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

Shivansh Patel[1*], Xinchen Yin[1*], Wenlong Huang[2], Shubham Garg[3], Hooshang Nayyeri[3], Li Fei-Fei[2], Svetlana Lazebnik[1], Yunzhu Li[4]

***Abstract*— Task specification for robotic manipulation in open-world environments is challenging. Importantly, this process requires 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 IKER conditioned on these keypoints. It operates on the spatial relationships between keypoints, enabling precise SE(3) control and leveraging VLMs as proxies to encode human priors about robotic behaviors. We reconstruct real-world scenes in simulation and use the generated rewards to train 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/**

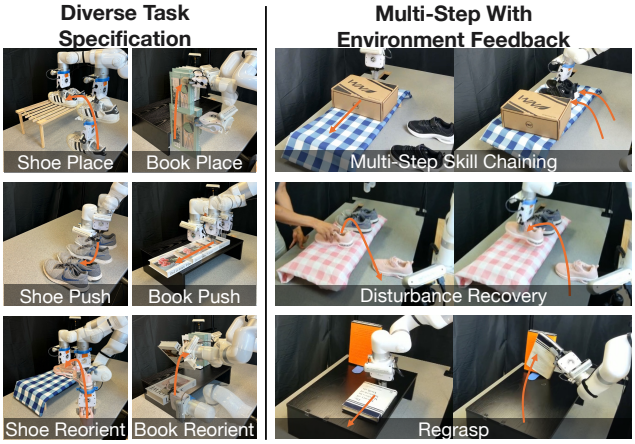

Fig. 1: **Capabilities of Our Framework.** Our framework is designed to handle a wide range of real-world tasks. It can be seamlessly chained to execute multi-step tasks. It exhibits notable capabilities, including robustness to disturbances and the ability to solve problems creatively.

I. Introduction

Figure 3 illustrates a robot tasked with placing a pair of shoes on a rack, but a shoe box is occupying the rack, leaving insufficient space for both shoes. The robot cannot perform the requested action immediately. It 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 often involve multiple implicit steps. In such cases, rigid, predefined instructions fail to capture the complexities of interaction required to accomplish the goal. Crucially, task specifications must incorporate human priors—expectations about how the robot should behave. For instance, rather than attempting to squeeze the shoes in awkwardly, the robot should first clear space.

Recent advancements in vision-language models (VLMs) have demonstrated their ability to encode rich world knowledge by pretraining on vast and diverse datasets [1–8]. This capability makes them particularly promising for task specification in robotic systems operating in unstructured, real-world environments, where tasks often involve multiple implicit steps. VLMs excel in processing free-form language descriptions, allowing them to interpret complex task instructions expressed in natural language. Their broad world knowledge helps bridge the gap between human expectations and robot behavior, capturing human-like behavioral priors and strategies for problem-solving. However, leveraging VLMs for robotic control presents two technical challenges: 1) *Fine-grained SE(3) pose control*, requiring precise manipulation of objects in 3D space; 2) *Iterative decision-making*, where VLMs refine their predictions through embodied feedback—the environment changes as a result of the robot's actions.

In this work, we introduce **Iterative Keypoint Reward (IKER)**, a visually grounded reward function for robotic manipulation that addresses these challenges. IKER leverages keypoints to define reward functions based on their spatial relationships, enabling precise control over object poses in SE(3) space. Inspired by recent work [9], we draw the observation that we can encode both positions and orientations of the objects 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 VLMs continuously update task specifications based on embodied feedback from the robot's interactions with the environment. This iterative process encodes human-like behavioral priors into the robot's actions, ensuring that it refines its behavior dynamically—adjusting

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

strategies and adopting intermediate steps, such as repositioning objects for a better grasp.

To ensure scalable and robust policy training, we employ a real-to-sim-to-real transfer. 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, grounding the task specifications in the actual environment. These rewards are then transferred to simulation for training. Finally, the optimized policies are deployed back into the real world, ensuring that the robot's behavior is both grounded in real-world data and robustly trained in simulation.

Leveraging this framework, we demonstrate the efficacy of IKER across diverse scenarios involving real-world objects like shoes and books. These scenarios include both prehensile tasks, such as grasping and placing shoes on racks, and non-prehensile tasks, like pushing or 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.

