Causal Reasoning in Simulation for Structure and Transfer Learning of Robot Manipulation Policies

Timothy E. Lee, Jialiang (Alan) Zhao, Amrita S. Sawhney, Siddharth Girdhar, and Oliver Kroemer

Abstract— We present CREST, an approach for causal reasoning in simulation to learn the relevant state space for a robot manipulation policy. Our approach conducts interventions using internal models, which are simulations with approximate dynamics and simplified assumptions. These interventions elicit the structure between the state and action spaces, enabling construction of neural network policies with only relevant states as input. These policies are pretrained using the internal model with domain randomization over the relevant states. The policy network weights are then transferred to the target domain (e.g., the real world) for fine tuning. We perform extensive policy transfer experiments in simulation for two representative manipulation tasks: block stacking and crate opening. Our policies are shown to be more robust to domain shifts, more sample efficient to learn, and scale to more complex settings with larger state spaces. We also show improved zero-shot sim-to-real transfer of our policies for the block stacking task.

I. INTRODUCTION

Real-world environments, such as homes and restaurants, often contain a large number of objects that a robot can manipulate. However, usually only a small set of objects and state variables are actually relevant for performing a given manipulation task. The capability of reasoning about what aspects of the state space are relevant for the task would lead to more efficient learning and greater skill versatility.

Current approaches to learning versatile manipulation skills often utilize simulation-to-real (sim-to-real) transfer learning [1]–[4], wherein the skill is learned in a simulation and then deployed and fine-tuned (if feasible) on the real robot. Sim-to-real learning is often combined with domain randomization (DR) [5], which involves training the skill on a wide range of task instances in simulation such that the resulting skill is more robust and generalizes across task variations. However, for scenes with distractor objects, the policy still takes the irrelevant state features as inputs. Larger domain shifts in the irrelevant features can therefore still be detrimental to the performance of the skill policy. Rather than relying only on DR, the robot can use model-based reasoning to identify the "structure" of a policy — the interplay between relevant state inputs and control outputs.

In this paper, we propose using causal reasoning to improve sim-to-real transfer through conducting interventions in simulation to determine which state variables are relevant for the successful execution of a task using a given controller. We refer to the resulting algorithm as **CREST**: **C**ausal **R**easoning for **E**fficient **S**tructure **T**ransfer. The relevant state

variables from CREST are used to construct policies that encode the causal structure of the manipulation task. These policies are initially trained in simulation using domain randomization over only the states that have explicitly been determined as relevant. Moreover, policies that only use relevant state variables require significantly fewer parameters and, by construction, are robust to distribution shifts in irrelevant state spaces. In this manner, our approach produces lightweight policies that are designed for efficient online adaptation to unforeseen distribution shifts that may occur when bridging the sim-to-real gap. This contrasts with existing sim-to-real approaches that train large policies over enormous state spaces, where the costs for achieving robust zero-shot transfer may be intractable.

Our proposed approach was successfully evaluated on block stacking and crate opening tasks. Although our method is intended for sim-to-real transfer, we primarily conduct our experiments by transferring to task simulations in NVIDIA Isaac Gym [6], a high-fidelity physics simulator, as a proxy for real systems. This is necessary to experimentally evaluate distributions shifts that would be intractable to evaluate (but nonetheless feasible) in practical manipulation scenarios. Additionally, we validate our approach for zero-shot, sim-to-real performance for the block stacking task.

The contributions of our work are as follows.

- We propose CREST, an algorithm that uses causal reasoning in simulation to identify the relevant input state variables for generalizing manipulation policies.
- We propose two neural network architectures that are constructed using the causal information from CREST.
- We conduct rigorous transfer learning experiments to demonstrate these policies generalize across task scenarios, scale in relevant state complexity, and are robust to distribution shifts.
- We propose CREST as one approach to a broader methodology of structure-based transfer learning from simulation as a new paradigm for sim-to-real robot learning, i.e., *structural* sim-to-real.

II. RELATED WORKS

Our work relates to research areas in robotics and machine learning for structure-based learning, causality, attention, and sim-to-real transfer of higher-level policies.

Structure-based learning, causality, and attention. Causal reasoning for structure, transfer, and reinforcement learning is an emerging area of research [7], [8], having been demonstrated for transferring multi-armed bandits policies [9], examining distribution shifts via imitation learn-

ing [10], modeling physical interactions from videos [11], and clustering causal factors [12]. Causality reasons about the data generation process with respect to an underlying model [13], which our CREST policies encode. The motivation of our approach to transfer learned causal structure is similar to that described by [14]. In our work, we represent this structure through the policy inputs, and we demonstrate the approach achieves sim-to-real transfer.

