Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

Reconciling Reality through Simulation: A Real-to-Sim-to-Real Approach for Robust Manipulation

Marcel Torne[1,4] **Anthony Simeonov**[1,4] **Zechu Li**[1,3,4] **April Chan**[1,4]
Tao Chen[1,4] **Abhishek Gupta**[2*] **Pulkit Agrawal**[1,4*]
[1]Massachusets Institute of Technology [2]University of Washington [3]TU Darmstadt
[4] Improbable AI Lab

***Abstract*—Imitation learning methods need significant human supervision to learn policies robust to changes in object poses, physical disturbances, and visual distractors. Reinforcement learning, on the other hand, can explore the environment autonomously to learn robust behaviors but may require impractical amounts of unsafe real-world data collection. To learn performant, robust policies without the burden of unsafe real-world data collection or extensive human supervision, we propose RialTo, a system for robustifying real-world imitation learning policies via reinforcement learning in "digital twin" simulation environments constructed on the fly from small amounts of real-world data. To enable this real-to-sim-to-real pipeline, RialTo proposes an easy-to-use interface for quickly scanning and constructing digital twins of real-world environments. We also introduce a novel "inverse distillation" procedure for bringing real-world demonstrations into simulated environments for efficient fine-tuning, with minimal human intervention and engineering required. We evaluate RialTo across a variety of robotic manipulation problems in the real world, such as robustly stacking dishes on a rack, placing books on a shelf, and six other tasks. RialTo increases (over 67%) in policy robustness without requiring extensive human data collection. Project website at https://real-to-sim-to-real.github.io/RialTo/.**

I. Introduction

Imagine a robot that can de-clutter kitchens by putting dishes on a dish rack. Consider all the environmental variations that might be encountered: different configurations of plates or changes in rack positions, a plate unexpectedly slipping in the gripper during transit, and visual distractions, including clutter and lighting changes. For the robot to be effective, it must robustly solve the task across the various scene and object perturbations, without being brittle to transient scene disturbances. Our desiderata is a framework that makes it *easy* for humans to program the robot to achieve a task *robustly* under these variations or disturbances. To be a scalable choice for deployment, the framework should not make task-specific assumptions and must seamlessly apply to many tasks.

To design these types of robust robot controllers, one could attempt to train policies across a massive range of scenes and with highly variable objects [12, 21]. This is hard-pressed to provide a scalable solution to robotic learning for two reasons - (1) it is challenging to actually collect or synthesize data across a massive range of scenes since content creation can be challenging in simulation and data collection can be challenging for the real world, (2) a widely general, robust policy may be overly conservative, lowering its performance on the specific target domains encountered on deployment. Alternatively, we suggest that to maximally benefit a *specific* user, it is more critical that the robot achieves high success in their *particular* home environment, showing robustness to various local disturbances and distractors that might be encountered in this setting. With this in mind, our goal is to develop a robot learning technique that requires minimal human effort to synthesize visuomotor manipulation controllers that are extremely robust for task performance in deployment environments. The question becomes - how do we acquire these robust controllers without requiring prohibitive amounts of effort for data collection or simulation engineering?

A potential technique for data-driven learning of robotic control policies is to adopt the paradigm of imitation learning (IL), learning from expert demonstration data [57, 55, 25]. However, controllers learned via imitation learning tend to exhibit limited robustness unless a large number of demonstrations are collected. Furthermore, imitation learning does not learn to recover from mistakes or out-of-distribution disturbances unless such behaviors were intentionally demonstrated. This makes direct imitation learning algorithms unsuitable for widespread, robust deployment in real-world scenarios.

The alternative paradigm of reinforcement learning (RL) allows robots to train on self-collected data, reducing the burden on humans for *extensive* data collection [31] and to discover robust recovery behaviors *beyond* a set of pre-collected demonstrations (e.g., re-grasping when an object is dropped, re-aligning when an object moves in the gripper, adjusting to external perturbations, etc. — see examples in Fig. 1). However, directly performing RL in the real world is prohibitively slow, often results in unsafe data collection, and is challenging due to problems like resets and reward specification [74]. Therefore, currently, it's impractical in many cases to employ RL for learning robust control policies directly in the real world. Simulation, on the other hand, offers the ability to collect significant amounts of data broadly, cheaply, safely, and with privileged information [12, 34, 60, 39, 1]. However,

* Equal advising

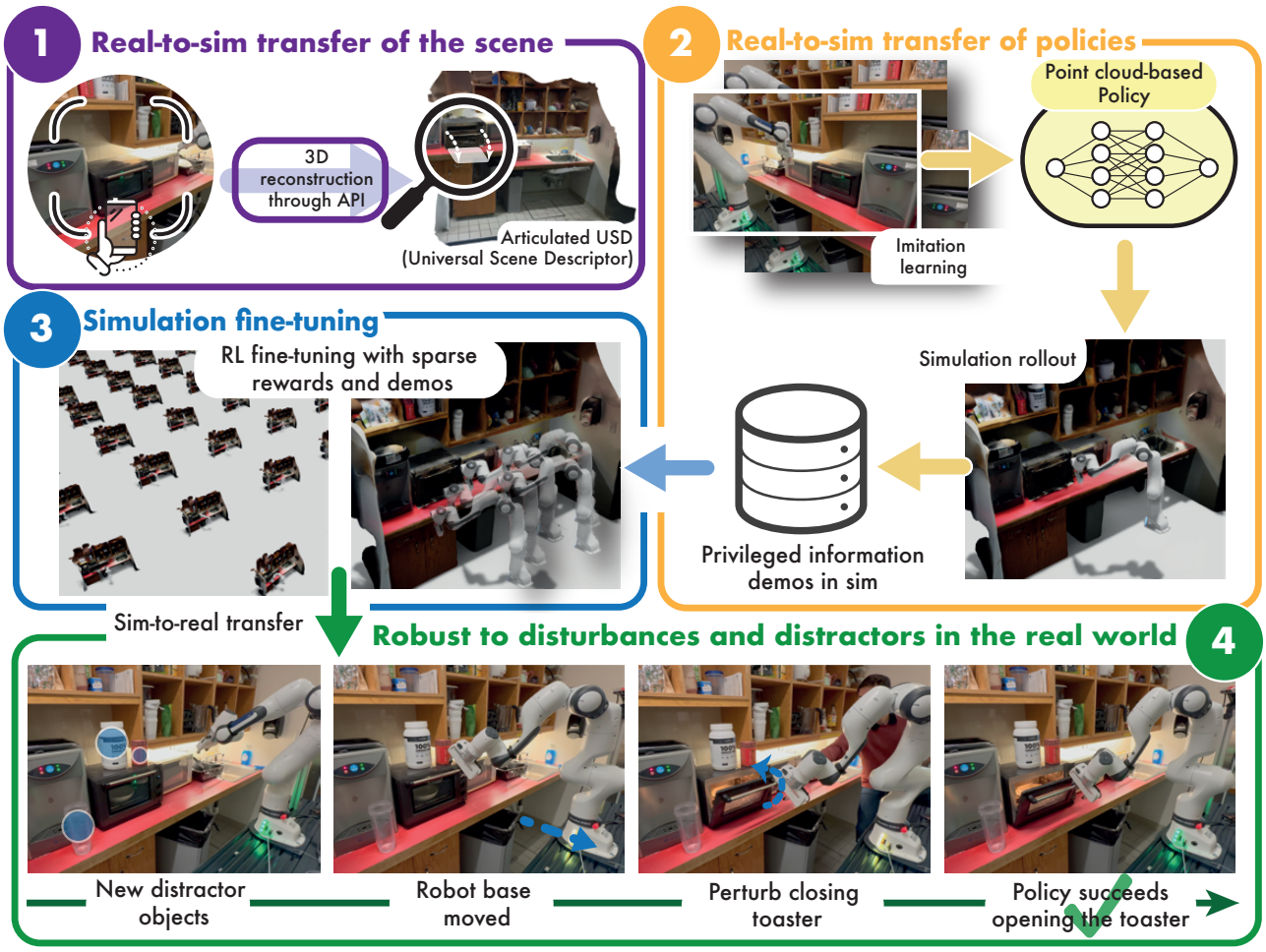

Fig. 1. **RialTo system overview**. 1) Transfer the real-world scene to the simulator through an easy-to-use API (see Section III-B). 2) Transfer a policy learned from real-world demonstrations to collect a set of demonstrations with privileged information in simulation. We note this step is optional, and **RialTo** is compatible with skipping this step and providing demonstrations in simulation (see Section IV-C2) 3) Use the collected set of demonstrations to bias exploration in the RL fine-tuning with sparse rewards of a state-based policy (see Section III-C) 4) Perform teacher-student distillation and deploy the policy in the real world obtaining robust behaviors (see Section III-D).

manually constructing geometrically, visually, and physically realistic simulation environments for problems like robotic manipulation in the home can be time and labor-intensive, making it an impractical alternative at scale.

To safely and efficiently learn robust manipulation behaviors, our key insight is to train RL controllers on *quickly* constructed simulation scenes. By leveraging a video from the target deployment domain, we can obtain scenes complete with accurate geometry and articulation that reflect the appearance and kinematics of the real world. These "in-domain" simulation environments can serve as a sandbox to safely and quickly learn robust policies across various disturbances and distractors, without requiring expensive exploration in the real world. We show how imitation learning policies trained with small numbers of real-world demonstrations can be robustified via large-scale RL fine-tuning in simulation on these constructed simulation environments, using minimal amounts of human effort in terms of environment design and reward engineering. To remove the burden of reward engineering, we leverage a set of real-world demonstrations that bootstrap efficient fine-tuning with reinforcement learning. These real-world demonstrations help narrow the sim-to-real gap and increase the performance of our policies, as shown in Section IV-B. However, transferring real-world demonstrations into simulation is non-trivial because we do not have access to the Lagrangian state of the environment (e.g. object poses). We therefore propose a new "inverse-distillation" technique that enables transferring real-world demos into the simulation. After using RL in constructed simulation environments to robustify the real-world imitation learning policies, the fine-tuned policies can be transferred back to the real world with significantly improved success rates and robustness to test-time disturbances.

Overall, our pipeline simultaneously improves the effectiveness of both reinforcement *and* imitation learning. RL in simulation helps make imitation learning policies deployment-ready without requiring prohibitive amounts of unsafe, interactive data collection in the real world. At the same time,

bootstrapping from real-world demonstration data via inverse distillation makes the exploration problem tractable for RL fine-tuning in simulation. This minimizes the amount of task-specific engineering required by algorithm designers such as designing the dense rewards or manually designing the scenes.

Concretely, we propose **RialTo**, a system for robustifying real-world imitation learning policies without requiring significant human effort, by constructing realistic simulation analogs for real-world environments on the fly and using these for robust policy learning. Our contributions include:

- A simple policy learning pipeline that synthesizes controllers to perform diverse manipulation tasks in the real world that (i) reduces human effort in constructing environments and specifying rewards, (ii) produces robust policies that transfer to real-world, cluttered scenes, showing robustness to disturbances and distractors, (iii) requires minimal amounts of expensive and unsafe data collection in the real world.
- A novel algorithm for transferring demonstrations from the real world to the reconstructed simulation to bootstrap efficient reinforcement learning from the low-level Lagrangian state for policy fine-tuning. We show that this real-to-sim transfer of human demonstrations both improves efficiency and biases policies toward realistic behavior in simulation which effectively transfers back to the real world.
- An intuitive graphical interface for quickly scanning and constructing digital twins of real-world scenes with articulation, separated objects, and accurate geometries.
- We present extensive experimental evaluation showing that **RialTo** produces reactive policies that solve several manipulation tasks in real-world scenes under physical disturbances and visual distractions. Across eight diverse tasks, our pipeline provides an improvement of 67% over baselines in average success rate across scenarios with varying object poses, visual distractors, and physical perturbations.

II. Related Work

Learning Visuomotor Control from Demonstrations: Behavior cloning (BC) of expert trajectories can effectively acquire robot control policies that operate in the real world [19, 13, 71, 6, 20, 38]. While several works have used BC to learn performant policies from small to moderately-sized datasets [13, 71, 38], performance tends to drop when the policy must generalize to variations in scene layouts and appearance. Techniques for improving BC often require much larger-scale data collection [6, 56], raising scalability concerns. Other techniques support generalization with intermediate representations [20] and leverage generative models to add visual distractors [70, 37]. These can improve robustness to visual distractors but do not address physical or dynamic disturbances, as these require producing actions not present in the data.

