible object manipulation,” *arXiv preprint arXiv: Arxiv-2002.02538*, 2020.

[77] V. Lim, H. Huang, L. Y. Chen, J. Wang, J. Ichnowski, D. Seita, M. Laskey, and K. Goldberg, “Planar robot casting with real2sim2real self-supervised learning,” *arXiv preprint arXiv: Arxiv-2111.04814*, 2021.

[78] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, “Using simulation and domain adaptation to improve efficiency of deep robotic grasping,” in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[79] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki, “Meta reinforcement learning for sim-to-real domain adaptation,” 2019. [Online]. Available: <https://arxiv.org/abs/1909.12906>

[80] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, “Rl-cyclegan: Reinforcement learning aware simulation-to-real,” 2020. [Online]. Available: <https://arxiv.org/abs/2006.09001>

[81] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, “Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 627–12 637.

[82] Y. Du, D. Ho, A. Alemi, E. Jang, and M. Khansari, “Bayesian imitation learning for end-to-end mobile manipulation,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 5531–5546.

[83] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, “Solving rubik’s cube with a robot hand,” *arXiv preprint arXiv: Arxiv-1910.07113*, 2019.

[84] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[85] R. Antonova, F. Ramos, R. Possas, and D. Fox, “Bayessim: Scalable parameter inference for adaptive domain randomization with isaacgym,” *arXiv preprint arXiv:2107.04527*, 2021.

[86] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning quadruped locomotion over challenging terrain,” *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[87] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Sim-to-real transfer of robotic control with dynamics randomization,” in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[88] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, “Closing the sim-to-real loop: Adapting simulation randomization with real world experience,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8973–8979.

[89] M. Torne, A. Simeonov, Z. Li, A. Chan, T. Chen, A. Gupta, and P. Agrawal, “Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation,” *arXiv preprint arXiv:2403.03949*, 2024.

[90] B. Wen, W. Yang, J. Kautz, and S. Birchfield, “Foundationpose: Unified 6d pose estimation and tracking of novel objects,” *arXiv preprint arXiv:2312.08344*, 2023.

[91] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, “Isaac gym: High performance gpu-based physics simulation for robot learning,” 2021.

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

[93] V. Konda and J. Tsitsiklis, “Actor-critic algorithms,” *Advances in neural information processing systems*, vol. 12, 1999.

[94] D.-A. Clevert, “Fast and accurate deep network learning by exponential linear units (elus),” *arXiv preprint arXiv:1511.07289*, 2015.

[95] H.-S. Fang, C. Wang, H. Fang, M. Gou, J. Liu, H. Yan, W. Liu, Y. Xie, and C. Lu, “Anygrasp: Robust and efficient grasp perception in spatial and temporal domains,” *IEEE Transactions on Robotics*, 2023.

[96] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[97] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, “Data-efficient deep reinforcement learning for dexterous manipulation,” *arXiv preprint arXiv:1704.03073*, 2017.

[98] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, “Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards,” *arXiv preprint arXiv:1707.08817*, 2017.

[99] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations,” *arXiv preprint arXiv:1709.10087*, 2017.

[100] B. Chen, Z. Xu, S. Kirmani, B. Ichter, D. Sadigh, L. Guibas, and F. Xia, “Spatialvlm: Endowing vision-language models with spatial reasoning capabilities,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14 455–14 465.

[101] M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy, A. Mahendran, A. Arnab, M. Dehghani, Z. Shen *et al.*, “Simple open-vocabulary object detection with vision transformers,” *arXiv preprint arXiv:2205.06230*, 2022.

APPENDIX

A. Grasping Subroutine

During training, the gripper fingers open only in the *grasp* mode, where the end-effector approaches the object with open fingers and then closes them to grasp the object. We employ a heuristic-based grasp for faster training. In real-world, the gripper fingers remain closed until the grasp mode is triggered. AnyGrasp predicts an appropriate grasp pose and the fingers close at the predicted position. To address the sim-to-real gap, we add randomization to the heuristic grasp pose during simulation. This allows the policy to generalize more effectively, resulting in more robust and reliable policies in the real-world.