Our view of policy structure can be seen as an explicitly encoded form of state space attention, achieved via construction of policies using only relevant inputs. We are primarily concerned with object feature states, which enable significantly smaller neural networks to be constructed under our assumptions. As a comparison, [15] learns implicit attention to task-relevant objects to generalize manipulation skills. This approach requires a few example trajectories to be provided and uses a vision-based state representation, making explicit reasoning about states more challenging.

Similar to [16], our approach can also generalize policies to unforeseen dynamic distribution shifts; ours does so primarily through more efficient fine-tuning.

Our work is also similar in spirit to the work of Nouri and Littman [17] in terms of achieving dimensionality reduction for reinforcement learning. Whereas our work seeks to reduce the dimensionality for the state's possible influence on the task policy under different contexts, they instead demonstrate dimensionality reduction in the action space. Kolter and Ng [18] as well as Parr et al. [19] approximate the value function by learning the relevant basis functions.

Sim-to-real and higher-level policies. Our work is a form of simulation-to-reality transfer of controllers [1], [5], [20], [21]. Unlike in the typical sim-to-real paradigm, our policies are not required to transfer zero-shot. Rather, the dimensionality reduction afforded by transferring the relevant state space with CREST enables more efficient online adaptation. Unlike [22], we do not transfer any target samples back to the internal model. Similar to the problem settings of [23]–[25], our agent predicts the best parameters for a controller.

III. PROBLEM FORMULATION

We formulate our problem as a multi-task reinforcement learning problem, wherein a policy π is learned to complete a series of tasks $\mathcal{T}_i$. Each task is modeled as a Markov decision process (MDP). The state space $\mathcal{S}$ and action space $\mathcal{A}$ are the same across tasks. However, each task $\mathcal{T}_i$ defines a separate initial state $s_0(c_i)$, transition function $p(s'|s, a, c_i)$, and reward function $r(a, s, c_i)$ that are parameterized by the task's context variables $c_i \in \mathcal{C}$. We assume that the robot has an internal model $p_{\text{int}}(s'|s, a, c_i)$ that approximates the transition function of the target task domain $p(s'|s, a, c_i)$. The context variables capture object parameter variations, such as shapes, appearances, and initial states. We assume that the context variables are always set such that the task is feasible.

To solve the tasks, the robot learns a policy $\pi(a|s, c)$ that is decomposed into two parts [26]: an upper policy $\pi(\theta|c)$ and a lower control policy $\pi_\theta(a|s)$, where $\theta \in \mathbb{R}^d$ are the controller parameters. Transferring the parameters from the internal

model to the target domain may require fine-tuning. At the start of each task, the upper policy, which is responsible for generalizing between different task contexts, selects a set of parameters for the control policy to use throughout the task execution. We use multilayer perceptron (MLP) neural networks for the upper policy. In principle, the control policies can take on a variety of parameterized forms, such as motor primitives, planners, waypoint trajectories, or linear feedback controllers, depending on the task. We assume a control policy with known preconditions is available.

Of the context variables c that describe the object variations in the scene, only a subset may be relevant for the policy. We refer to this subset of relevant variables as $\tau \subseteq c$, such that the robot can learn a policy $\pi(\theta|\tau)$ to complete the tasks. Our goal is to determine τ through causal reasoning with the internal model, yielding policies with fewer input variables as compared to naively using all of c.

The set of controller parameters can be divided into individual parameters $[\theta]_j \forall j \in 1, ..., d$, where $[\theta]_j$ indicates the jth element of vector θ. Each of these parameters may rely on a different set of relevant variables. We can thus further divide the problem into determining a set of context variables τ_j for each policy parameter $[\theta]_j$, such that the robot can learn a partitioned task policy $\pi([\theta]_j|\tau_j) \forall j \in 1, ..., d$.

IV. CAUSAL STRUCTURE LEARNING

CREST uses an internal model of the task to learn the relevant context τ and parameter-specific mappings τ_j that define the structure of the upper policy (c.f, Sec. V-A).

A. Internal Model for Causal Reasoning

Our approach assumes an *internal model*, an approximate simulation of the task, is available. Analogous to mental models and approximate physics models [27], [28], the internal model facilitates reasoning about the effects of different context variations Δc on completing the task with policy parameters θ. Varying the relevant context parameters τ will affect the execution and outcome of the task in the internal model while irrelevant ones will not.

Given its approximate nature, the solution obtained in the internal model is not necessarily expected to transfer zero-shot to reality. Instead, the internal model provides both an estimate of the policy structure and a task-specific initialization via network pretraining. Intuitively, it is easier to reason about *which* variables are important for a model rather than exactly characterizing the exact model itself. For example, the internal model can capture that the weight of an object affects the required pushing force, but the exact details of frictional interaction may be approximated for the purposes of pretraining the policy. We assume that the internal model approximates the task sufficiently well and includes all context variables c that vary in the target domain. Given the existing challenges in causal representation learning [29], we additionally assume the representation of c is amenable for determining the underlying model for θ via causal interventions [13].