Fine-tuning Imitation with RL and Improving RL with Demonstrations: Reinforcement learning has been used to improve the performance of models originally trained with imitation learning. RL has exploded in its capacity for fine-tuning LLMs [48] and image generation models [4], learning rewards from human feedback [14]. In robotics, prior work has explored techniques such as offline RL [69, 46, 33], learning world models [41, 18], and online fine-tuning in the real world [2, 3, 22, 69]. Expert demonstrations have also been used to bootstrap exploration and policy learning with RL [28, 27, 53, 73]. We similarly combine imitation and RL to guide exploration in sparse reward settings. However, our pipeline showcases how demonstrations additionally benefit RL by biasing policies toward physically plausible solutions that compensate for imperfect physics simulation.

Sim-to-real policy transfer: RL in simulation has been used to synthesize impressive control policies in a variety of domains such as locomotion [39, 34, 32], dexterous in-hand manipulation [11, 12, 1, 23], and drone flight [60]. Many simulation-based RL methods leverage some form of domain randomization [64, 50], system identification [26, 62], or improved simulator visuals [54, 24] to reduce the simulation-to-reality (sim-to-real) domain gap. Prior work has also shown the benefit of "teacher-student" distillation [12, 32, 59, 9], wherein privileged "teacher" policies learned quickly with RL are distilled into "student" policies that operate on sensor observations. To acquire transferable controllers, we similarly leverage GPU-accelerated simulation, teacher-student training, and domain randomization across parallel environments. However, we address the more challenging scenario of household manipulation, which is characterized by richer visual scenes, and minimize the necessary engineering effort by relying on sparse rewards. We also simplify sim-to-real by training on digital twin assets and co-training with real data [66].

Real-to-sim transfer of scenes: Designing realistic simulation environments has been studied from the perspective of synthesizing digital assets that reflect real objects. Prior work has used tools from 3D reconstruction [29] and inverse graphics [10] for creating digital twins, and such real-to-sim pipelines have been used for both rigid and deformable [61] objects. These approaches are all compatible with our system and could be used to automate real-to-sim scene transfer and reduce human effort. Our work similarly leverages advancements in 3D vision [63] for reconstructing object geometry, but we also introduce an easy-to-use GUI for building a URDF/USD with accurate articulations. Furthermore, our GUI could be used to improve the aforementioned methods by making it easier to collect a large dataset of human-annotated articulated scenes. The accuracy of the simulator could be improved further combining our GUI with the latest system identification research [40]. **Real-to-sim-to-real transfer**: Prior work has used NeRF [42] and other 3D reconstruction techniques to create realistic scene representations for improving manipulation [72], navigation [16, 8] and locomotion [7]. These works, however, only use the visual component of the synthetic scene and do not involve any physical interaction with a reconstructed geometry. As a result, these systems cannot adjust to environmental changes beyond visual distrac-

tions. For instance, different grasp poses may require different placements, and a policy cannot discover these novel behaviors without physically interacting with the environment during training. A limited number of works have learned policies that interact with the reconstructed environments, but they either simplify the reconstructed shapes [36] or are limited to simple grasp motions [67].

III. RialTo: A Real-to-Sim-to-Real System for Robust Robotic Manipulation

A. System Overview

Our goal is to obtain a control policy that maps real-world sensory observations to robot actions. We only assume access to a small set of demonstrations (~ 15) containing (observation, action) trajectories collected by an expert, although in principle **RialTo** can also be used to robustify large, expressive pretrained models as well. Our approach robustifies real-world imitation learning policies using simulation-based RL to make learned controllers robust to disturbances and distractors not present in the demos. The proposed pipeline, **RialTo**, achieves this with four main steps (Fig 1):

1) We construct geometrically, visually, and kinematically accurate simulation environments from real-world image capture. We leverage 3D reconstruction tools and develop an easy-to-use graphical interface for adding articulations and physical properties.
2) We obtain a set of successful trajectories containing privileged information (such as Lagrangian state, e.g. object and joint poses) in simulation. We propose an "inverse distillation" algorithm to transfer a policy learned from real-world demonstrations to create a dataset of trajectories (i.e., demos) in the simulation environment.
3) The synthesized simulation demos bootstrap efficient fine-tuning with RL in simulation using an *easy-to-design* sparse reward function and low-dimensional state space, with added randomization to make the policy robust to environmental variations.
4) The learned policy is transferred to reality by distilling a state-based simulation policy into a policy operating from raw sensor observations available in the real world [9, 12]. During distillation, we also co-trained with the original real-world demonstrations to capitalize on the combined benefits of simulation-based robustification and in-domain real-world data.

The following sections describe each component in detail, along with a full system overview in Fig 1.

B. Real-to-Sim Transfer for Scalable Scene Generation

The first step of **RialTo** is to construct geometrically, visually, and kinematically realistic simulated scenes for policy training. This requires (i) generating accurate textured 3D geometry from real-world images and (ii) specifying articulations and physical parameters. For geometry reconstruction, we use existing off-the-shelf 3-D reconstruction techniques. Our pipeline is agnostic to the particular method used, and we have verified the approach with a variety of scanning apps (e.g., Polycam [51] and ARCode [15]) and 3D reconstruction pipelines [63, 44], each of which convert a set of multi-view 2D images (or a video) into a textured 3D mesh. The raw mesh denoted $\mathcal{G}$, is typically exported as a single globally-unified geometry, which is unsuitable for direct policy learning. Scene objects are not separated and the kinematics of objects with internal joints are not reflected. Physical parameters like mass and friction are also required and unspecified. We therefore further process the raw mesh $\mathcal{G}$ into a set of separate bodies/links $\{\mathcal{G}_i\}_{i=1}^{M}$ with kinematic relations $\mathcal{K}$ and physical parameters $\mathcal{P}$.

While there are various automated techniques for automatically segmenting and adding articulations to meshes [29], in this work, we take a simple human-centric approach. We offer a simple graphical interface for humans to quickly separate meshes and add articulations (see Fig. 2). Our GUI allows users to upload their own meshes and drag/drop, reposition, and reorient them in the global scene. Users can then separate meshes and add joints between different mesh elements, allowing objects like drawers, fridges, and cabinets to be scanned and processed. Importantly, our interface is lightweight, intuitive, and requires minimal domain-specific knowledge. We conducted a study (Section VI) evaluating six non-expert users' experiences with the GUI and found they could scan complex scenes and populate them with a couple of articulated objects in under 15 minutes of active interaction time. Examples of real-world environments with their corresponding digital twins are shown in Fig 4 and Appendix Fig. 16.

The next question is —how do we infer the physics parameters that faithfully replicate the real world? While accurately identifying physical parameters is possible, this can be challenging without considerable interaction [5, 68]. While adapting to dynamics variations is an important direction for future work, in this system we set a single default value for mass and friction uniformly across objects and compensate for the sim-to-real gap to actual real-world values by constraining the learned policy to be close to a small number of real-world demonstrations as discussed in Section III-C.

This procedure produces a scene $\mathcal{S} = \{\{\mathcal{G}_i\}_{i=1}^{M}, \mathcal{K}, \mathcal{P}\}$ represented in a USD/URDF file that references the separated meshes and their respective geometric $(\mathcal{G}_i\}_{i=1}^{M})$, kinematics $(\mathcal{K})$ and physical parameters $(\mathcal{P})$. This environment can subsequently be used for large-scale policy robustification in simulation.

C. Robustifying Real-World Imitation Learning Policies in Simulation

Given the simulation environment generated in Section III-B, the next step in **RialTo** involves learning a robust policy in simulation that can solve desired tasks from a wide variety of configurations and environmental conditions. While this can be done by training policies from scratch in simulation, this is often a prohibitively slow process, requiring considerable manual engineering. Instead, we will adopt a fine-tuning-based approach, using reinforcement learning in

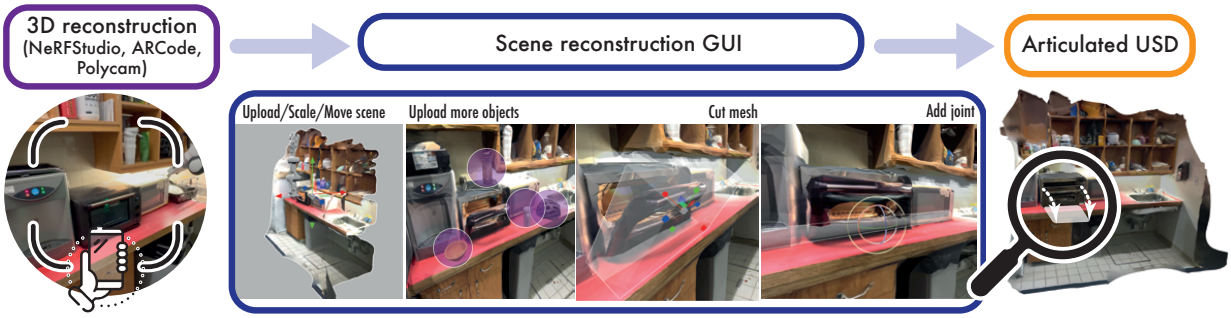

Fig. 2. Overview of the real-to-sim pipeline for transfering scenes to the simulator. The first stage consists of scanning the environment, using off-the-shelf tools such as NeRFStudio, ARCode, or Polycam. Each has its strengths and weaknesses and should be used appropriately (see Appendix XII for recommendations). The second stage consists of uploading the reconstructed scene into **RialTo**'s GUI where the user can cut the mesh, specify joints, and organize the scene as desired. Once complete, the scene can be downloaded as a USD asset, which can be directly imported into the simulator.

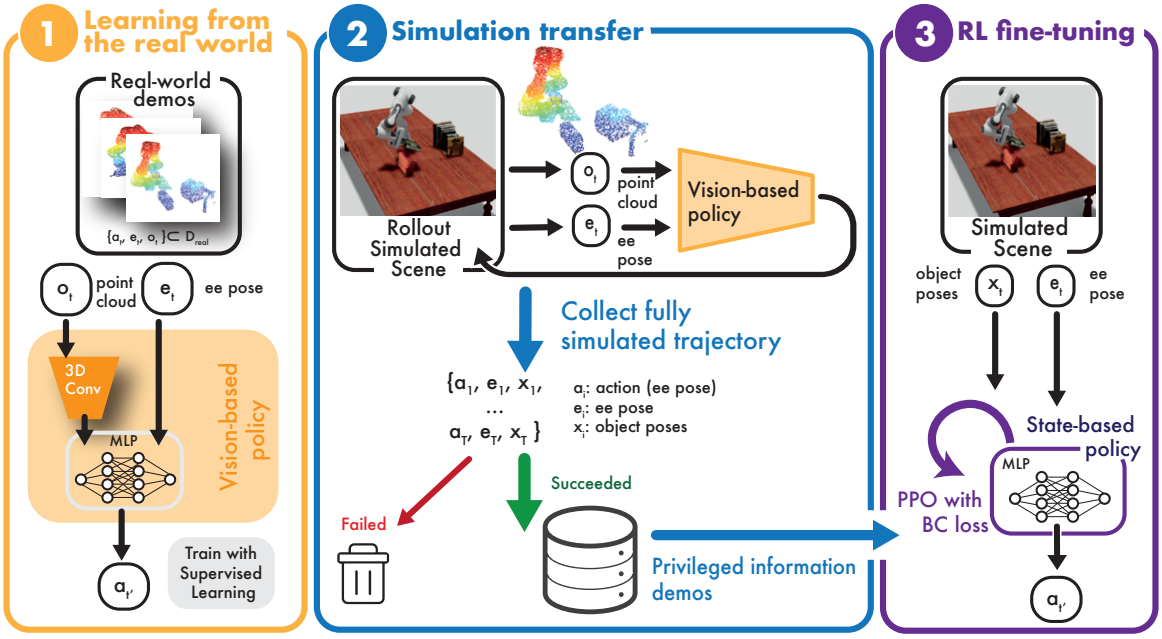

Fig. 3. **Inverse distillation & RL fine-tuning**. We introduce a novel procedure for going from point cloud-based policies trained from real-world demonstrations $\mathcal{D}_{\text{real}}$ to a robust privileged state-based policy in simulation. 1) Train a vision-based policy with supervised learning on $\mathcal{D}_{\text{real}}$ 2) Rollout the vision-based policy on the simulation rendered point clouds and collect a set of 15 privileged demonstrations with object poses, $\mathcal{D}_{\text{sim}}$ 3) Train a robust state-based policy with RL and a sparse reward, adding a BC loss fitting $\mathcal{D}_{\text{sim}}$ to bias exploration and set a prior on real-world-transferable policies.

simulation to fine-tune a policy initialized from a small number of expert demonstrations collected in the real world. Since training RL directly from visual observations is challenging, we would ideally like to finetune simulation policies that are based on a privileged Lagrangian state. However, real-world demonstrations do *not* have access to the low-level state information in the environment. To enable the bootstrapping of RL finetuning in simulation from a privileged state using real-world demonstrations, we introduce a novel "inverse distillation" (Section III-C1) procedure that is able to take real-world demonstrations with only raw sensor observations and actions and transfer them to simulation demonstrations, complete with low-level privileged state information. These privileged information demonstrations can then be used to instantiate an efficient RL-based fine-tuning procedure (Section III-C2) in simulation to massively improve policy robustness.