In summary, our contributions are as follows:

- We introduce a visually grounded reward representation IKER, that serves as flexible task specification, enabling robots to tackle complex open-world tasks.
- We demonstrate that IKER possesses the unique advantage of incorporating human-like behavioral priors and an iterative reward shaping process.
- We integrate IKER with a real-to-sim-to-real framework to perform both prehensile and non-prehensile tasks, which is robust in challenging long-horizon tasks.

II. Related Work

VLMs in Robotics. VLMs have appeared as a prominent tool in robotics [10–23]. 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, 24]. Our work aligns with the former, with an emphasis on flexible and adaptable task specifications in complex, real-world environments.

For task specification, many works employ VLMs to break down complex tasks into manageable subtasks. Ahn *et al.* [10] use VLMs to parse long-horizon tasks and decompose them into sequential steps executable by the robots. Belkhale *et al.* [25] 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 enhancing adaptability in open-world scenarios by dynamically interpreting tasks without relying on fixed assumptions about the environment. This flexibility enables more robust handling of diverse and complex tasks in the real-world.

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.* [26] 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, ensuring adaptability throughout the task execution. Additionally, our approach also supports non-prehensile manipulations, such as pushing.

Some works have also explored using VLMs to specify task objectives through reward function generation. Works such as [27–30] employ LLMs to generate reward functions that train policies. However, most of these approaches have limitations for real-world applicability. Some lack demonstrations on real robots [28], or are restricted to a single real-world scenario [30], or focus on highly constrained tasks like a robot dog walking on a ball [29]. In contrast, our work demonstrates the versatility and robustness of VLM-generated rewards by tackling diverse and challenging real-world manipulation tasks, highlighting the effectiveness of our approach in multi-step, complex environments.

Real-Sim-Real. Real-to-sim has gained significant attention for its ability to facilitate efficient agent training. Once a scene is transferred to simulation, it can be used for a wide range of tasks, including RL training. Several approaches focus on reconstructing rigid bodies for use in simulation [31–33]. For instance, Kappler et al. [31] introduced a method for reconstructing rigid objects to facilitate grasping. Some works rather focus on reconstructing articulated objects [34–38]. Huang et al.[34] presented methods for reconstructing the occluded shapes of articulated objects. Jiang *et al.* [38] introduced a framework, DITTO, to generate digital twins of articulated objects from real-world interactions. In our work, we utilize BundleSDF [39] to generate object meshes that are transferred to the simulation. We use it because it is fast and has state-of-the-art performance.

Sim-to-real transfers has shown great performance in a variety of skills, including tabletop manipulation [40, 41], mobile manipulation [42, 43], dynamic manipulation [44], dexterous manipulation [45, 46], and locomotion [47, 48]. However, directly deploying learned policies to physical robots cannot guarantee successful performance due to sim-to-real gap. To bridge sim-to-real gap, researchers has developed many techniques, such as system identification [49–52], domain adaptation [53–55], and domain randomization [47, 56–58]. In our work, we use domain randomization

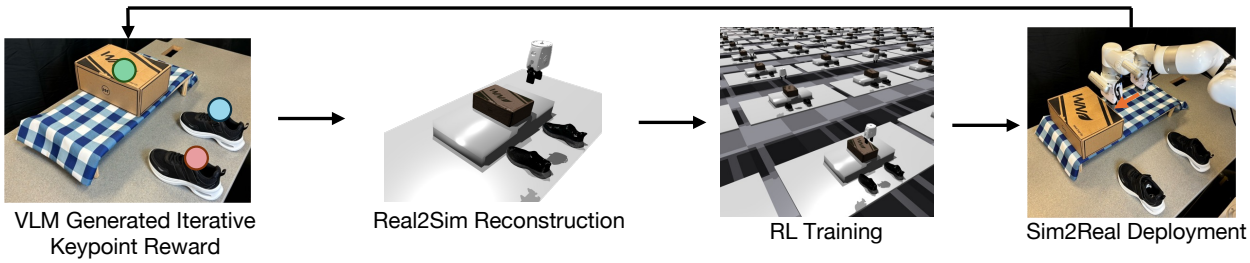

Fig. 2: **Framework Overview.** Iterative Keypoint Reward (IKER) is a visually grounded reward generated by Vision-Language Models (VLMs) as task specification. The framework reconstructs the real-world scene in simulation, and the generated reward is used to train RL policies which are subsequently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