B. Causal Reasoning to Determine Relevant Contexts

At its core, CREST uses simulation-based causal reasoning to determine the relevant context variables for the policy input. This process is divided into two phases: 1) determining the overall set of relevant variables τ, and 2) determining the relevant variables τ_j for specific policy parameters $[\theta]_j$.

Causal interventions to determine relevant variable set.
In this phase, the relevance of a state variable is determined by posing the following question: "If a context variable were different, would the same policy execution still complete the task successfully?" To answer this, we first uniformly sample a context $c_i \sim p(c)$ and solve the resulting task in simulation to acquire the corresponding policy parameters θ_i. In practice, the policy parameters are optimized using Relative Entropy Policy Search [30]. Importantly, the policy is solved for only this specific context c_i (not the general policy).

Given the solved task, we conduct interventions Δc to determine if the policy parameters θ_i remain valid for the new context $c_i + \Delta c$. Interventions Δc are conducted to only alter one context variable at a time, i.e. $||\Delta c||_0 = 1$. If the policy subsequently failed, the intervened variable is considered relevant and thus appended to τ. If the policy succeeded, despite the intervention, then the variable is considered irrelevant and is not required for the general policy. The resulting relevant variable set τ is sufficient for constructing Reduced MLP policies (c.f., Sec. V-A).

Causal interventions to determine individual policy mappings. Reducing the set of state variables from the full set c to the relevant set τ may greatly reduce the size of the policy input. However, for some problems, each of the policy parameters may only depend on a subset of τ. Therefore, in the next phase, the individual mappings from the relevant state variables to the controller parameters are determined by posing the question: "Does altering this relevant context variable require this policy parameter to be changed?"

We begin from the previous phase, where c_i with solution θ_i is available with interventions Δc. In this phase, interventions are only applied to relevant context variables τ.

For each new context $c_i + \Delta c$, the task is solved to obtain the resulting policy parameters $\theta_i + \Delta \theta$, starting the optimization from the original parameters θ_i. The optimization will often alter all of the policy parameters, i.e., all elements of $\Delta \theta$ are non-zero, and the magnitude of their changes are not reliable estimators of their importance. Instead, a solution that minimizes the number of non-zero changes, i.e., $\min ||\Delta \theta||_0$, is obtained using search. This search involves setting subsets of elements in $\Delta \theta$ to zero and evaluating if the resulting policy still solves the task with context $c_i + \Delta c$. In our experiments, we used a breadth-first search to find a solution with a minimal set of parameter changes. Once a subset of parameter changes has been found, the context variable intervened on in Δc is added to the sets τ_j of relevant inputs for the policy parameters with non-zero elements in the final $\Delta \theta$. The output of this phase (the parameter-specific variables τ_j) can then be used to learn Partitioned MLP policies (c.f., Sec. V-A).

TABLE I: CREST evaluation for a toy environment on an aggregate ("Agg.") and mapping-specific ("Map.") basis. Accuracy ("Acc.") is whether all relevant states were detected. False positives ("F.P.") are states that were incorrect detected as relevant. 100 trials are used.

Class	Dim.	Noise	Agg. Acc.	Agg. F.P.	Map. Acc.	Map. F.P.
Linear	8	None	1.00	0.00	0.98	0.19
Nonlinear	8	None	1.00	0.00	0.97	0.23
Linear	20	None	1.00	0.00	0.99	0.53
Nonlinear	20	None	1.00	0.00	0.97	0.24
Linear	8	Limited	1.00	0.23	0.99	0.32
Nonlinear	8	Limited	1.00	0.13	0.95	0.22
Linear	20	Limited	1.00	0.22	0.98	0.50
Nonlinear	20	Limited	1.00	0.12	0.96	0.23

C. CREST Evaluation

Although we primarily use CREST for manipulation policies, we quantify the accuracy of CREST in a limited, application-agnostic manner. Table I shows the results of CREST on an environment that replicates the causal structure of hierarchical manipulation policies. This environment is designed so that, given a set of ground truth mappings between context variables and policy parameters, a controller with randomized structure is generated for the agent to "manipulate" the environment to a goal location determined by the causal structure of the problem. These mappings were either linear or (weakly) nonlinear.

Under the assumptions of our controller and the context c, CREST excels at determining whether a variable is relevant. This is expected by construction of the underlying mathematics of this environment and represents an expected upper bound. We choose action space dimension of 8 and 20 (larger than our transfer learning experiments, c.f., Sec. VI) and select the state space accordingly to permit calculation of ground truth for testing. We also introduce action noise to test the robustness to uncertainty from interventions, leading to more variables detected. Relatively higher false positive rates arise in higher dimensions, but the overall reduction to the relevant set of variables is nonetheless significant.