1) Inverse-distillation from Real-to-Sim for Privileged Policy Transfer: We assume a human provides a small number of demonstrations in the real world $\mathcal{D}_{\text{real}} = \{(o_1^i, a_1^i), \ldots, (o_H^i, a_H^i)\}_{i=1}^N$, where trajectories contain observations o (3D point clouds) and actions a (delta end-effector pose). Considering that simulation-based RL fine-tuning is far more efficient and performant when operating from a compact state representation [32, 11] (see Section V-C) and we wish to use real-world human demonstrations to avoid the difficulties with running RL from scratch (see Section V-B), we want to transfer our observation-action demonstrations from the real world to simulation in a way that allows for subsequent RL

fine-tuning in simulation from compact state-based representations. This presents a challenge because we do *not* have an explicit state estimation system that provides a Lagrangian state for the collected demonstrations in the real world. We instead introduce a procedure, called "inverse-distillation", for converting our real-world set of demonstrations into a set of trajectories in simulation that are paired with privileged low-level state information.

Given the demonstrations $\mathcal{D}_{\text{real}}$, we can naturally train a policy $\pi_{\text{real}}(a|o)$ on this dataset via imitation learning. "Inverse distillation" involves executing this perception-based learned policy $\pi_{\text{real}}(a|o)$ in simulation, based on simulated sensor observations o, to collect a dataset $\mathcal{D}_{\text{sim}} = \{(o_1^i, a_1^i, s_1^i) \ldots, (o_H^i, a_H^i, s_H^i)\}_{i=1}^M$ of successful trajectories which contain privileged state information s_t^i. The key insight here is that while we do not have access to the Lagrangian state in the real-world demonstrations when a learned real-world imitation policy is executed from *perceptual* inputs in simulation, low-level privileged Lagrangian state information can naturally be collected from the simulation as well since the pairing between perceptual observations and Lagrangian state is known apriori in simulation. Since the goal is to improve *beyond* the real-world imitation policy $\pi_{\text{real}}(a|o)$, we can then perform RL fine-tuning, incorporating the privileged demonstration dataset $\mathcal{D}_{\text{sim}}$ into the training process, as discussed in the following subsection.

2) Reinforcement Learning Fine-tuning in Simulation: Given the privileged information dataset $\mathcal{D}_{\text{sim}}$, and the constructed simulation environment the goal is to learn a robust policy $\pi^*_{\text{sim}}(a|s)$ using reinforcement learning. There are two key challenges in doing so in a *scalable* way: (1) resolving exploration challenges with minimal reward engineering, and (2) ensuring the policy learns behaviors that will transfer to the real world. ‘ We find that both challenges can be addressed by a simple demonstration augmented reinforcement learning procedure [59, 45, 53], using the Lagrangian state-based dataset $\mathcal{D}_{\text{sim}}$. To avoid reward engineering, we define a simple reward function that detects if the scene is in a desired goal state (detailed sparse reward functions used in each task in Appendix VIII). We build on the proximal policy optimization [58] algorithm with the addition of an imitation learning loss as follows (where $\hat{A}_t$ is the estimator of the advantage function at step t [58], and V_ϕ is the learned value function):

$$\begin{aligned}\max_{\theta,\phi} \alpha \sum_{(s_t,a_t,r_t)\in\tau_{\pi_{\theta_{\text{old}}}}} \min\Big(&\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}(a_t|s_t)}}\hat{A}_t, \\ &\text{clip}\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}(a_t|s_t)}}, 1-\epsilon, 1+\epsilon\Big)\hat{A}_t\Big) \\ +\beta \sum_{(s_t,V_t^{\text{targ}})\in\tau_{\pi_{\theta_{\text{old}}}}} &(V_\phi(s_t) - V_t^{\text{targ}})^2 \\ +\gamma \sum_{(s_i,a_i)\in\mathcal{D}_{\text{sim}}} &\frac{\pi_\theta(a_i|s_i)}{\sum_{a_c}\pi_\theta(a_c|s_i)}\end{aligned} \tag{1}$$

In addition to mitigating issues associated with exploration [45, 53], leveraging the additional imitation learning term in the objective helps bias the policy toward physically plausible, safe solutions that improve transfer of behaviors to reality. During this process, we can train the policy for robustness by randomizing initial robot/object/goal poses. Appendix VIII contains complete details of our training procedure. The result is a robust policy $\pi^*_{\text{sim}}(a|s)$ operating from Lagrangian state that is successful from a wide variety of configurations and environmental conditions.

D. Teacher-Student Distillation with Co-Training on Real-World Data for Sim-to-Real Transfer

In previous sections, we described a method for efficiently learning a robust policy $\pi^*_{\text{sim}}(a|s)$ in simulation using privileged state information. However, in the real world, this privileged information is unavailable. Policy deployment requires operating directly from sensory observations (such as point clouds) in the environment. To achieve this, we build on the framework of teacher-student distillation (with interactive DAgger labeling)[56, 12] where the privileged information policy $\pi^*_{\text{sim}}(a|s)$ serves as a teacher and the perceptual policy $\pi^*_{\text{real}}(a|o)$ is the student. Since there is inevitable domain shift between simulation and real domains, this training procedure can be further augmented by co-training the distillation objective with a mix of the original real-world demonstration data $\mathcal{D}_{\text{real}}$ and simulation data drawn from $\pi^*_{\text{sim}}(a|s)$ (via the DAgger objective [12]). This results in the following co-training objective for teacher-student policy learning:

$$\begin{aligned}\max_\theta \alpha \sum_{(s_i,o_i,a_i)\sim\tau_{\pi_\theta}} &\frac{\pi_\theta(\pi_{\text{teacher}}(s_i)|o_i)}{\sum_{a_c}\pi_\theta(a_c|o_i)} \\ +\beta \sum_{(o_i,a_i)\in\mathcal{D}_{\text{real}}} &\frac{\pi_\theta(a_i|o_i)}{\sum_{a_c}\pi_\theta(a_c|o_i)}\end{aligned} \tag{2}$$

Here the first term corresponds to DAgger training in simulation, while the second term co-trains on real-world expert data. This allows the policy to take advantage of small amounts of high-quality real-world data to bridge the perceptual gap between simulation and real-world scenes and improve generalization compared to only using the data from simulation. We empirically demonstrate (Section III-D) that this significantly increases the resulting success rate in the real world. On a practical note, we refer the reader to Appendix IX for additional details on the student-teacher training scheme that enables it to be successful in the proposed problem setting.

IV. Experimental Evaluation

Our experiments are designed to answer the following questions about **RialTo**: (a) Does **RialTo** provide real-world policies robust to variations in configurations, appearance, and disturbances? (b) Does co-training policies with real-world data benefit real-world evaluation performance? (c) Is the real-to-sim transfer of scenes and policies necessary for training

For the sake of this work, we will assume that the optimal actions for the student and teacher coincide, and there are no information gathering specific challenges induced by partial observability [59]

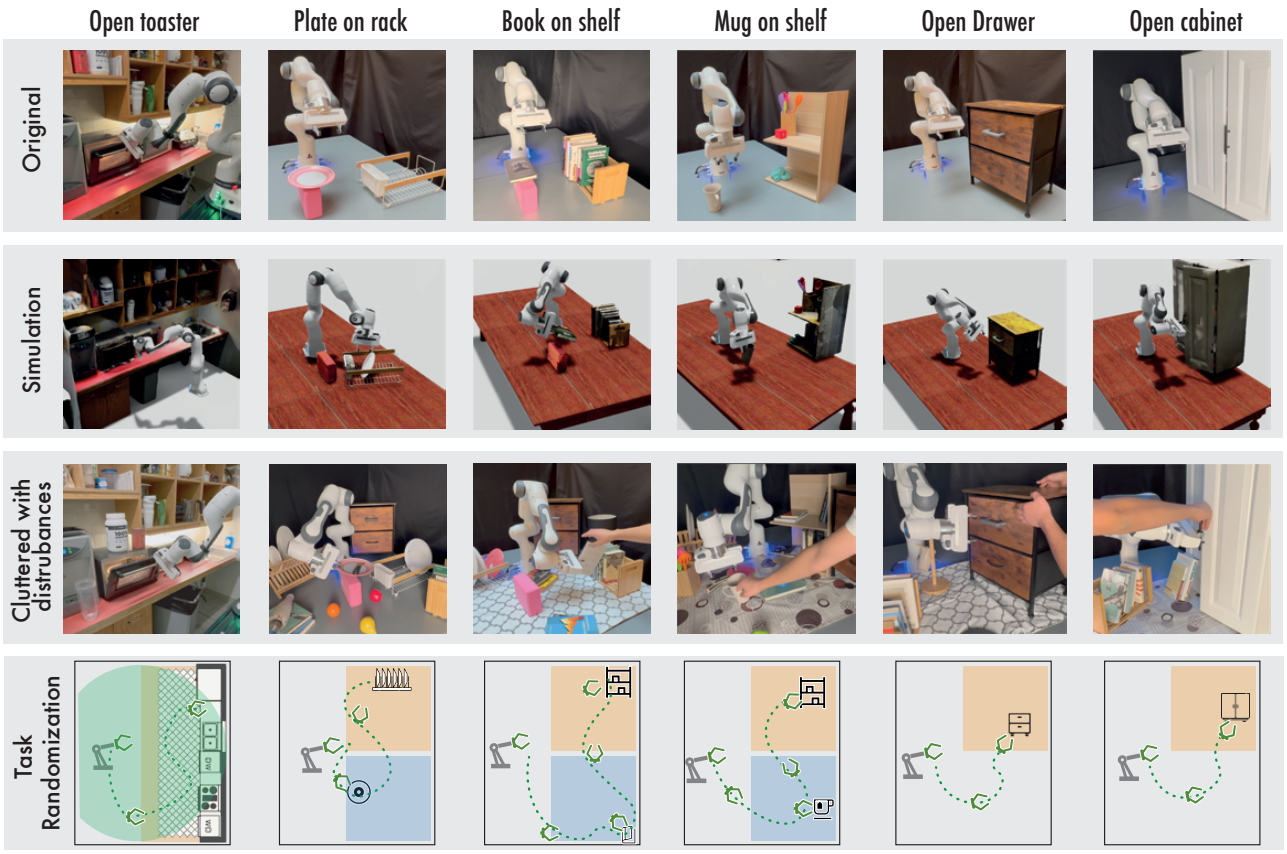

Fig. 4. We depict the six tasks used to evaluate **RialTo**. From top to bottom, we first show the original environment where we collect the demonstrations, second the simulated environment, third the environment where we do our final evaluation containing clutter and disturbances, and fourth the task randomization overview each shaded area corresponds to an approximation of how much randomization each object/robot have.

efficiency and the resulting performance? (d) Does **RialTo** scale up to more in-the-wild scenes?

To answer these questions, we evaluate **RialTo** in eight different tasks, shown in Figure 4 and 8. These include 6-DoF grasping and reorientation of free objects (*book on a shelf*, *plate on a rack*, *mug on a shelf*) and 6-DoF grasping and interacting with articulated objects (*drawer* and *cabinet*) on a tabletop and *opening a toaster*, *plate on a rack*, *putting a cup in the trash* in more uncontrolled scenes. More details on the tasks such as their sparse reward functions and randomization setups are presented in Appendix VIII. For each task, we consider three different disturbance levels in increasing order of difficulty (see Appendix VIII for more details):

1) *Randomizing object poses*: at the beginning of each episode we randomize the object and/or robot poses.
2) *Adding visual distractors*: at the beginning of each episode we also add visual distractors in a cluttered way.
3) *Applying physical disturbances*: we apply physical disturbances throughout the episode rollout. We change the pose of the object being manipulated or the target location where the object needs to be placed, close the drawer/toaster/cabinet being manipulated, and move the robot base when possible.

We conduct our experiments on a Franka Panda arm with the default parallel jaw gripper, using 6 DoF Cartesian end effector position control. For perceptual inputs, we obtain 3D point cloud observations from a single calibrated depth camera. More details on the hardware setup can be found in Appendix X. All of the results in the real world are evaluated using the best policy obtained for each method, we report the average across at least 10 rollouts and the bootstrapped standard deviation. Videos of highlights and evaluation runs are available in the website.