Algorithm 1 IKER Execution Framework

Initialize: `done` ← `false`, `execution_history` ← [ ]
while `true` **do**
  (`keypoint_locations`, `keypoints_image`) ← GetKeypoints()
  `code` ← QueryVLM(`keypoints_image`, `execution_history`)
  (`done`, `final_keypoints`) ← `Execute(code)`
  if `done` is `true` **then**
    `break`
  end if
  Append (`keypoints_image`, `code`) to `executionHistory`
  s_i ← `TransferSceneToSimulation()`
  π_i ← `LearnPolicy`(s_i, final_keypoints)
  `ExecutePolicyInRealWorld`(π_i)
end while

as it does not require any interaction data from real-world during training. It relies entirely on simulation and makes policies robust by exposing them to a wide variety of randomized conditions within simulation. Recently, Torne *et al.* [59] 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

Herein we first 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.

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 sequentially obtain a set 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. In this scenario, 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 keypoints $k_{j=1}^{K}$ is given (discussed later in Sec. III-B), each specifying a 3D position in the task space

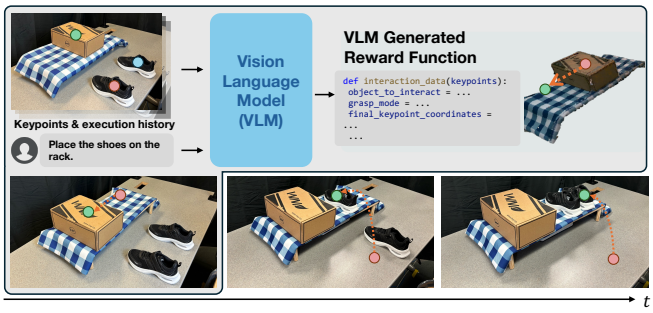

Fig. 3: **Iterative Keypoint Reward Generation.** We first obtain keypoints in the scene. These, combined with a human command and execution history, are processed by a Vision-Language Model (VLM) to generate code that maps keypoints to the reward function.

and marked numerically as $1, \ldots, K$ in the RGB observation. Using these keypoints, our objective is to automatically generate a reward function, termed IKER, that maps the keypoint positions to a scalar reward $f^{(i)} : \mathbb{R}^{K\times 3} \to \mathbb{R}$.

To generate the reward function $f^{(i)}$, we use a VLM (GPT-4o [1] in our case), which is provided with the context C_i defined as:

$$C_i = \left\{ I, O_1, f^{(1)}, \ldots, O_{i-1}, f^{(i-1)}, O_i \right\}.$$

This context includes:

1) The human instruction I describing the task.
2) The sequence of previous observations and reward functions up to step $i-1$, i.e. $\{O_1, f^{(1)}, \ldots, O_{i-1}, f^{(i-1)}\}$.
3) The current RGB observation O_i with overlaid keypoints.

Additionally, the VLM is guided by a prompt (discussed in Sec. B) that instructs to generate a Python function for the reward $f^{(i)}$. The prompt directs the VLM to break down the task into executable steps, 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. It also instructs the VLM to present all outputs in a prescribed code format and set the flag `done = True`, signaling that the task is completed.

Upon receiving the final keypoint locations by executing the generated code, we compute a scalar reward that evaluates the policy's performance. The reward function, r_{total}, is designed to facilitate learning by combining several 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 object in the direction of the target location.
- 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.

The total reward function is defined as:

$$r_{\text{total}} = \alpha_{\text{dist}} r_{\text{dist}} + \alpha_{\text{dir}} r_{\text{dir}} + \alpha_{\text{align}} r_{\text{align}} + \alpha_{\text{bonus}} r_{\text{bonus}} + \alpha_{\text{penalty}} r_{\text{penalty}}$$

where α_{dist}, α_{dir}, α_{align}, α_{bonus}, and α_{penalty} are weighting hyperparameters that we set. This reward structure enables the agent to learn complex manipulation tasks by providing continuous feedback.