V. POLICY LEARNING AND TRANSFER

Given the relevant context from CREST, the structure and transfer learning pipeline of our work begins through 1) constructing the appropriate policy; 2) pretraining using the internal model; and 3) fine-tuning in the target setting.

A. Policy Architectures

Given the full context c, the reduced context τ, and the parameter-specific contexts τ_j, the three MLP-based network architectures shown in Fig. 1 are constructed and trained using actor-critic approaches [31].

The baseline $\pi(\theta|c)$ uses a standard MLP network. The approach $\pi(\theta|\tau)$ uses a Reduced MLP (RMLP): an MLP network where the inputs are reduced to only the relevant context τ. The approach $\pi([\theta]_j|\tau_j)$ has independent sets of fully connected layers for each policy parameter θ_j, but with potentially overlapping inputs depending on τ_j. This Partitioned MLP (PMLP) network represents the structural causal model [13] for each θ_j with τ_j as parent variables.

To provide a fair comparison, we choose the weights for the PMLP according to heuristics and multiply the number
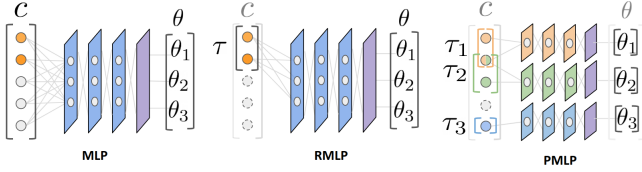
Fig. 1: A visualization of the different policy types. CREST is used to construct both the Reduced MLP (RMLP) and Partitioned MLP (PMLP). The baseline MLP is also shown for comparison. The relevant states are also used for the critic portion of the networks (only the actor portion is shown). The notation used is $[w_1,...,w_d]$, specifying the hidden units and depth of both the actor and critic.

of hidden units by the size of the action space to size the RMLP and MLP. We originally sized the PMLP network according to [32] to provide theoretical guarantees regarding function approximation for one-element outputs (i.e., each θ_j), but the resulting network size for the baseline MLP was intractable to train. All neural network weights are randomly initialized per orthogonal initialization [33] using $\sqrt{2}$ and 0.01 for the scale terms of the hidden layers and output layers, respectively. All networks use tanh activations.

B. Network Training and Transfer

For both the internal model and target domain, we train our policies using PPO [34] with Stable Baselines [35]. First, each network is pretrained with the internal model until the task family is solved. Then, we transfer the network weights to the target domain and evaluate the policy to determine whether the policy transfers zero-shot. Otherwise, fine-tuning is performed. Although freezing network layers has been explored for fine-tuning control policies [36], we permit the entire network to adapt because of the approximate nature of pretraining with the internal model. The learned policy is considered to have solved the task family if it successfully achieves a predefined reward threshold on 50 validation tasks; this evaluation occurs after each policy update.

VI. EXPERIMENTAL RESULTS

We evaluate how CREST can construct policies with greater (target) sample-efficiency and robustness for the robot manipulation tasks of block stacking and crate opening (Fig. 2). Our experiments for each task follow the structure and transfer learning pipeline motivating our approach: 1) use CREST to determine the causal structure of the task; 2) construct and pretrain policies with this structure; and 3) transfer and fine-tune these policies in the target domain. The target domains include manipulation tasks in NVIDIA Isaac Gym, enabling rigorous investigation of representative distribution shifts that may occur when deploying sim-to-real policies. For the block stacking task, we additionally leverage a real robot system to assess sim-to-real transfer.

Target simulation and training was conducted using a NVIDIA DGX-1. Pretraining was done using a NVIDIA GeForce RTX 2080. Samples from the block stacking and crate opening internal models were 400 and 65 times faster to obtain than target simulation samples, respectively. This is consistent with the concept of the internal model as a cheap, approximate simulator, whereas samples from the target domain are costly and therefore desirable to minimize.

Ten independent trials (from internal model pretraining to target fine-tuning) are conducted for each simulation experiment to provide statistically meaningful results given the variance inherent in model-free learning. Statistics are provided in terms of mean and ± 1 standard deviation of policy updates requires to solve the task family. Samples are provided per 1000 ("k-Samples") using a batch size of 512.

The supplementary materials describe further experiment details, such as the setup for the real block stacking target.

A. Block Stacking

The network architectures for the block stacking policies are specified in Table II. Although our policies are nonlinear, note this particular task is linear between τ and θ.

Task representation. In the block stacking task, the context vector $c = [c_{B_1}^{\mathrm{T}}, \ldots, c_{B_{N_B}}^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{7N_B}$ consists of the concatenation of N_B individual block contexts. The context vector for block b is $c_{B_b} = [x_b^w, z_b^w, \psi_b, h_b, \boldsymbol{C}_b^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^7$ In the above equations, x_b^w and z_b^w are the world x- and z-positions of the blocks. Each block orientation is defined by its rotation angle ψ_b about the block's vertical axis (y). The y-dimension, or height, of each block is h_i. Lastly, the block color $\boldsymbol{C}_b = [R_b, G_b, B_b]^{\mathrm{T}}$ is specified via red-green-blue tuple. Note that y_b^w is not part of the context, as the initial scene always consists of blocks on the workspace plane.