Throughout the next sections, we will evaluate **RialTo** against the following set of baselines and ablations: 1) Imitation learning from 15 and 50 demos (Section IV-A); 2) No co-training on real-world data (Section IV-B); 3) Co-training on demonstrations in simulation (Section IV-B); 4) **RialTo** from simulation demos (Section IV-C2); 5) Learning from an untargeted set of simulated assets (Section IV-C1); 6) **RialTo** without distractors (Section V-A); 7) **RialTo** without demos (Section V-B)

A. ***RialTo*** *Learns Robust Policies via Real-to-Sim-to-Real*

In this section, we aim to understand whether **RialTo** can solve complex tasks, showing robustness to variations in configurations, disturbances, and distractors. We compare our approach of real-to-sim-to-real RL fine-tuning against a policy

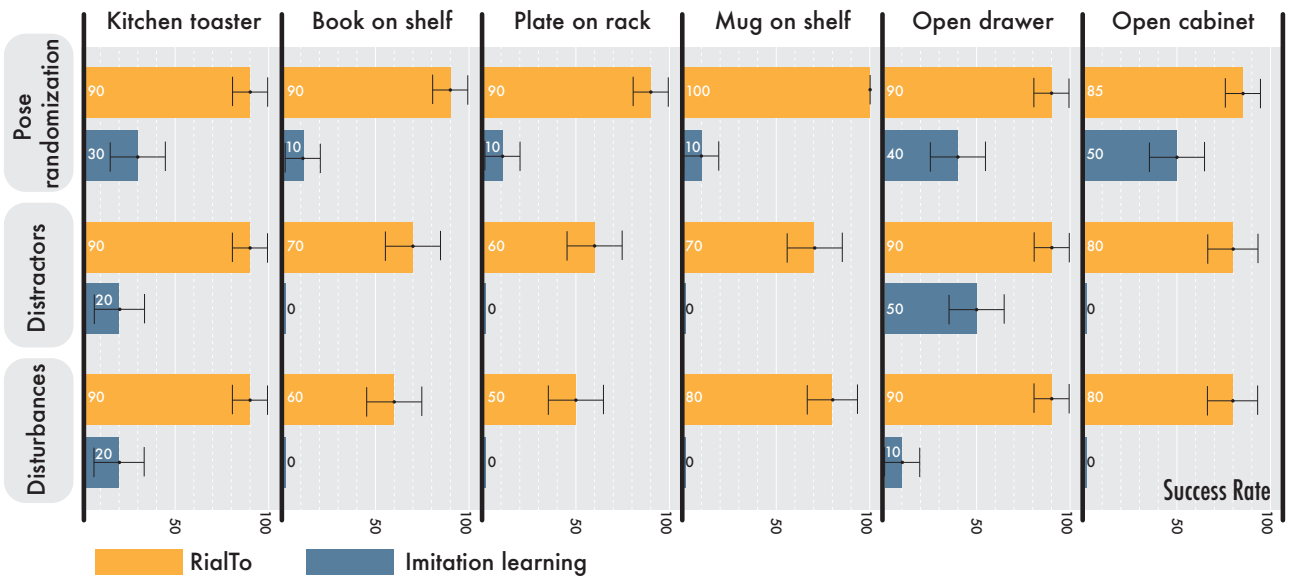

Fig. 5. Comparison of **RialTo** against imitation learning both from 15 demonstrations. **RialTo** provides robust policies across tasks and levels of distractions while imitation learning severely suffers when adding distractors and disturbances.

trained only on real-world demos via standard imitation learning (BC). We report the results of running **RialTo**'s pipeline starting from 15 demos collected directly in simulation and co-training with 15 real-world demos during the teacher-student distillation. In Section IV-C2 we show a comparison of running **RialTo** uniquely on real or sim demos.

The results in Figure 5 show **RialTo** maintains high performance across configuration levels, achieving on average 91% success across tasks for randomizing object poses, 77% with distractors, and 75% with disturbances. On the other hand, the presence of distractors and disturbances severely reduces the performance of pure imitation learning. For instance, when only randomizing the object poses, the BC baseline achieves an average of 25% success rate across tasks. Under more challenging conditions, the BC baseline drops to 11% and 5% overall performance on average for distractors and disturbances, respectively.

Figure 1, 10 and the videos in the website qualitatively show how the resulting policies are robust to many kinds of environment perturbations, including moving the robot, moving the manipulated object and target positions, and adding visual distractors that cause occlusion and distribution shift. The policy rollouts also demonstrate error recovery capabilities, correcting the robot's behavior in closed loop when, e.g., objects are misaligned or a grasp must be reattempted. This highlights that **RialTo** provides robustness that does not emerge by purely learning from demonstrations.

We also compare **RialTo** against behavior cloning with 50 demonstrations to show that the problem is not simply one of slightly more data. Collecting the 50 demonstrations takes a total time of 1 hour and 45 minutes, which is significantly more than the human effort for **RialTo** for which we collect 15 demos, in 30 minutes, and build the environment in 15 minutes

	Only randomization	Distractors	Disturbances
BC (15 demos)	10 ± 9%	0 ± 0%	0 ± 0%
BC (50 demos)	40 ± 15%	30 ± 16%	20 ± 13%
RialTo (15 demos)	**90** ± 9%	**70** ± 14%	**60** ± 16%

TABLE I
RialTo AND IMITATION LEARNING ON PLACING A BOOK ON THE SHELF.

of active time (see Section VI). Although more data improves the performance of direct imitation learning from 10% to 40%, 0% to 30%, and 0% to 20% for the three different levels of robustness, the results in Table I show that **RialTo** achieves approximately 2.5 times higher success rate than pure BC, despite using less than one third the number of demonstrations and taking less than half of the human supervision's time.

B. Impact of Co-Training with Real-World Data

Next, we assess the benefits offered by co-training with real-world demonstrations during teacher-student distillation, rather than just purely training policies in simulation. We consider the *book on shelf*, *plate on rack*, *mug on shelf*, and *open drawer* tasks (the two first being two of the harder tasks with lower overall performance). The results in Figure 6 illustrate that co-training the policy with 15 real-world demonstrations significantly increases real-world performance on some tasks(3.5x and 2x success rate increase for *book on shelf* and *plate on rack* with disturbances, when comparing co-training on real-world demos against co-training with sim demos) while keeping the same performance on tasks that already have a small sim-to-real gap. Qualitatively, we observe the co-trained policy is more conservative and safer to execute. For instance, the policy without co-training usually comes very close to the plate or the book, occasionally causing it to fall. The policy with co-training data, however, leaves

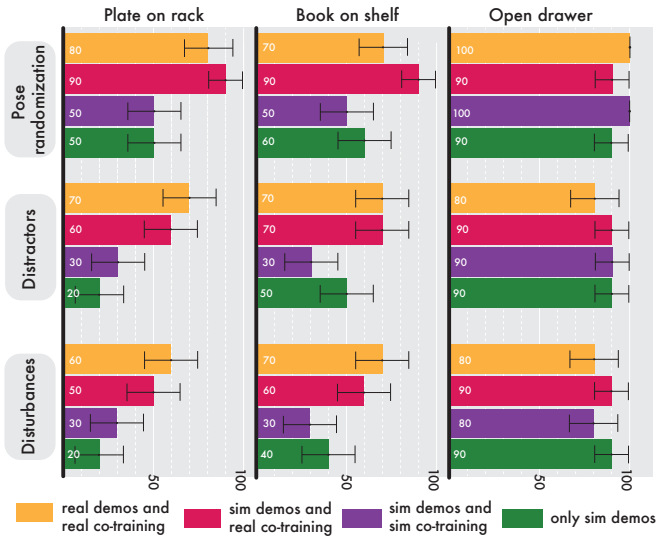

Fig. 6. Comparison between running **RialTo** on sim vs real data.The performance on the methods doing co-training with real-world demos is higher than using only simulated demos or no real-world co-training, on the harder tasks(*plate on rack* and *book on shelf*), and matches the performance in the easier tasks(*open drawer* and *mug on shelf*). Furthermore, starting from real-world or simulated demos does equally well.

more space between the hand and the book before grasping, which is closer to the demonstrated behavior. The observation that sim co-training performs significantly worse than real-world co-training, indicates that co-training with real-world demonstrations is helping in reducing the sim-to-real gap for both the visual distribution shift between simulated and real point clouds and the sim-to-real dynamics gap.

C. Is Real-to-Sim Transfer Necessary?

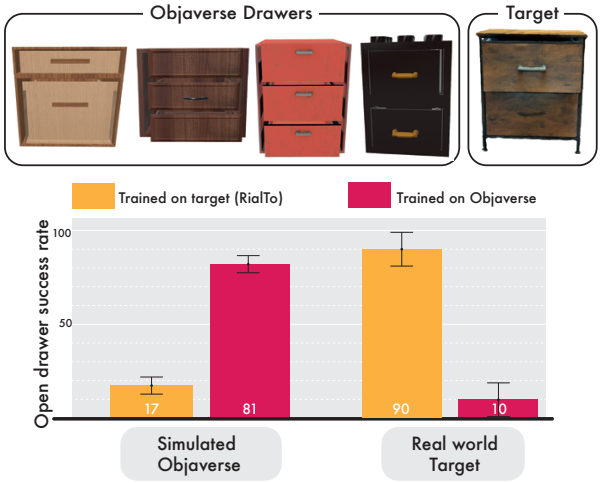

Fig. 7. Comparison between training with **RialTo** on the reconstruction of the target drawer against training on a set of four drawers from Objaverse[17]. We observe, that **RialTo** on the real-to-sim asset does significantly better (90% vs 10%) when testing in the real world on the target drawer compared to training on the set of randomized drawers.

1) Real-to-Sim Transfer of Scenes: Instead of reconstructing assets from the target environment, one could train a policy on a diverse set of synthetic assets and hope the model generalizes to the real-world target scene [12, 21, 65]. While this has shown promising results for object-level manipulation, such as in-hand reorientation [12], it is still an active area of work for scene-level manipulation and rearrangement [21]. Moreover, such methods require significant effort in creating a dataset of scenes and objects that enables the learned policies to generalize. Acquiring a controller that can act in many scenes is also a more challenging learning problem, requiring longer wall clock time, more compute, and additional engineering effort to train a performant policy on a larger and more diverse training set.

To probe the benefits of **RialTo** over such a sim-only training pipeline, we compared the performance against a policy trained using only synthetic assets. Using an amount of time effort roughly comparable to what is required from a single user following our real-to-sim approach (see Section VI), we collected a set of 4 drawers from the Objaverse dataset (see Figure 7). Although this is small compared to the growing size of 3D object datasets, we found it non-trivial to transfer articulated objects into simulation-ready USDs and we leave it as future work. Given these manually constructed diverse simulation scenes, we then trained a multi-task policy using **RialTo** from 20 demonstrations to open the 4 drawers. See Appendix IX-C for the minor modifications to incorporate multi-task policy learning to **RialTo**.

As shown in Figure 7, when evaluating the real target drawer, the policy trained on multiple drawers only achieves a 10% success rate, much lower than the 90% obtained by the policy trained on the target drawer in simulation. This leads us to conclude that to train a generalist agent, considerably more data and effort are needed as compared to the relatively simple real-to-sim procedure we describe for test time specialization. Moreover, this suggests that for performance on particular deployment environments, targeted generation of simulation environments via real-to-simulation pipelines may be more effective than indiscriminate, diverse procedural scene generation.

2) Real-to-sim transfer of policies: We additionally want to understand the impact of transferring policies from real-world demonstrations in comparison to running the pipeline starting with demos collected directly in simulation. This helps analyze whether instead of collecting demos both in simulation and in the real world (for the co-training) we can simply collect demos in the real world and do all the training with those.

Figure 6 shows the real-world performance of policies trained using **RialTo** when starting the RL fine-tuning step using real-world demonstrations as explained in III-C1 against using demonstrations provided directly in simulation. We observe that the performance for both cases is very close. These results show that **RialTo** successfully learns policies with demonstrations from either source of supervision as long as we keep co-training the policies with real-world data in the teacher-student distillation step. Firstly, this indicates that we

	Pose Randomization	Distractors
RialTo without distractor training	60 ± 15%	30 ± 15%
RialTo with distractor training	**100** ± 0%	**70** ± 15%

TABLE II
REAL-WORLD PERFORMANCE OF POLICIES TRAINED WITH AND WITHOUT DISTRACTORS ON THE TASK OF PLACING A MUG ON A SHELF.

do **not** need to collect both demos in sim and real, but we can run **RialTo** uniquely from the demos in the real world. Furthermore, this flexibility is a strength of our pipeline, as the ease of acquiring different sources of supervision may vary across deployment scenarios – i.e., one could use policies pretrained from large-scale real-world data or obtain data from a simulation-based crowdsourcing platform.