B. Domain Randomization Parameters

To enhance the robustness of our policies for effective real-to-sim-to-real transfer, we apply domain randomization to various object properties and initial conditions. Table III details the key randomized parameters and their respective ranges. These variations ensure that our learned policies generalize effectively to real-world conditions, mitigating the discrepancies between simulation and real-world.

Parameter	Range
Object Scale	[0.8, 1.2]
Mass	[0.3, 2.0]
Friction	[0.3, 1.8]
Restitution	[0.0, 1.0]
Compliance	[0.0, 1.0]
Center of Mass Perturbation	[-0.05, 0.05]
Initial Position Perturbation	[-0.02, 0.02]
Initial Orientation Perturbation	[-0.05, 0.05]
Grasp Position Noise	[-0.01, 0.01]
Grasp Orientation Noise	[-0.2, 0.2]

TABLE III: Domain randomization ranges for key object properties and initial conditions in simulation.

C. VLM Prompts

The VLM receives the image overlaid with keypoints $1, \dots, K$, along with the task description as text. These are given to the VLM, along with the prompt. We do not provide any in-context examples with the prompt. Our prompt for single-step tasks is as follows:

```
## Instructions
Your job is to help with moving rigid objects in real-world by writing code in python.
The task is given as an image of the environment, overlaid with keypoints marked with their indices, along with a text instruction.
These keypoints are in 3D space, and are projected onto the 2D image. They are attached with the objects, and move along with them.
So to determine where a specific point should go, you should specify where its corresponding keypoint should go. The coordinate system is marked at the bottom right in the image, with a vertical arrow pointing forward in the positive x direction and the horizontal arrow pointing to the left in the positive y direction.
The code should predict the final keypoint locations relative to their matching keypoints. Use all matching keypoints to determine the final position of the moving object, not just a single reference point.
Note:
- You should determine if you need to grasp or push the object. You should output a boolean grasp_mode for that.
```

- Some objects should not be moved. Hence, the final location of key points on them should be the same as the initial locations.
- If you need to interact with an object, the final location of only the keypoints marked on it should change. Hence, you should first try to understand which object should move.
- You should try to understand where the moving object should go relative to other stationary objects and use all the matching keypoints for alignment. Then you can give the final locations of keypoints of moving objects relative to keypoints on stationary objects.
- Positive x direction points towards up and positive y direction points towards left.
- The input to the function is a dictionary of keypoint coordinates. So keys will be strings like "1", "2", ... and their values will be numpy arrays ([x, y, z]) representing the 3D location of the keypoint corresponding to that index.
- To represent coordinates relative to other keypoints, you can make predictions like:


```
keypoint_coordinates['1'] = keypoint_coordinates['2'] + np.array([delta_x, delta_y, delta_z]).
```

 For instance, if keypoint 1 needs to be placed to the left of keypoint 2, then $\delta_x = 0$, $\delta_y = 0.1$, and $\delta_z = 0$. So can predict $keypoint_coordinates['1'] = keypoint_coordinates['2'] + np.array([0, 0.1, 0])$. The units here are in meters and left direction corresponds to + y-axis.
- Make use of semantics. Some objects should be placed in a certain way, like a left shoe should be placed on the left of the right shoe.

Structure your output in a single python code block as follows:

```
def get_interaction_data(keypoint_coordinates):
    """ Put your explanation here. """
    object_to_interact = ?
    keypoint_indices_to_interact = ?
    grasp_mode = ?
    ## final keypoint calculation for each keypoint in
    keypoint_indices_to_interact
    keypoint_coordinates['keypoint_indices_to_interact
    [0]'] = ? # Write calculation here. You may use
    multiple lines
    # Repeat for other keypoints
    return object_to_interact,
    keypoint_indices_to_interact, grasp_mode,
    keypoint_coordinates

## Query
Query Task: '[TASK]'
Query Image: [IMAGE_WITH_KEYPOINTS]
```