The control policy $\pi(a|s, \theta_b)$ for block stacking is a sequential straight-line skill parameterized by $\theta_b = [\theta_{\Delta x}, \theta_{\Delta y}, \theta_{\Delta z}]^{\mathrm{T}} \in \mathbb{R}^3$. This skill specifies waypoints that the robot traverses via impedance control by lifting the source block vertically, moving horizontally, and descending to the desired location. The skill preconditions are that the block is grasped and there are no obstructions to moving the object. The reward function is determined from the source block's position and the goal position upon the target block.

Using the internal model, CREST correctly obtained the relevant context variables as $\tau = [x_0^w, x_1^w, h_1, z_0^w, z_1^w]^T$, $\tau_{\Delta x} = [x_0^w, x_1^w]^T$, $\tau_{\Delta y} = [h_1]$, and $\tau_{\Delta z} = [z_0^w, z_1^w]^T$.

Nominal transfer for increasing context size. We conduct transfer experiments for $N_B = \{2, 6, 10, 14, 18\}$, with each N_B conducted independently. Our approach scales with the relevant part of the context space (Fig. 3), bounding the sample requirements for the target (as well as the cheaper, internal model). The increasing number of irrelevant dimensions from more blocks are eliminated by CREST prior to conducting domain randomization during pretraining.

For the case of $N_B = 10$, we trained directly in the target domain without transfer and observed similar results as for pretraining, suggesting the internal model accords well with the target domain. This explains why our approaches exhibit good zero-shot behavior over increasing context dimensions, unlike the baseline whose initial performance degrades as the number of irrelevant contexts increase.

Distribution shift in irrelevant contexts. We now evaluate the robustness of the learned block stacking policy to distributions shifts in irrelevant context variables. We conduct
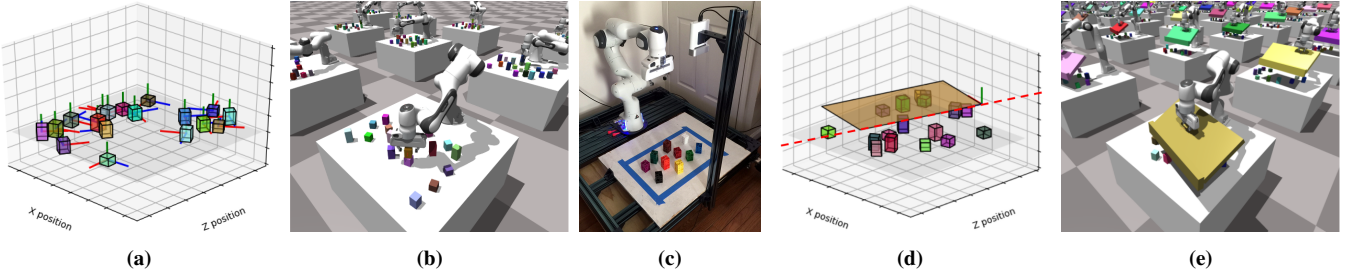
Fig. 2: Transfer experiments for block stacking and crate opening manipulation tasks. Policies are pretrained in the internal model ((a),(d)) and then transferred to the target domain ((b),(c),(e)). Target domains consist of replications of real systems using a Franka Panda robot, along with a real system for block stacking. Both tasks have distractor objects. For block stacking, only two blocks are necessary to generalize the policy. For crate opening, blocks represent distractor objects (e.g., if the crate were for a chest containing toys).
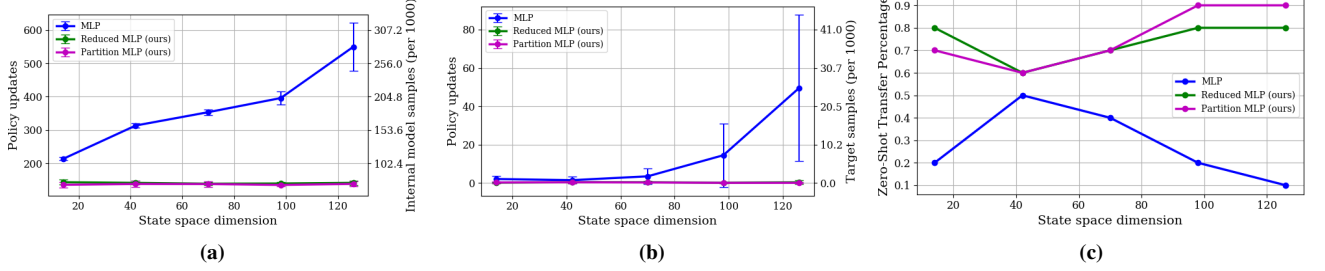
Fig. 3: Sample complexity of training a solved block stacking policy based on context dimension for (a) internal model and (b) target setting. c) Zero-shot transfer percentage, wherein the transferred policy needs no further target training to solve the task.