D. Scaling *RialTo* to In-the-Wild Environments

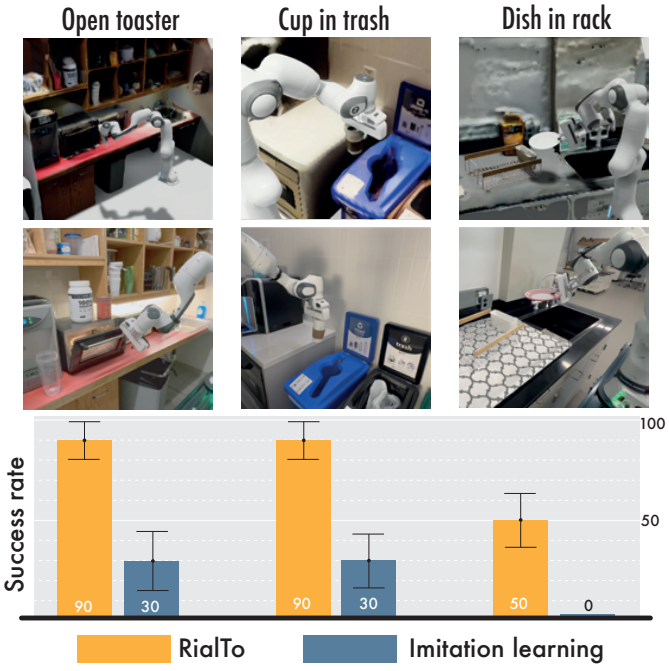

Fig. 8. We test **RialTo** on uncontrolled and in-the-wild scenes, and we see we can continue to solve a variety of tasks more robustly than imitation learning techniques.

In this section, we scale up **RialTo** to more uncontrolled and in-the-wild environments. We test **RialTo** on three different tasks: *open the microwave in a kitchen* (also shown in Section IV-A), *put a cup in the trash*, and *bring the plate from the sink to the dishrack*. We observe that **RialTo** scales up to these more diverse scenes and continues to perform significantly better than standard imitation learning techniques. In particular, **RialTo** brings on average a 57% improvement upon standard imitation learning, see Fig 8.

V. FURTHER ANALYSIS AND ABLATIONS

A. Training with Distractors

When performing teacher-student distillation we performed randomization with additional visual distractors to train a more robust policy that succeeds even in visual clutter. We analyze how this affects the final robustness of the learned policy. For the sake of analysis, we consider the performance on the *mug on the shelf* task. The small size of the mug and its resemblance in shape and size to other daily objects make the visual component of this task particularly challenging when other objects are also present. Our findings in Table II show that adding distractors during training increases the success rate from 30% to 70% when testing the policy in environments with distractors. We also observe a performance improvement in setups with no distractors suggesting that such training also supports better sim-to-real policy transfer.

B. Comparison to RL from Scratch

We hypothesize two key advantages of incorporating demonstrations in the finetuning process: (1) aiding exploration, and (2) biasing the policy toward behaviors that transfer well to reality. Results in Table III show that training from PPO from scratch fails (0% success) in three out of five tasks and much poorer performance in the other two tasks. On tasks with non-zero success, we observed that the policy exploits simulator inaccuracies and learns behaviors that are unlikely to transfer to reality. (see Appendix Fig. 15). For example, the PPO policy opens the toaster by pushing on the bottom of the toaster, leveraging the slight misplacement of the joint on the toaster. Such behaviors are unsafe and would not transfer to reality, underlining the importance of using demonstrations during policy robustification.

C. RL from Vision

RialTo's "inverse distillation" procedure to a compact state-space adds some methodological overhead to the system when compared to the possibility of doing RL fine-tuning directly on visual observations. However, as reported in Appendix Fig. 14, on the task of drawer opening, RL from compact states achieves a 96% success rate after 12 hours of wall-clock time, while RL from vision only achieves a 1% success rate after 35 hours. Hence, inverse distilling to state space is necessary because training RL from vision with sparse rewards is prohibitively slow, motivating the methodology outlined in Section III-C1.

VI. USER STUDY

We analyzed the usability of **RialTo**'s pipeline for bringing real-world scenes to simulation. We ran a user study over 6 people, User 6 being an expert who used the GUI before and Users 1-5 never did any work on simulators before. Each participant was tasked with creating an articulated scene using the provided GUI. More precisely, their task was to: 1) scan a big scene, 2) cut one object, 3) scan and upload a smaller object, and 4) add a joint to the scene. From Figure 9, we found that the average total time to create a scene was 25 minutes and 12 seconds of which only 14 minutes and 40 seconds were active work. We also observed that the expert user accomplished the task faster than the rest, and twice as fast as the slowest user. This indicates that with practice, our

	Open toaster	Book on shelf	Plate on rack	Mug on shelf	Open drawer
RL from scratch with 0 demos	62 ± 2%	0 ± 0%	2 ± 0%	0 ± 0%	0 ± 0%
RL fine-tuning from 15 real demos	91 ± 1%	**90** ± 1%	**81** ± 2%	**81** ± 2%	**96** ± 1%
RL fine-tuning from 15 sim demos	**96** ± 1%	**89** ± 1%	**82** ± 2%	**82** ± 2%	**95** ± 1%

TABLE III
COMPARISON OF TRAINING RL FROM SCRATCH AGAINST RL FROM REAL AND SIM DEMOS. RL FROM SIM AND REAL DEMOS SEEM TO BE EQUIVALENT IN MOST CASES, BUT RL FROM SCRATCH BARELY SOLVES THE TASK.

GUI allows users to become faster at generating scenes. We conclude that doing the real-to-sim transfer of the scenes using the proposed GUI seems to be an intuitive process that is neither time nor labor-intensive when compared to collecting many demonstrations in the real world. We provide more details about the study in Appendix XIII.

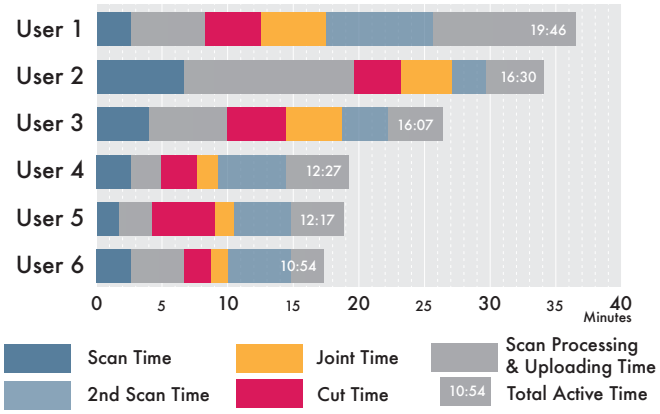

Fig. 9. 3D reconstruction GUI's user study breakdown times. On average it takes 14 minutes and 40 seconds of active time or 25 minutes and 12 seconds of total time to create a scene through our proposed pipeline.

VII. LIMITATIONS AND CONCLUSION

Limitations: While our use of 3D point clouds instead of RGB enables easier sim-to-real transfer, we require accurate depth sensors that can struggle to detect thin, transparent, and reflective objects. Future work may investigate applying **RialTo** to train policies that operate on RGB images or RGBD, as our framework makes no fundamental assumptions that prevent using different sensor modalities. We are also limited to training policies for tasks that can be easily simulated and for real-world objects that can be turned into digital assets. Currently, this is primarily limited to articulated rigid bodies, but advancements in simulating and representing deformables should allow our approach to be applied to more challenging objects. Even though we show **RialTo** works on fast controllers, these are still relatively slow to minimize the sim-to-real gap in dynamics, thereafter there is potential to investigate tasks for which faster controllers are needed. In this work, we consider relatively quasistatic problems, where exact identification of physics parameters is not necessary for the constructed simulation. This will become important as more complex environments are encountered. Finally, as we explain in Section XIV, **RialTo** currently takes around 2 days of wall-clock time end-to-end to train a policy for each task, this time bottleneck makes continual learning infeasible and understanding how to obtain policies faster with minimal human supervision would be valuable. We expect with more efficient techniques for learning with point clouds and better parallelization, this procedure can be sped up significantly.

Conclusion: This work presents **RialTo**, a system for acquiring policies that are robust to environmental variations and disturbances on real-world deployment. Our system achieves robustness through the complementary strengths of real-world imitation learning and large-scale RL on digital twin simulations constructed on the fly. Our results show that by importing 3-D reconstructions of real scenes into simulation and collecting a small amount of demonstration data, non-expert users can program manipulation controllers that succeed under challenging conditions with minimal human effort, showing enhanced levels of robustness and generalization.

ACKNOWLEDGMENTS

The authors would like to thank the Improbable AI Lab and the WEIRD Lab members for their valuable feedback and support in developing this project. In particular, we would like to acknowledge Antonia Bronars and Jacob Berg for helpful suggestions on improving the clarity of the manuscript, and Marius Memmel for providing valuable insights on learning from point clouds in the early stages of the project. This work was partly supported by the Sony Research Award, the US Government, and Hyundai Motor Company.

Author Contributions

Marcel Torne conceived the overall project goals, investigated how to obtain real-to-sim transfer of scenes and policies, and robustly do sim-to-real transfer of policies, wrote all the code for the policy learning pipeline and **RialTo**'s GUI for real-to-sim transfer of scenes, ran simulation and real-world experiments, wrote the paper, and was the primary author of the paper.

Anthony Simeonov helped with setting up the robot hardware, made technical suggestions on learning policies from point clouds, helped with the task of placing the plate on the rack, and actively helped with writing the paper.

Zechu Li assisted in the early stage of conceiving the project and helped develop **RialTo**'s GUI for the real-to-sim transfer of the scenes.

April Chan led the user study experiments to analyze **RialTo**'s GUI.

Tao Chen provided valuable insights and recommendations on sim-to-real transfer.

Abhishek Gupta was involved in conceiving the goals of the project, assisted with finding the scope of the paper, suggested baselines and ablations, played an active role in writing the paper and co-advised the project.

Pulkit Agrawal suggested the idea of doing real-to-sim transfer of scenes, was involved in conceiving the goals of the project, suggested baselines and ablations, helped edit the paper, and co-advised the project.

References

[1] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39 (1):3–20, 2020.

[2] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. *arXiv preprint arXiv:2302.02948*, 2023.

[3] Max Balsells, Marcel Torne, Zihan Wang, Samedh Desai, Pulkit Agrawal, and Abhishek Gupta. Autonomous robotic reinforcement learning with asynchronous human feedback. *arXiv preprint arXiv:2310.20608*, 2023.

[4] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. *arXiv preprint arXiv:2305.13301*, 2023.

[5] Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav S Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33(6):1273–1291, 2017.

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

[7] Arunkumar Byravan, Jan Humplik, Leonard Hasenclever, Arthur Brussee, Francesco Nori, Tuomas Haarnoja, Ben Moran, Steven Bohez, Fereshteh Sadeghi, Bojan Vujatovic, et al. Nerf2real: Sim2real transfer of vision-guided bipedal motion skills using neural radiance fields. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9362–9369. IEEE, 2023.

[8] Matthew Chang, Theophile Gervet, Mukul Khanna, Sriram Yenamandra, Dhruv Shah, So Yeon Min, Kavit Shah, Chris Paxton, Saurabh Gupta, Dhruv Batra, et al. Goat: Go to any thing. *arXiv preprint arXiv:2311.06430*, 2023.

[9] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[10] Qiuyu Chen, Marius Memmel, Alex Fang, Aaron Walsman, Dieter Fox, and Abhishek Gupta. Urdformer: Constructing interactive realistic scenes from real images via simulation and generative modeling. In *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition@ CoRL2023*, 2023.

[11] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pages 297–307. PMLR, 2022.

[12] Tao Chen, Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84):eadc9244, 2023.

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

[14] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[15] AR Code. Ar code. *https://ar-code.com/*, 2022.

[16] Matt Deitke, Rose Hendrix, Ali Farhadi, Kiana Ehsani, and Aniruddha Kembhavi. Phone2proc: Bringing robust robots into our chaotic world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9665–9675, 2023.

[17] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023.

[18] Yunhai Feng, Nicklas Hansen, Ziyan Xiong, Chandramouli Rajagopalan, and Xiaolong Wang. Finetuning offline world models in the real world. *arXiv preprint arXiv:2310.16029*, 2023.

[19] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

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

[21] Ran Gong, Jiangyong Huang, Yizhou Zhao, Haoran Geng, Xiaofeng Gao, Qingyang Wu, Wensi Ai, Ziheng Zhou, Demetri Terzopoulos, Song-Chun Zhu, et al. Arnold: A benchmark for language-grounded task learning with continuous states in realistic 3d scenes. *arXiv preprint arXiv:2304.04321*, 2023.