The prompt for multi-step tasks is as follows:

```
## Instructions
Your job is to help with moving rigid objects in real-world by writing code in python.
The task is given as an image of the environment, overlaid with keypoints marked with their indices, along with a text instruction.
These keypoints are in 3D space, and are projected onto the 2D image. They are attached with the objects, and move along with them.
So to determine where a specific point should go, you should specify where its corresponding keypoint should go. The coordinate system is marked at the bottom right in the image, with a vertical arrow pointing forward in the positive x direction and the horizontal arrow pointing to the left in the positive y direction.
The code should predict the final keypoint locations relative to their matching keypoints. Use all matching keypoints to determine the final position of the moving object, not just a single reference point.
Note:
- You should determine if you need to grasp or push the object. You should output a boolean grasp_mode for that.
  Generally, big objects can only be pushed as they are too big to be grasped.
- Some objects should not be moved. Hence, the final location of key points on them should be the same as the initial locations.
```



Fig. 7: Examples of keypoint-marked images with corresponding predicted codes. The top row represents the starting point, with subsequent rows illustrating the progression step by step. The VLM first predicts to push the box to create space, followed by sequential placement of the shoes.

- If you need to interact with an object, the final location of only the keypoints marked on it should change. Hence, you should first try to understand which object should move.

- You should try to understand where the moving object should go relative to other stationary objects and use all the matching keypoints for alignment. Then you can give the final locations of keypoints of moving objects relative to keypoints on stationary objects.

- Positive x direction points towards up and positive y direction points towards left.

- The input to the function is a dictionary of keypoint coordinates. So keys will be strings like "1", "2", ... and their values will be numpy arrays ([x, y, z]) representing the 3D location of the keypoint corresponding to that index.

- To represent coordinates relative to other keypoints, you can make predictions like:

```
keypoint_coordinates['1'] = keypoint_coordinates['2'] + np.array([delta_x, delta_y, delta_z]).
```

For instance, if the keypoint 1 needs to be placed to the left of keypoint 2, then delta_x = 0, delta_y = 0.1, and delta_z = 0.

So can predict keypoint_coordinates['1'] = keypoint_coordinates['2'] + np.array([0, 0.1, 0]). The units here are in meters and left direction corresponds to + y-axis.

- Make use of semantics. Some objects should be placed in a certain way, like a left shoe should be placed on the left of the right shoe.

- Some tasks involve multiple stages, so you will also predict the overall plan. Then you will write the description and code for the current stage. You will interact with only one object in a stage. Placing or pushing a single object will be considered a single stage. Grasping is not considered as a separate stage.

- You are free to make minor changes to the plan, or change the plan altogether if you think is necessary.

- We will keep adding the previous states of the environment as images and the corresponding code to the description. This will show how the task progressed. At the start, you will only see the task description.

- You should predict done=True when the task is complete, otherwise False. Only predict done=True when you see that the task is completed.

Structure your output in a single python code block as

```

follows:

def get_interaction_data(keypoint_coordinates):
    """
        Previous Plan Description
        Current Plan Description
        Current stage description
    """
    done = ?
    if done:
        return
    object_to_interact = ?
    keypoint_indices_to_interact = ?
    grasp_mode = ?
    # final keypoint calculation for each keypoint in
    keypoint_indices_to_interact
    keypoint_coordinates['keypoint_indices_to_interact
    [0]'] = ? # Write calculation here. You may use
    multiple lines
    # Repeat for other keypoints
    return object_to_interact,
    keypoint_indices_to_interact, grasp_mode,
    keypoint_coordinates

## Query
Query Task: '[TASK]'
Query Image: [IMAGE_WITH_KEYPOINTS]