TABLE II: Networks used for the block stacking task.

Network	Parameters	Input Dim. (Total)	Architecture
MLP ($N_B = 2$)	3298	14 (14)	[24, 24, 24]
MLP ($N_B = 6$)	4642	42 (42)	[24, 24, 24]
MLP ($N_B = 10$)	5986	70 (70)	[24, 24, 24]
MLP ($N_B = 14$)	7330	98 (98)	[24, 24, 24]
MLP ($N_B = 18$)	8674	126 (126)	[24, 24, 24]
RMLP (ours)	2866	5 ($7N_B$)	[24, 24, 24]
PMLP (ours)	754	5 ($7N_B$)	[8, 8, 8] x 3

two transfer experiments, wherein the policies are pretrained using only half of the color space. In the first case, the target has the same context distribution as the internal model. In the second case, the target has the opposite color space (without overlap). The experimental results (Table III) elucidate how a seemingly inconsequential variable can degrade policy execution through a distribution shift that the robot is not trained to expect. Our approaches generate policies that are robust to these irrelevant domain shifts by construction; as CREST explicitly identifies this dimension as unimportant and excludes it from target learning.

Sim-to-real policy evaluation. Lastly, to validate our approach for sim-to-real transfer, we evaluate the zero-shot policy performance on a real robot system that implements the block stacking task with $N_B = 10$ (Fig. 2c). As shown in Table IV, our policies successfully demonstrate greater zero-shot, sim-to-real transfer as compared to the baseline.

B. Crate Opening

The crate opening experiment is nonlinear between τ and θ, and the internal model, which is kinematic, presents a greater sim-to-real gap than block stacking. Therefore, we also consider a second partitioned network, PMLP-R, with the same number of weights as the RMLP, to elicit possible influence of network expressivity in this domain due to the

structural assumptions of the PMLP. The crate experiment primarily focuses on dynamics and context shifts, rather than varying numbers of objects, so unlike in blocks, the networks (Table V) are the same for all experiments. Beyond our two experiments in nominal transfer and dynamics shift, we also conducted a color shift experiment with similar results as with the blocks experiment.

Task representation. For the crate opening task, the context vector is $c = [c_C^\mathrm{T}, c_{B_1}^\mathrm{T}, \ldots, c_{B_{10}}^\mathrm{T}]^\mathrm{T} \in \mathbb{R}^{80}$. The block context is as defined previously. The crate context is $c_C = [\boldsymbol{p}_C^{w\mathrm{T}}, \Phi, x_g^C, z_g^C, \Theta_o, \boldsymbol{C}_C^\mathrm{T}]^\mathrm{T} \in \mathbb{R}^{10}$, where $\boldsymbol{p}_C^w = [x_C^w, y_C^w, z_C^w]^\mathrm{T}$ is the position of the crate coordinate frame with respect to the world frame with vertical angle Φ. The crate is always initially closed (horizontal), but the desired goal angle is specified by Θ_o. The robot interacts with the crate via a grasp point specified in the frame of the crate by x_g^C and z_g^C which are orthogonal and parallel to the crate rotational axis, respectively. The color of the crate is $\boldsymbol{C}_C$.

The control policy $\pi(a|s, \theta_a)$ is a robot skill that executes circular arcs emerging from the grasp point with the following parameterization: $\theta_a = [\boldsymbol{\theta}_{\boldsymbol{p}_a^w}^\mathrm{T}, \theta_{\Delta\gamma}, \theta_{\Delta\phi}]^\mathrm{T} \in \mathbb{R}^5$, where $\boldsymbol{\theta}_{\boldsymbol{p}_a^w} = [\theta_{x_a^w}, \theta_{y_a^w}, \theta_{z_a^w}]^\mathrm{T}$ is the sphere position used to calculate the radius from the grasp point. Then, the arc is traced out $\theta_{\Delta\gamma}$ in azimuth and $\theta_{\Delta\phi}$ in inclination in polar coordinates from the grasp point. The skill preconditions are that the crate is grasped and unobstructed. To learn this policy, the reward function is calculated from the crate angle error and total kinematic error.

In this formulation, CREST determined the following relevant context variables, which are expected based on rigid

TABLE III: Transfer results for a distribution shift in 30 context variables (color) that are irrelevant for the block stacking policy.