[22] Abhishek Gupta, Justin Yu, Tony Z Zhao, Vikash Kumar, Aaron Rovinsky, Kelvin Xu, Thomas Devlin, and Sergey Levine. Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages

6664–6671. IEEE, 2021.

[23] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE, 2023.

[24] Daniel Ho, Kanishka Rao, Zhuo Xu, Eric Jang, Mohi Khansari, and Yunfei Bai. Retinagan: An object-aware approach to sim-to-real transfer. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10920–10926. IEEE, 2021.

[25] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50 (2):1–35, 2017.

[26] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[27] Stephen James and Andrew J Davison. Q-attention: Enabling efficient learning for vision-based robotic manipulation. *IEEE Robotics and Automation Letters*, 7(2): 1612–1619, 2022.

[28] Stephen James, Kentaro Wada, Tristan Laidlow, and Andrew J Davison. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13739–13748, 2022.

[29] Zhenyu Jiang, Cheng-Chun Hsu, and Yuke Zhu. Ditto: Building digital twins of articulated objects from interaction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5616–5626, 2022.

[30] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[31] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[32] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[33] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[34] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5 (47):eabc5986, 2020.

[35] Yixin Lin, Austin S. Wang, Giovanni Sutanto, Akshara Rai, and Franziska Meier. Polymetis. https://facebookresearch.github.io/fairo/polymetis/, 2021.

[36] Naijun Liu, Yinghao Cai, Tao Lu, Rui Wang, and Shuo Wang. Real–sim–real transfer for real-world robot control policy learning with deep reinforcement learning. *Applied Sciences*, 10(5):1555, 2020.

[37] Zhao Mandi, Homanga Bharadhwaj, Vincent Moens, Shuran Song, Aravind Rajeswaran, and Vikash Kumar. Cacti: A framework for scalable multi-task multi-scene visual imitation learning. *arXiv preprint arXiv:2212.05711*, 2022.

[38] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[39] Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *arXiv preprint arXiv:2205.02824*, 2022.

[40] Marius Memmel, Andrew Wagenmaker, Chuning Zhu, Patrick Yin, Dieter Fox, and Abhishek Gupta. Asid: Active exploration for system identification in robotic manipulation. *arXiv preprint arXiv:2404.12308*, 2024.

[41] Russell Mendonca, Shikhar Bahl, and Deepak Pathak. Structured world models from human videos. *arXiv preprint arXiv:2308.10901*, 2023.

[42] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[43] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 2023.

[44] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

[45] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[46] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[47] NVIDIA. Nvidia isaac-sim. *https://developer.nvidia.com/isaac-sim*, May 2022.

[48] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.

Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[49] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020.

[50] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[51] Polycam. Polycam. *https://poly.cam*, 2020.

[52] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[53] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[54] Kanishka Rao, Chris Harris, Alex Irpan, Sergey Levine, Julian Ibarz, and Mohi Khansari. Rl-cyclegan: Reinforcement learning aware simulation-to-real. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11157–11166, 2020.

[55] Nathan Ratliff, J Andrew Bagnell, and Siddhartha S Srinivasa. Imitation learning for locomotion and manipulation. In *2007 7th IEEE-RAS International Conference on Humanoid Robots*, pages 392–397. IEEE, 2007.

[56] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[57] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547, 2003.

[58] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[59] Idan Shenfeld, Zhang-Wei Hong, Aviv Tamar, and Pulkit Agrawal. Tgrl: An algorithm for teacher guided reinforcement learning. In *International Conference on Machine Learning*, pages 31077–31093. PMLR, 2023.

[60] Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82):eadg1462, 2023.

[61] Priya Sundaresan, Rika Antonova, and Jeannette Bohgl. Diffcloud: Real-to-sim from point clouds with differentiable simulation and rendering of deformable objects. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10828–10835. IEEE, 2022.

[62] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[63] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, et al. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–12, 2023.

[64] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter 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)*, pages 23–30. IEEE, 2017.

[65] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

[66] Lirui Wang, Jialiang Zhao, Yilun Du, Edward H Adelson, and Russ Tedrake. Poco: Policy composition from and for heterogeneous robot learning. *arXiv preprint arXiv:2402.02511*, 2024.

[67] Luobin Wang, Runlin Guo, Quan Vuong, Yuzhe Qin, Hao Su, and Henrik Christensen. A real2sim2real method for robust object grasping with neural surface reconstruction. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–8. IEEE, 2023.

[68] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *arXiv preprint arXiv:1906.03853*, 2019.

[69] Jingyun Yang, Max Sobol Mark, Brandon Vu, Archit Sharma, Jeannette Bohg, and Chelsea Finn. Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning. *arXiv preprint arXiv:2310.15145*, 2023.

[70] Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Jodilyn Peralta, Brian Ichter, et al. Scaling robot learning with semantically imagined experience. *arXiv preprint arXiv:2302.11550*, 2023.

[71] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[72] Allan Zhou, Moo Jin Kim, Lirui Wang, Pete Florence, and Chelsea Finn. Nerf in the palm of your hand: Corrective augmentation for robotics via novel-view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17907–17917, 2023.

[73] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3651–3657. IEEE, 2019.

[74] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.

Next, we provide additional details of our work. More concretely:

- **Task Details** VIII: provides more details on the tasks used to evaluate **RialTo** and the baselines.
- **Implementation Details** IX: provides more detailed information on the exact hyperparameters such as network architectures, point cloud processing, and dataset sizes using in **RialTo**.
- **Further Analysis** XI: we provide further details on **RialTo**, more concretely on running RL from vision, RL from scratch and on the sim-to-real gap.
- **Hardware Setup** X: Details on the robot hardware and cameras used for the experiments.
- **GUI for Real-to-Sim Transfer of Scenes** XII: We provide further details on the GUI that we proposed together with advice on which scanning methods to use for each scenario.
- **GUI User Study** XIII: We explain how we ran the User Study together with visualizations of the scanned scenes.
- **Compute Resources**XIV: We give details on the compute used to run the experiments.

VIII. Task details

In this section of the appendix, we describe additional details about each task. Across tasks, the state space consists of a concatenation of all of the poses of the objects present in the scenes together with the states of the joints and the state of the robot. The action space consists of a discretized end-effector delta pose of dimension 14. More concretely, we have 6 actions for the delta position, which moves ± 0.03 meters in each axis, 6 more actions for rotating ± 0.2 radians in each axis, and 2 final actions for opening and closing the gripper.

As we explain in Section III-B, we define a success function that will be used for selecting successful trajectories in the inverse distillation procedure and as a sparse reward in the RL fine-tuning phase. Next, we specify which are the success functions for each of the tasks:

- **Kitchen Toaster**: $success =$ toaster_joint > 0.65 && condition(gripper_open)
- **Open Drawer**: $success =$ drawer_joint > 0.1 && condition(gripper_open)
- **Open Cabinet**: $success =$ cabinet_joint > 0.1 && condition(gripper_open)
- **Plate on the rack**: $success =$ $||$plate_site $-$ rack_site$||_2 < 0.2$ && rack_y_axis $\cdot$ plate_z_axis > 0.9 && condition(gripper_open)
- **Book on shelf**: $success =$ $||$book_site $-$ shelf_site$||_2 < 0.12$ && condition(gripper_open)
- **Mug on shelf**: $success =$ $||$mug_site $-$ shelf_site$||_2 < 0.12$ && mug_z_axis $\cdot$ shelf_z_axis > 0.95 && condition(gripper_open)
- **Plate on the rack in the kitchen**: $success =$ $||$plate_site $-$ rack_site$||_2 < 0.2$ && rack_y_axis $\cdot$ plate_z_axis > 0.9 && condition(gripper_open)
- **Cup in trash**: $success =$ $||$cup_site$-$trash_site$||_2 < 0.07$ && condition(gripper_open)

A. Simulation details

For simulating each one of the tasks, we use the latest simulator from NVIDIA, IsaacSim [47]. Furthermore, to develop our code we were inspired by the Orbit codebase [43], one of the first publicly available codebases that run Reinforcement Learning and Robot Learning algorithms on Isaac Sim.

Regarding the simulation parameters of the environments, as mentioned in the text, we set default values in our GUI and these are the same that are used across the environments. In more detail, we use convex decomposition with 64 hull vertices and 32 convex hulls as the collision mesh for all objects. These values could vary in some environments, but we have found they are in general a good default value. There is one exception, the dish on the rack task, where the rack needs to be simulated very precisely, in that case, we used SDF mesh decomposition with 256 resolution which returns high-fidelity collision meshes. Note that all these options can be changed from our GUI. Regarding the physics parameters, we set the dynamic and static frictions of the objects to be 0.5, the joint frictions to be 0.1, and the mass of the objects to be 0.41kg. Note that in many of the tasks, we also leverage setting fixed joints on the objects, to make sure these won't move, for example, on the shelf or kitchen.

IX. Implementation details

A. Network architectures

1) State-based policy: As described in Section III-C2, we fine-tune a state-based policy with privileged information in the simulator. This policy is a simple Multi-Layer Perceptron (MLP) with two layers of size 256 each. This takes as input the privileged state from the simulator and outputs a Categorical distribution of size 14 encoding the probabilities for sampling each discrete end-effector action. For our PPO with BC loss implementation, we build on top of the Stable Baselines 3 repository [52]. The network for the value function shares the first layer with the actor. See Table VI for more details.

2) Point cloud policy: For both the inverse distillation procedure (Section III-C1) and the last teacher-student distillation steps (Section III-D) we train a policy that takes as input the point cloud observation together with the state of the robot (end-effector pose and state) and outputs a Categorical distribution of size 14 encoding the probabilities for each action. The network architecture consists of an encoder of the point clouds that maps to an embedding of size 128. Then this embedding is concatenated to the state of the robot (size 9) and is passed through an MLP of size 256,256. Regarding the point cloud encoder, we use the same volumetric 3D point cloud encoder proposed in Convolutional Occupancy Networks [49], consisting of a local point net followed by a 3D U-Net which outputs a dense voxel grid of features. These features are then pooled with both a max pooling layer and an average pooling layer and the resulting two vectors are concatenated to obtain the final point cloud encoding of size 128.

Fig. 10. Overview of the disturbances that **RialTo** is robust to in the different tasks that we evaluated it on.

Task	USD Name	Episode length	Randomized	Position	Position	Orientation	Orientation
Parameters			Object Ids	Min (x,y,z)	Max (x,y,z)	Min (z-axis)	Max (z-axis)
Kitchen toaster	kitchentoaster3.usd	130	[267]	[0.3,-0.2,-0.2]	[0.7,0.1,0.2]	[-0.1]	[0.1]
Plate on rack	dishinrackv3.usd	150	[278, [270,287]]	[-0.4,-0.035,0]	[0,0.25,0]	[-0.52,0]	[0.52,0]
Mug on shelf	mugandshelf2.usd	150	[267,263]	[[-0.3,0,0], [-0.1,0.25,0]]	[[0.25,0.3,0.07], [0.4,0.4,0]]	[-0.52,-0.54]	[0.52, 0.54]
Book on shelf	booknshelve.usd	130	[277, [268,272]]	[[-0.25,-0.12,0], [-0.15,-0.05,0]]	[[0.15,0.28,0], [0.15,0.15,0]]	[-0.52,0]	[0.52,0]
Open cabinet	cabinet.usd	90	[268]	[-0.5,-0.1,0.1]	[0,0.3,-0.1]	[-0.52]	[0.52]
Open drawer	drawerbiggerhandle.usd	80	[268]	[-0.26,-0.07,-0.05]	[0.16,0.27,0]	-0.5	0.5
Cup in trash	cupntrash.usd	90	[263, 266]	[[[-0.2, -0.3, -0.2], [-0.2,-0.12,0]]]	[[[0.2, 0.1, 0.2], [0.2,0.2,0]]]	[0,0]	[0,0]
Plate on rack from kitchen	dishsinklab.usd	110	[[263, 278, 270]]	[[[-0.25, -0.1, -0.1], [-0.1,0.05,0], [-0.2,0,0]]]	[[[0.1, 0.2, 0.1], [0.1,0.15,0], [0,0,0]]]	[0,-0.3,0]	[0,0.3,0]

TABLE IV
SPECIFIC PARAMETERS FOR EACH ONE OF THE TASKS.