B. Transferring real-world scene to simulation

We transfer the real-world scene within the workspace boundary to simulation. First, we 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 [39], 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 [60] 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.

We leverage the generated meshes 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. Keypoints that are too close in the image projection are removed. Conversely, for static objects like shoe racks, which are part of the environment, keypoints are uniformly distributed across their surfaces.

C. Train policy in Simulation

We directly 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. In simulation, we employ a heuristics-based grasp for faster training.

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 $\mathcal{K}_o = \{\mathbf{k}_{o_1}, \ldots, \mathbf{k}_{o_n}\} \in \mathbb{R}^{3n}$, and their corresponding target positions $\mathcal{K}_t = \{\mathbf{k}_{t_1}, \ldots, \mathbf{k}_{t_n}\} \in \mathbb{R}^{3n}$. $\mathcal{K}_t$ is derived from the reward function f 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.

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 [61] simulator with the PPO [62] algorithm. We use an actor-critic architecture [63] 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 [64] 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: Recognizing the challenges inherent in transferring policies between the simulation and the real world, we employ domain randomization to bridge the real-to-sim-to-real gaps. Domain randomization is applied to object properties like friction, mass, center of mass, restitution, compliance, and geometry. We further randomize the object position, the gripper location, as well as the grasp within a range. We found these to be especially crucial for non-prehensile tasks like pushing.

D. Deploy Trained Policy in Real-World

The trained RL policy π_i is then deployed directly in the real-world. Since the policy outputs end-effector pose, we employ inverse kinematics to compute the joint angles at each timestep. The RL policy operates at 10Hz frequency, producing action commands that are then clipped to ensure the end effector remains within the workspace limits. For keypoint tracking, we utilize FoundationPose [60] to estimate the object's pose. These pose estimates are subsequently used to compute the keypoint locations that are defined relative to the objects. We use AnyGrasp [65] to detect grasps in the real-world. The VLM predicts the object to interact with, and the optimal grasp is selected from AnyGrasp detection using back and top views of the object.

IV. Experiments and Analysis

We aim to address the following research questions: (1) We investigate the effectiveness of IKER as a reward representation. Specifically, we examine whether Iterative Keypoint Reward can effectively represent a wide range of reward functions necessary for training diverse manipulation

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.

skills by leveraging world knowledge from VLMs. Additionally, we explore the feasibility of constructing a pipeline that utilizes IKER for real-to-sim-to-real transfer. (2) Leveraging task-level feedback for replanning, can the pipeline perform multi-step tasks in dynamic environments?

A. Experimental Setup, Metrics and Baselines

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

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, which are then used to query the VLM.

We compare to a human-annotated variant of IKER, where reward functions are human-specified, allowing evaluation without the VLM influence. We also compare our method with using object pose as the state representation, which is the conventional approach in RL training [66–70]. In this 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. B.

We evaluate our approach across four scenarios: 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, we report success rates averaged over 128 environments, with trials considered successful if the average keypoint distance to 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. Our experiments span four representative tasks, with results 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 suggest that IKER effectively captures diverse behaviors, comparable to object poses, making it a promising alternative for RL policy training.

In the automated 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. Furthermore, with the