Network	IM Updates (k-Samples)	Target Updates (k-Samples), no shift	Zero-Shot Transfer no shift	Target Updates (k-Samples), shift	Zero-Shot Transfer, shift
MLP	237.5 (121.6) $\pm$ 11.6 (5.9)	1.6 (0.8) $\pm$ 1.5 (0.8)	4	17.3 (8.9) $\pm$ 4.8 (2.5)	0
RMLP (ours)	137.6 (70.5) $\pm$ 7.2 (3.7)	0.5 (0.3) $\pm$ 0.9 (0.5)	7	1.0 (0.5) $\pm$ 1.5 (0.8)	6
PMLP (ours)	139.2 (71.3) $\pm$ 8.2 (4.2)	0.0 (0.0) $\pm$ 0.0 (0.0)	10	0.1 (0.1) $\pm$ 0.3 (0.2)	9

TABLE IV: Sim-to-real policy evaluation results for block stacking with $N_B = 10$. The reward threshold for zero-shot transfer is -0.025 (about half of the block width). We also note how often the block was successfully stacked. "GT" is a ground truth policy to illustrate the degree of uncertainty present within the robot perception and control system. Each policy was evaluated 10 times.

Policy	Reward	Zero-Shot Transfer	Block Stacked
MLP	-0.033 $\pm$ 0.012	3	1
RMLP (ours)	-0.018 $\pm$ 0.007	9	4
PMLP (ours)	-0.014 $\pm$ 0.004	10	6
GT	-0.009 $\pm$ 0.003	10	10

TABLE V: Networks used for the crate opening task.

Network	Parameters	Input Dim. (Total)	Architecture
MLP	13496	80 (80)	[40, 40, 40]
RMLP (ours)	7576	6 (80)	[40, 40, 40]
PMLP (ours)	1152	6 (80)	[8, 8, 8] x 5
PMLP-R (ours)	8032	6 (80)	[24, 24, 24] x 5

body articulation kinematics:

$$\tau = [\, x_C^w, \, y_C^w, \, z_C^w, \, \Phi, \, z_g^C, \, \Theta_g\,]^T$$
$$\tau_{x_a^w} = [x_C^w, \, \Phi, \, z_g^C]^T, \tau_{y_a^w} = [y_C^w]^T, \tau_{z_a^w} = [z_C^w, \, \Phi, \, z_g^C,]^T,$$
$$\tau_{\Delta\gamma} = [\Phi, \, \Theta_g]^T, \tau_{\Delta\phi} = [\Phi, \, \Theta_g]^T$$

Nominal transfer. The transfer learning results for the crate opening policy is shown in Table VI. Pretraining the model reduced the number of target updates for the non-partitioned networks. However, this was not the case for the partitioned networks, regardless of size. This is likely a result of the discrepancy between the internal model and the target domain, which also explains the difference between the policy updates required for pretraining versus training directly in the target. However, the reduction of relevant variables reduces the number of updates required to train directly in the target for both the RMLP and PMLP.

Dynamics distribution shift. Unlike the block stacking problem, the modeling gap between the internal model and target setting is sufficiently large that the trained policies incur a significant performance degradation upon first evaluating in the target domain. We investigated this further by transferring the policies to two target settings with different crate stiffness values. To focus on this dynamics shift, no other shifts (e.g., in context space) were induced.

The results in Table VII suggest that increasing the stiffness is sufficient as a proxy for increasing the modeling difference between the internal model and the target. The optimal parameters for the kinematic case (internal model) are not necessarily the same as the target domain with realistic dynamics of manipulation using impedance control. Therefore, greater modeling differences implies that greater search in policy parameter space is required to converge to parameters that generalize in the target domain.

In all cases, we see that the RMLP network performs best. As a likely consequence of a less expressive network with

TABLE VI: Pretraining and transfer results for crate opening policies compared to training directly in target (without transfer).

Network	IM Updates (k-Samples)	Target Updates (k-Samples), transfer	Target Updates (k-Samples), direct
MLP	45.0 $\pm$ 3.16 (23.04 $\pm$ 1.62)	12.20 $\pm$ 1.72 (6.25 $\pm$ 0.88)	38.70 $\pm$ 10.99 (19.8 $\pm$ 5.63)
RMLP (ours)	32.40 $\pm$ 3.67 (16.59 $\pm$ 1.88)	7.0 $\pm$ 1.18 (3.58 $\pm$ 0.61)	16.40 $\pm$ 4.05 (8.40 $\pm$ 2.08)
PMLP (ours)	48.20 $\pm$ 13.33 (24.68 $\pm$ 6.83)	14.0 $\pm$ 3.74 (7.17 $\pm$ 1.92)	14.20 $\pm$ 6.32 (7.27 $\pm$ 3.24)
PMLP-R (ours)	51.30 $\pm$ 10.82 (26.27 $\pm$ 5.54)	14.3 $\pm$ 4.34 (7.32 $\pm$ 2.22)	15.80 $\pm$ 3.97 (8.09 $\pm$ 2.03)

TABLE VII: Fine-tuning for crate opening policies with increasing crate stiffness and correspondingly greater transition model difference between the internal model and target task.