B. Teacher-student distillation

Given the state-based policy $\pi_{\text{sim}}(a|s)$ learned in the simulator, we wish to distill it into a policy $\pi^*_{\text{sim}}(a|o)$ that takes the point cloud observation and outputs the action. We take the standard teacher-student distillation approach [32, 12]. The first step consists of doing imitation learning on a set of trajectories given by the expert policy $\pi_{\text{sim}}(a|s)$ rollout. This set of trajectories needs to be carefully designed to build an implicit curriculum so that we can learn the student policy successfully. When designing this dataset of trajectories, we mix 15000 trajectories rendering full point clouds (where all faces of the objects are visible, which is obtained through directly sampling points from the mesh, as proposed in [12]), 5000 trajectories rendered from a camera viewpoint that is approximately the same position as the camera in the real world, a set of 2000 trajectories also generated from the same camera viewpoint in sim but adding distractor objects (see Figure 12), finally, we mix the 15 real-world trajectories. The four different splits in the dataset are sampled equally, with 1/4 probability each.

Task Parameters	Position (x,y,z) Camera	Rotation (quat) Camera	Crop Min Camera	Crop Max Camera	Size Image
Kitchen toaster	[0.0, -0.37, 0.68]	[0.82,0.34,-0.20, -0.41]	[-0.8,-0.8,-0.8]	[0.8,0.8,0.8]	(640,480)
Plate on rack	[0.95,-0.4,0.68]	[0.78,0.36, 0.21, 0.46]	[-0.3,-0.6,0.02]	[0.9,0.6,1]	(640,480)
Mug on shelf	[0.95,-0.4,0.68]	[0.78,0.36, 0.21, 0.46]	[-0.3,-0.6,0.02]	[0.9,0.6,1]	(640,480)
Book on shelf	[0.95,-0.4,0.68]	[0.78,0.36, 0.21, 0.46]	[-0.3,-0.6,0.02]	[0.9,0.6,1]	(640,480)
Open cabinet	[0.95,-0.4,0.68]	[0.78,0.36, 0.21, 0.46]	[-0.3,-0.6,0.02]	[0.9,0.6,1]	(640,480)
Open drawer	[0.95,-0.4,0.68]	[0.78,0.36, 0.21, 0.46]	[-0.3,-0.6,0.02]	[0.9,0.6,1]	(640,480)
Cup in trash	[0.0, -0.37, 0.68]	[0.82,0.34,-0.20, -0.41]	[-1,-1,-1]	[1,1,1]	(640,480)
Plate on rack from kitchen	[0.0, -0.37, 0.68]	[0.82,0.34,-0.20, -0.41]	[-0.8,-0.8,-0.8]	[0.8,0.8,0.8]	(640,480)

TABLE V
CAMERA PARAMETERS FOR EACH TASK.

MLP layers	PPO n_steps	PPO batch size	PPO BC batch size	PPO BC weight	Gradient Clipping
256,256	episode length	31257	32	0.1	5

TABLE VI
STATE-BASED POLICY TRAINING PARAMETERS. THE REST OF THE PARAMETERS ARE THE DEFAULT AS DESCRIBED IN STABLE BASELINES 3[52].

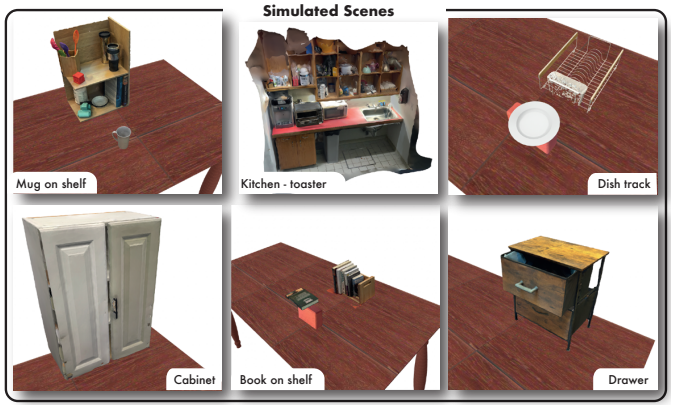

Fig. 11. Overview of the scenes generated using our GUI and used for evaluating **RialTo**.

After this first distillation step, we perform a step of DAgger [56], where we roll out the policy $\pi^*_{\text{sim}}(a|o)$ and relabel the actions with $\pi_{\text{sim}}(a|s)$. In this second and last step, we mix the DAgger dataset with the trajectories with distractors in sim and the real-world trajectories and sample trajectories. Again each dataset is sampled equally with 1/3 probability each.

Finally, the details for generating and randomizing the point clouds are available in Table VII and were largely inspired by [12]. The parameters for training the point cloud-based network are available in VIII.

C. Simulated Assets Baseline Details

To implement the baseline with multiple simulation assets we had to incorporate two modifications for enabling the multi-task policy learning: 1) at each episode we select a drawer randomly from the set of drawers 2) we expand the observation space of the state-based policy to include the index of the drawer selected to open.

Fig. 12. Distractor objects used to get a robust policy to visual distractors in the teacher-student distillation step III-D.

D. Imitation Learning Baseline

For the imitation learning baseline, we collect 15 (unless otherwise specified) real-world demonstrations using a keyboard interface. We preprocess the point clouds in the same manner as for the teacher-student distillation training (see Section IX-B. We complete the point cloud sampling points from the arm mesh leveraging the joints from the real robot. We also add the same randomization: jitter, dropout, and translation.

1) Imitation learning with new assets: We implemented an additional baseline where we added point clouds sampled from different object meshes (see Figure 12) into the real-world point cloud to make the policy more robust to distractors. However, no improvement in the robustness of this baseline was found as seen in Figure IX. We hypothesize that this is the case because the added meshes into the point cloud do not bring any occlusions which is one of the main challenges when adding distractors in point clouds.

X. HARDWARE SETUP

Our experiments are run on two different Panda Franka arms. One is, the Panda Franka arm 2, which is mounted on a fixed table, we run the book on the shelf, mug on the shelf, dish on the rack, open the cabinet, and open the drawer there. Then we also ran part of our experiments, on a Panda

Total pcd points	Sample Arm Points (#)	Dropout ratio	Jitter ratio	Jitter noise	Sample Object Meshes Points	Pcd Normalization	Pcd Scale	Grid Size
6000	3000	[0.1,0.3]	0.3	$\mathcal{N}(0, 0.01)$	1000	[0,0,0] (toaster) [0.35,0,0.4] (others)	0.625 (toaster) 1 (others)	32x32x32

TABLE VII
POINT CLOUD GENERATION AND RANDOMIZATION PARAMETERS.

MLP layers	lr	Optimizer	Batch Size	Nb full pcd traj	Nb simulated pcd traj	Nb simulated pcd traj (distractors)	Nb real traj
256,256	0.0003	AdamW	32-64	15000	5000	1000	15

TABLE VIII
POINT CLOUD TEACHER-STUDENT DISTILLATION PARAMETERS.

	Pose randomization	Distractors	Disturbances
IL	**40** $\pm$ 15%	**50** $\pm$ 17%	**10** $\pm$ 9%
IL with distractors	**50** $\pm$ 17%	20 $\pm$ 13%	**10** $\pm$ 9%

TABLE IX
COMPARISON OF THE PLAIN IMITATION LEARNING BASELINE (IL) AGAINST ADDING NEW DISTRACTORS (IL WITH DISTRACTORS) ON THE TASK OF OPENING THE DRAWER. NO IMPROVEMENT IS OBSERVED.

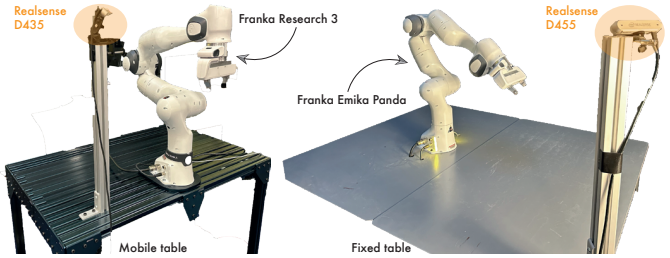

Fig. 13. Overview of the hardware setup used for evaluating **RialTo**. left: used for the kitchen toaster task, right: used for the book on the shelf, mug on the shelf, dish on the rack, open cabinet, and open drawer tasks.

Franka arm 3, mounted on a mobile table, more concretely, the open toaster in the kitchen was the task run on this arm. The communication between the higher and lower level controller of the arm is done through Polymetis [35].

We mount one calibrated camera per setup to extract the depth maps that will be passed to our vision policies. More concretely we use the Intel depth Realsense camera D455 on the first setup and the Intel depth Realsense camera D435 on the second setup. See Figure 13 for more details on the robot setup.

XI. FURTHER ANALYSIS

A. RL from vision

Part of the inefficiency of running RL from vision comes from the increased memory required to compute the policy loss for vision-based RL – on the same GPU, the batch size for vision-based policies is 100x smaller than the batch size used for compact state policies. Rendering point clouds in simulation is also approximately 10x slower than running the pipeline without any rendering. When adding these factors up, RL from vision becomes much slower and practically infeasible given our setup with sparse rewards.

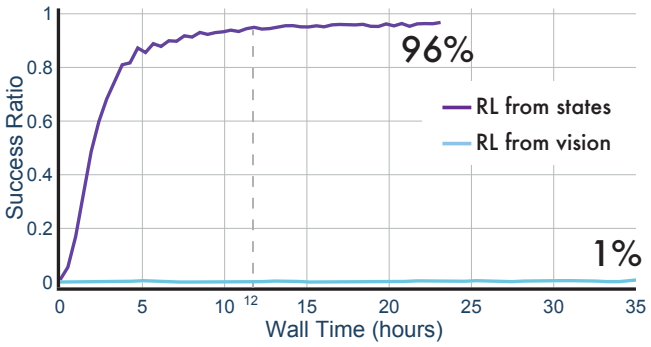

Fig. 14. Wall clock time comparison of running PPO from vision against from compact states.

B. RL from Scratch

In Figure 15, we qualitatively observe the phenomena that we mention in III-C2, where the policy trained from scratch, without demos, exploits the model's inaccuracies. In this specific case, we observe that the policy leverages the slightly incorrectly placed joint to open the microwave in an unnatural way that wouldn't transfer to the real world.

C. RL from different amounts of real-world data

In this section, we analyze further how many real-world demonstrations are needed to successfully fine-tune policies with RL in simulation. We start with 0,5,10,15 real-world demonstrations and inverse-distill the policy by collecting 15 sim trajectories from this real-world trained policy. We observe in table X that for the task of placing a book on the shelf, there is a step function where the PPO has a 0% success rate until 15 demos are used. The reason is that with less than 15 demos

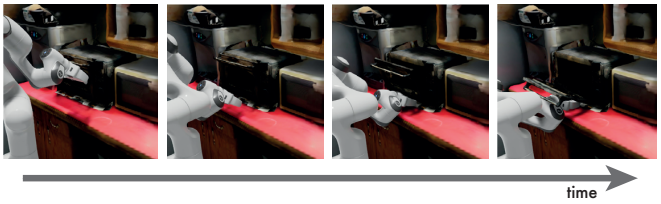

Fig. 15. Visualization of a rollout of the final policy learned with RL without demos and achieving a 62% accuracy on opening the toaster in simulation. We observe the resulting policy that learns without demos exploits the model's inaccuracies, thereafter it will not transfer to the real world.

	Book on shelf	Open drawer
RL fine-tuning from 0 real demos	0 ± 0%	0 ± 0%
RL fine-tuning from 5 real demos	0 ± 0%	89 ± 1%
RL fine-tuning from 10 real demos	0 ± 0%	**96** ± 1%
RL fine-tuning from 15 real demos	**90** ± 2%	**96** ± 1%

TABLE X
COMPARISON OF TRAINING RL FROM DIFFERENT AMOUNTS OF REAL-WORLD DEMOS.

the real-world policy does not transfer to the simulation hence no sim demos can be collected during the inverse distillation procedure. Thereafter the RL fine-tuned policy starts from scratch when using < 15 real-world demos. On the other side, for the easier task of opening a drawer, we observe this step function earlier, where at > 5 demos we can do RL fine-tuning from demos and obtain successful policies.

D. Mixing *RialTo* with synthetic data

We run **RialTo** combining the data from the synthetic assets experiment (see Figure 7) together with the simulated target environment data and study whether we get any performance gain by combining these two sources of data on the task of opening the drawer. We observe in Table XI that there is no clear improvement when combining the simulated assets with the target asset. One reason could be that more synthetic data is needed to observe an increase in performance. The other hypothesis is that learning only on the target environment (**RialTo**) is enough and the 10% left to reach 100% success rate in the real world comes from the sim-to-real gap.

E. *RialTo* Multi-Task

We propose a *multi-task* version of **RialTo**. We train *multi-task* **RialTo** on the tasks of opening a drawer, putting a mug on the shelf, cup in the trash, and dish on the rack environments.