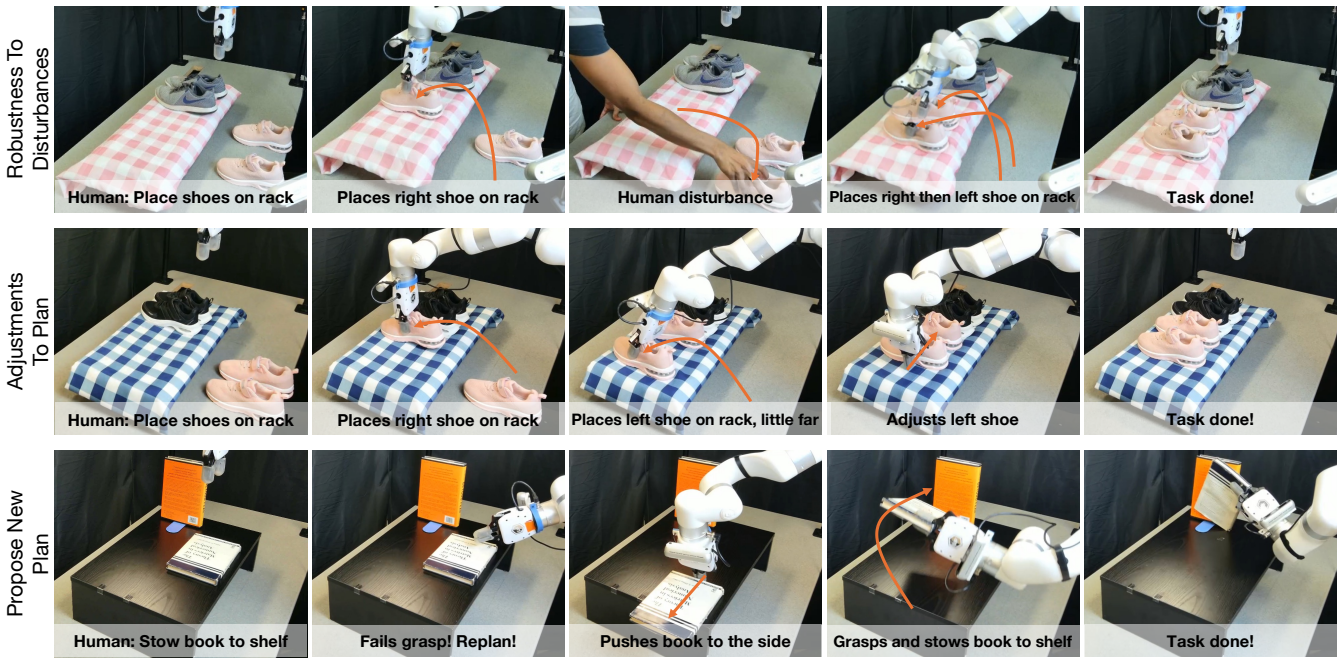

Fig. 5: **Scenarios demonstrating capabilities of our framework.** The framework is robust to disturbances and can adapt in response to unexpected events. Additionally, it can propose new plans when the original ones become infeasible.

ongoing advancements in VLMs, we expect these challenges will become less prominent over time. Additionally, some failures are caused by physical dynamics when pushing objects—at times. This issue can be partially mitigated by configuring the simulator with more accurate physics settings to better reflect real-world interactions.

C. Iterative Replanning for Multi-Step Tasks

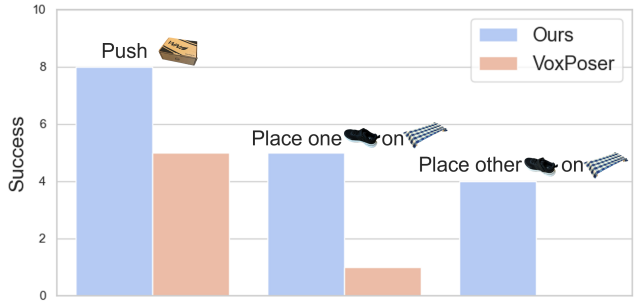

Fig. 6: **Multi-Step Task Chaining Comparison with VoxPoser.** Our framework outperforms VoxPoser at every step of the task sequence.

We demonstrate the robot's iterative chaining ability with a task involving three sequential actions: pushing a shoe box to create space, then placing a pair of shoes on a rack. Failure in any task leads to failure in the next. We test 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 excels at synthesizing motion plans for a diverse range of manipulation tasks from free-form language instructions. Notably, their plans are open-loop and lack feedback to the VLM for refining specifications at each step. To adapt it to our tasks, we enhanced VoxPoser with two major modifications: (1) VoxPoser used OWL-ViT 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, failed grasping attempts, collisions with the environment during object manipulation, or improper placement of the shoes—resulting in both shoes being stacked on top of each other and subsequently falling.

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 handled effectively. In response, the VLM predicts an alternative strategy to complete the task, adjusting the approach to ensure successful placement.