```

The prompt for baseline that uses pose input for single-step tasks is as follows:

```

## Instructions
Your job is to help with moving rigid objects in a real-world environment by writing code in Python.
The task is given as an image of the environment, along with text instructions.
The coordinate system is marked at the bottom right in the image, with a vertical arrow pointing forward in the positive x direction and the horizontal arrow pointing to the left in the positive y direction.
The objects are treated as rigid bodies and are labeled with numbers. You need to predict the final pose of the moving object relative to the pose of any object. It can be its own pose or pose of other objects in the image.
Note:
- You should determine if you need to grasp or push the object. You should output a boolean grasp_mode for that.
- Some objects should not be moved. Hence, the final pose of those objects will be the same as the initial pose.
- If you need to interact with an object, the final pose of that object should change.
- You should first try to understand which object should move relative to its pose, or poses of the other objects. Then you can give the final pose of the moving object relative to the other object poses.
- Positive x direction points upwards, and positive y direction points to the left.
- The input to the function is a dictionary of object poses. So the keys will be strings representing object labels like "1", "2", ..., and their values will be numpy arrays [x, y, z, r, p, y], where x, y, z represent the position in meters and r, p, y represent the orientation in radians.
- To represent poses relative to other objects, you can make predictions like: object_poses['1'] = object_poses['2'] + np.array([delta_x, delta_y, delta_z, delta_r, delta_p, delta_yaw]). For instance, if object 1 needs to be placed to the left of object 2, then: delta_x = 0, delta_y = 0.1, delta_z = 0, delta_r = 0, delta_p = 0, delta_yaw = 0. So, you can predict: object_poses['1'] = object_poses['2'] + np.array([0, 0.1, 0, 0, 0, 0]). The units for positions are in meters, and for orientations, they are in radians. The left direction corresponds to the positive y-axis.
- Make use of semantics. Some objects should be placed in a certain way, like a left shoe should be placed on the left of the right shoe.
Structure your output in a single Python code block as follows:

def get_final_poses(object_poses):
    """
        Put your explanation here. """
    object_to_interact = ?
    grasp_mode = ?

```

```

## final pose calculation for each object
object_poses['object_to_interact'] = ? # Write
calculation here. You may use multiple lines
return object_to_interact, grasp_mode, object_poses

## Query
Query Task: '[TASK]'
Query Image: [IMAGE]

```

D. Case study of a complex task

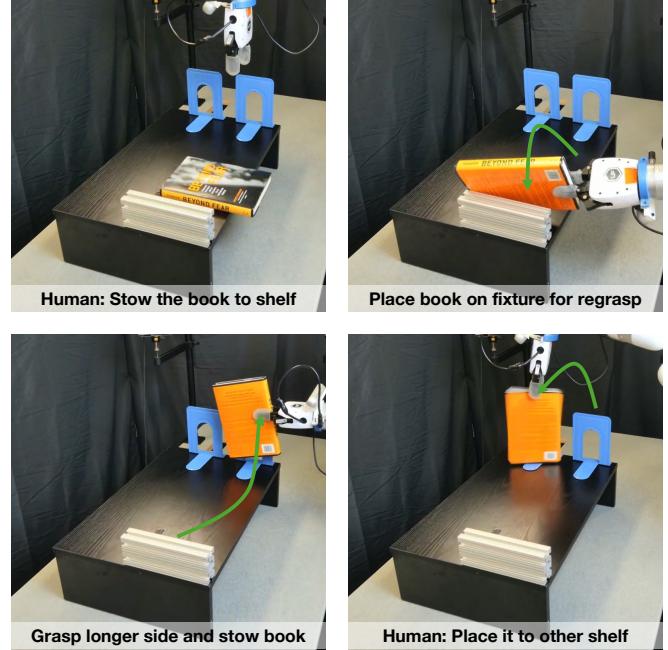


Fig. 8: Case study of a complex task with in-context examples. The robot uses the environment to regrasp and stow the book. Then, a human updates the instructions to place it on the other shelf.

We present results on a complex 3D understanding task. The task involves stowing a book on a shelf, where the book is initially positioned with only its shorter edge graspable. The instruction is to place the book on the shelf. However, the robot cannot place the book directly with the shorter edge grasped, as this would result in a collision between the book and the table due to the position of its arm. To complete this task, the robot must perform multiple steps: first, it needs to regrasp the book along its longer edge using some part of the environment, and only then can it stow the book on the shelf. After the robot places the book on the initial shelf, a human intervenes by adding an instruction to move the book to a different shelf.

Given the complexity of this long-horizon task, we employ in-context examples to guide the VLM. With this change, our system is able to successfully perform the task. Figure 8 illustrates the progression of the task.