The proposed *multi-task* **RialTo** procedure is the following:

	Pose randomization	Distractors
RialTo	**90** ± 9%	**90** ± 9%
RialTo + synthetic assets	**90** ± 9%	**80** ± 13%

TABLE XI
COMPARISON OF USING **RIALTO** WITH ADDED SYNTHETIC ASSETS AGAINST STANDARD **RIALTO** ON THE TASK OF OPENING THE DRAWER IN THE REAL WORLD. NO IMPROVEMENT IS OBSERVED.

	Open drawer	Mug on shelf
Imitation learning	40 ± 17%	10 ± 9%
RialTo	**90** ± 9%	**100** ± 0%
RialTo multitask	**90** ± 9%	80 ± 15%

TABLE XII
COMPARISON OF TRAINING **RIALTO** ON MULTIPLE TASKS AGAINST SINGLE-TASK **RIALTO**. NO IMPROVEMENT IS OBSERVED.

1) Train separate state-based single-task policies per task
2) Collect trajectories from each one of the tasks with the state-based policies
3) Distill these trajectories into a single multi-task policy conditioned with the task-id
4) Run multiple iterations of DAgger on each task sequentially to obtain a final multi-task policy

We evaluate this policy in the real world on two of the tasks and observe in Table XII that in *opening the drawer*, the performance of multi-task **RialTo** matches single-task (90% success). However, the performance slightly decreases on the *mug on the shelf task* (from 100% on single-task to 80% on multi-task). Nevertheless, the performance is still above the imitation learning baseline (40% for the drawer and 10% for the mug on the shelf). We did not tune any hyperparameters, and we kept the same network size that we used for the **RialTo** experiments. We should be able to bring the performance of the mug on the shelf task to match the single-task policy with some hyperparameter tuning.

We showed that **RialTo** can be easily adapted to train multi-task policies. We hypothesize that we need to train in more environments to obtain multi-task generalization.

F. Sim-to-real gap

We analyze and propose an explanation for the observed sim-to-real gap in Table XIII, where we show the performance of the final point cloud-based policy in both simulation and the real world. We observe that in general, the sim-to-real gap does not seem to be present. In some cases such as for the mug on shelf task, we observe that the performance in simulation is worse than the performance in the real world. The main reason for this disparity is that we want to make the simulation harder than the real-world environment to make sure that we will be able to recover a good robust policy in the real world.

XII. GUI FOR REAL-TO-SIM TRANSFER OF SCENES

In the main text and video, we provide an overview of the features and capabilities of our GUI. Additional valuable features include the ability to populate the scene with assets from object datasets such as Objaverse [17]. This allows for randomizing surrounding clutter and supporting policy training that generalizes to distractor objects (see Section V-A).

1) 3D reconstruction software used: We mainly used 3 different methods/apps for obtaining the 3D meshes from videos:

1) Polycam [51] is used to scan larger scenes, such as the kitchen. Polycam makes effective use of the built-in

	Kitchen toaster	Book on shelf	Plate on rack	Mug on shelf	Open drawer	Open cabinet
Performance in simulation	90 ± 4%	84 ± 5%	80 ± 6%	72 ± 6%	95 ± 3%	92 ± 4%
Performance in the real world	90 ± 9%	90 ± 9%	90 ± 9%	100 ± 0%	90 ± 9%	85 ± 8%

TABLE XIII
COMPARISON OF PERFORMANCE IN SIMULATION (TOP) AND THE REAL WORLD (BOTTOM).

iPhone depth sensor which helps extract realistic surface geometry for large uniform flat surface (e.g., a kitchen counter). However, we find it struggles with fine-grained details. Polycam outputs a GLTF file, which we convert directly to a USD for loading into Isaac Sim using an online conversion tool.

2) AR Code [15] is used to extract high-quality meshes for single objects that can be viewed by images covering the full 360 degrees surrounding the object (e.g., cabinet, mug, microwave, drawer). While AR Code leads to more accurate geometry than Polycam for singulated objects, we still find it struggles on objects with very thin parts. AR Code directly outputs a USD file that can be loaded into Isaac Sim.
3) NeRFStudio [63] is used to capture objects that require significantly more detail to represent the geometry faithfully. For example, AR Code failed to capture the thin metal structures on the dish rack, whereas NeRFs are capable of representing these challenging geometric parts. We use the default "nerfacto" model and training parameters. This method trains a relatively small model on a single desktop GPU in about 10 minutes. After training converges, we use the NeRFStudio tools for extracting a 3D point cloud and obtaining a textured mesh with Poisson Surface Reconstruction [30]. This outputs an OBJ file, which we convert into a USD by first converting from OBJ to GLTF, and then converting from GLTF into USD (with both file conversions performed with an online conversion tool).

XIII. GUI USER STUDY

To test the functionality and versatility of the real-to-sim generation pipeline, we ran a user study over six people, where each participant was tasked with creating an articulated scene using the provided GUI. Every individual was given the same set of instructions that would guide them through the process of constructing a usable and accurate scene. At the start of each trial, the participant was instructed to download Polycam [51], which uses a mobile device's LiDAR to generate 3D models. The user then selected a location and captured their scene by taking a sequence of images. The time required to complete this step was recorded as "Scan Time." Once the images were captured, Polycam needed to process the pictures to transform the scene into a three-dimensional mesh. Once the mesh had been generated, the participant was then instructed to upload the articulated USD to a computer and convert this file into the GLB format (required by our GUI). Finally, the user uploaded

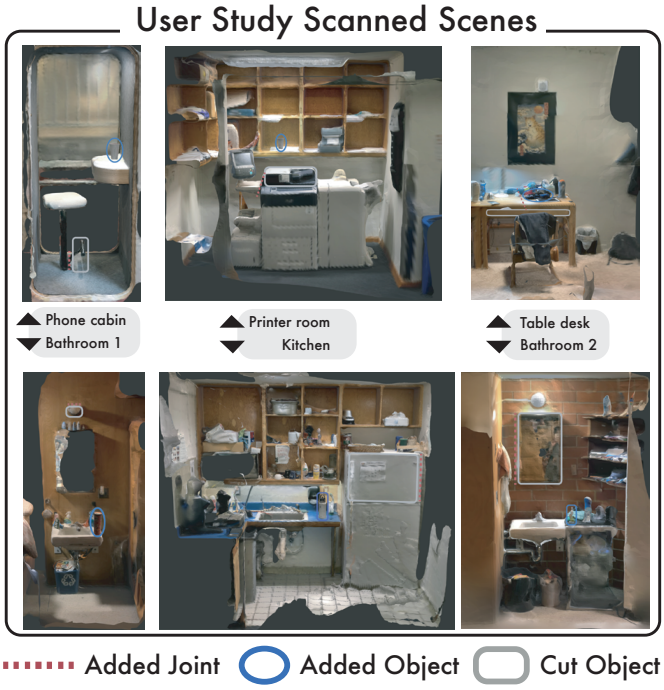

Fig. 16. Overview of the scenes assembled by the Users during the user study, see Section VI.

the GLB file into the provided GUI, and the time required to complete these steps was recorded as "Scan Processing and Uploading Time." Because the uploaded mesh was created using one scan, all objects in the scene are connected, and the user is unable to move a single item without shifting the entire background. Thus, in order to create a more realistic scene, the participant was asked to use the GUI to cut an object out of the scene, allowing this item to be manipulated independently of the background. The time it took for the user to cut this object from the original mesh was regarded as "Cut Time." In an attempt to further the realistic nature of this scene, the participant was then instructed to specify joint parameters and create a fixed joint that would allow an object in the scene to rotate about a specific point. For instance, a fixed joint at a door would allow the door to rotate about its hinge and generate an accurate simulation of door movement. The time required to create a fixed joint in the scene was recorded as "Joint Time." Lastly, to demonstrate the full capabilities of the GUI, the participant was asked to add another object to their current scene. They were instructed to download another 3D scanning application, AR Code [15], which was used to

	Scan	Process + Upload 1st Scan (idle)	Cut	Joint	2nd Scan	Process + Upload 2nd Scan (idle)	Total time	Total active time
User 1	2:25	5:41	4:15	4:56	8:10	10:45	36:12	19:46
User 2	6:30	12:57	3:32	3:51	2:37	4:19	33:46	16:30
User 3	3:52	5:52	4:35	4:14	3:26	4:15	26:14	16:07
User 4	2:34	2:06	2:48	1:41	5:14	4:33	19:06	12:27
User 5	1:32	2:33	4:43	1:28	4:34	3:50	18:40	12:17
User 6	2:30	3:52	2:08	1:17	4:59	2:26	17:12	10:54

TABLE XIV
DETAILED TIME SPENT BY EACH USER IN THE USER STUDY, SEE SECTION VI.

create the three-dimensional mesh of the additional object. The time required to generate this mesh was recorded as "Scan Time (2)." Then the participant again converted their mesh to GLB format and uploaded this file to the same GUI. Once uploaded, the object was placed in a realistic position within the scene, and the time elapsed during this step was added to the "Scan Processing and Uploading Time" category. Through this user study, we found that it took an average of 14.67 active minutes (excluding the "Scan Processing and Uploading Time" category) to create a scene that included one cut object, one fixed joint, and one additional object. However, it is important to note that User 6 had previous experience using this GUI, while all other users had no experience. Thus, if we disregard the results of User 6, we find the average time to create a scene to be 15.42 active minutes, which is not a significant difference. As a result, the real-to-sim transfer using the provided GUI seems to be an intuitive process that is neither time nor labor-intensive.

User 1 took the longest time to complete this series of tasks mostly due to their extensive upload period. Because User 1 scanned their environment for a lengthy period, their articulated USD file was larger than all other users. As a result, it took longer for them to upload their file to a computer and convert this file to GLB format. The abnormal size of User 1's file coupled with their difficulty operating the file conversion website led to a lengthy Scan Processing and Upload Time, which led to the slowest overall performance.

User 2 was the only user who was sent instructions digitally and completed the tasks remotely. An individual experienced with the real-to-sim pipeline was present for all other trials. Thus, this may have contributed to User 2's longer completion time, as their questions had to be answered remotely. However, User 2 did not have trouble with any particular section of the pipeline but rather took a longer time to complete each section.

User 3's experience with the real-to-sim pipeline went smoothly, as there were no obvious difficulties while scanning, uploading, or using the GUI. They followed the instructions quickly and precisely, resulting in a better completion time than Users 1 and 2.

Users 4 and 5 completed all tasks in the pipeline more quickly than User 3 because the background they chose was smaller with fewer details. Thus, they were able to scan their scenes faster, generating a smaller file that was able to be processed, uploaded, and converted more quickly. However, their speed did reduce the quality of their backgrounds, since the details in both scans are not as precise as the others. Thus, it seems User 3 completed the tasks quickly with the most accurate scan.

User 6 had previous experience with the real-to-sim pipeline, so they were able to use this expertise to quickly complete the tasks. The only abnormality with User 6's trial was their longer Scan Time for object 2. They had trouble with the "AR code" app during this trial, resulting in a longer Scan Time (2).

A. Scaling laws of the RialTo GUI

$$\begin{aligned} total\_active\_time = t_{\textit{scan scene}} \\ + t_{\textit{scan object}} \cdot N_{\textit{objects}} \\ + t_{\textit{cut object}} \cdot N_{\textit{cut objects}} \\ + t_{\textit{add joint}} \cdot N_{joints} \end{aligned} \tag{3}$$

We derive a relation to express the total active time needed to create a scene with respect to the number of joints and objects there are in the scene. The total active time to create a scene increases linearly in complexity with the number of objects and joints present in the scene, as seen in Relation 3. We define $N_{\textit{objects}}$ as the number of scanned objects that we want to add, $N_{\textit{cut objects}}$ as the number of objects that we want to extract from the scanned scene, N_{joints} as the number of joints the scene has. Taking the average times from our user study (see Table XIV) we find $t_{\textit{scan object}} = 4:50$, $t_{\textit{scan scene}} = 3:14$, $t_{\textit{add joint}} = 2:54$, $t_{\textit{cut object}} = 3:40$. Note that these values are on the conservative side since only one user was an expert, and with increased expertise, these coefficients become smaller.

XIV. COMPUTE RESOURCES

We run all of our experiments on an NVIDIA GeForce RTX 2080 or an NVIDIA GeForce RTX 3090. The first step of learning a vision policy from the real-world demos and collecting a set of 15 demonstrations in simulation takes an average of 7 hours. The next step of RL fine-tuning from demonstrations takes on average 20 hours to converge. Finally,

the teacher-student distillation step takes 24 hours between collecting the trajectories, distilling into the vision policy, and running the last step of DAgger. This adds up to a total of 2 days and 3 hours on average to train a policy for a given task.