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 require to capture objects from all views to obtain object meshes. This can be simplified by using more recent methods [71] that can generate meshes from a single view. Additionally, our real-to-sim transfer uses a simplified approach that may not fully capture the intricate geometries present in real-world environments. Additionally, while our framework reconstructs multiple objects in the environment, our current implementation does 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, Unnat Jain, and members of UIUC Vision and Robotics Labs for fruitful discussions. 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 vlms," *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] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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

[30] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu, "Text2reward: Automated dense reward function generation for reinforcement learning," *arXiv preprint arXiv:2309.11489*, 2023.

[31] 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.

[32] B. Wen, W. Lian, K. Bekris, and S. Schaal, "Catgrasp: Learning category-level task-relevant grasping in clutter from simulation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 6401–6408.

[33] ——, "You only demonstrate once: Category-level manipulation from single visual demonstration," *arXiv preprint arXiv:2201.12716*, 2022.

[34] 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.

[35] X. Li, H. Wang, L. Yi, L. J. Guibas, A. L. Abbott, and S. Song, "Category-level articulated object pose estimation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 3706–3715.

[36] X. Wang, B. Zhou, Y. Shi, X. Chen, Q. Zhao, and K. Xu, "Shape2motion: Joint analysis of motion parts and attributes from 3d shapes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8876–8884.

[37] 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.

[38] 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.

[39] 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.

[40] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," *arXiv preprint arXiv: Arxiv-2109.12098*, 2021.

[41] 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

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

[43] 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.

[44] 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.

[45] 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.

[46] 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

[47] 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

[48] 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

[49] K. J. Åström and P. Eykhoff, "System identification—a survey," *Automatica*, vol. 7, no. 2, pp. 123–162, 1971.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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

[55] 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

[56] 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.

[57] 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.

[58] R. Antonova, F. Ramos, R. Possas, and D. Fox, "Bayessimig: Scalable parameter inference for adaptive domain randomization with isaacgym," *arXiv preprint arXiv:2107.04527*, 2021.

[59] 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.

[60] 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.

[61] 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.

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

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

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

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

APPENDIX

A. Grasping Subroutine

During training, the gripper's fingers remain closed by default and only open during the grasp mode. In this mode, the end-effector approaches the object with its fingers open and then closes to grasp the object. A similar behavior is maintained in real-world: the fingers remain closed until the grasp mode is activated. In this mode, AnyGrasp predicts a suitable grasp pose, and the fingers close at the determined 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 grasps in the real-world.

B. VLM Prompts

The VLM receives the image overlaid with keypoints $1, \ldots, K$, 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,
overlayed 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-stage 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,
overlayed 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 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 using pose input 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]
```

C. *Case study of a complex task*

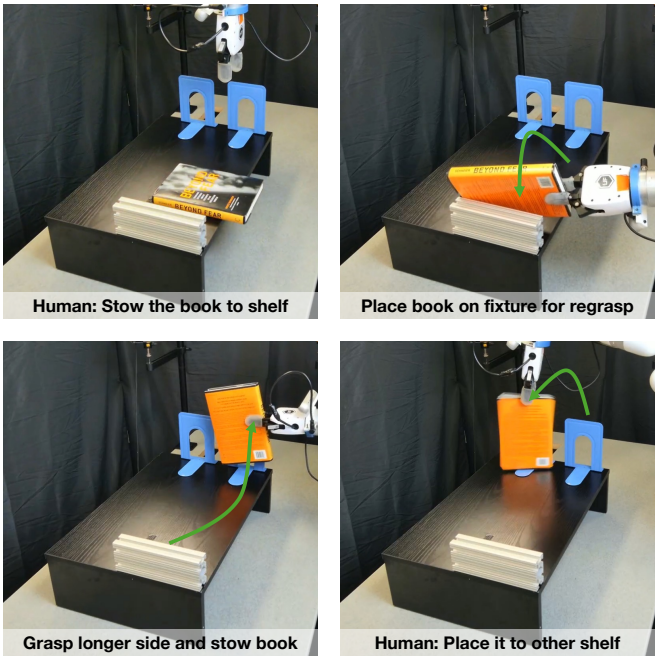

Fig. 7: Case study of a complex task

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 7 illustrates the progression of the task.