Network	Target Updates (k-Samples), light	Target Updates (k-Samples), nominal	Target Updates (k-Samples), stiff
MLP	3.90 $\pm$ 0.54 (2.00 $\pm$ 0.28)	12.20 $\pm$ 1.72 (6.25 $\pm$ 0.88)	27.30 $\pm$ 5.51 (13.98 $\pm$ 2.82)
RMLP (ours)	3.20 $\pm$ 0.60 (1.64 $\pm$ 0.31)	7.00 $\pm$ 1.18 (3.58 $\pm$ 0.61)	16.40 $\pm$ 5.90 (8.40 $\pm$ 3.02)
PMLP (ours)	9.10 $\pm$ 1.58 (4.66 $\pm$ 0.81)	14.00 $\pm$ 3.74 (7.17 $\pm$ 1.92)	24.00 $\pm$ 15.06 (12.29 $\pm$ 7.71)
PMLP-R (ours)	9.20 $\pm$ 1.54 (4.71 $\pm$ 0.79)	14.30 $\pm$ 4.34 (7.32 $\pm$ 2.22)	19.10 $\pm$ 4.91 (9.80 $\pm$ 2.51)

a larger dynamics gap, the smaller PMLP network demonstrated a significant variance increase in the higher stiffness case than the larger PMLP-R. Overall, our policies are more robust to distribution shifts in model dynamics. However, we note that partitioning imposes structure that may not be optimal for this problem, as the MLP outperformed the partitioned networks in the light and nominal stiffness cases.

VII. CONCLUSION

The causal reasoning afforded by CREST allows the robot to structure robot manipulation policies with fewer parameters that are more sample efficient and robust to domain shifts than a naive approach that includes all known contexts. Indeed, using causality to reason about the simulation of a task identifies *what* variables are important to generalize a policy, while domain randomized pretraining provides a strong, task-specific prior in terms of *how* they matter. We believe that CREST is one step towards a new paradigm for structural sim-to-real transfer of robot manipulation policies that are sufficiently lightweight to be adapted in-the-field to overcome unforeseen domain shifts.

For future work, we will investigate using precondition learning to relax the assumption that the policy execution is feasible. We will also explore how the robot can learn the internal model used as the causal reasoning engine.

REFERENCES

[1] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," *Int'l Conf. on Robotics and Automation (ICRA)*, 2018.

[2] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping," *Int'l Conf. on Robotics and Automation (ICRA)*, 2018.

[3] O. Kroemer, S. Niekum, and G. Konidaris, "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms," *arXiv preprint arXiv:1907.03146*, 2019.

[4] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey," *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020.

[5] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," *Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[6] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning," *Conf. on Robot Learning (CoRL)*, 2018.

[7] K. Zhang, B. Schölkopf, P. Spirtes, and C. Glymour, "Learning Causality and Causality-Related Learning: Some Recent Progress," *National Science Review*, vol. 5, no. 1, pp. 26–29, 2018.

[8] B. Schölkopf, "Causality for Machine Learning," *arXiv preprint arXiv:1911.10500*, 2019.

[9] J. Zhang and E. Bareinboim, "Transfer Learning in Multi-Armed Bandits: A Causal Approach," *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, 2017.

[10] P. de Haan, D. Jayaraman, and S. Levine, "Causal Confusion in Imitation Learning," *Conf. on Neural Information Processing Systems (NeurIPS)*, 2019.

[11] Y. Li, A. Torralba, A. Anandkumar, D. Fox, and A. Garg, "Causal Discovery in Physical Systems from Videos," *Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[12] S. A. Sontakke, A. Mehrjou, L. Itti, and B. Schölkopf, "Causal Curiosity: RL Agents Discovering Self-supervised Experiments for Causal Representation Learning," *arXiv preprint arXiv:2010.03110*, 2020.

[13] J. Pearl, *Causality*. Cambridge University Press, 2009.

[14] O. Ahmed, F. Träuble, A. Goyal, A. Neitz, M. Wüthrich, Y. Bengio, B. Schölkopf, and S. Bauer, "CausalWorld: A Robotic Manipulation Benchmark for Causal Structure and Transfer Learning," *arXiv preprint arXiv:2010.04296*, 2020.

[15] C. Devin, P. Abbeel, T. Darrell, and S. Levine, "Deep Object-Centric Representations for Generalizable Robot Learning," *Int'l Conf. on Robotics and Automation (ICRA)*, 2018.

[16] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the Unknown: Learning a Universal Policy with Online System Identification," *Robotics: Science and Systems (RSS)*, 2017.

[17] A. Nouri and M. L. Littman, "Dimension Reduction and its Application to Model-Based Exploration in Continuous Spaces," *Machine Learning*, vol. 81, no. 1, pp. 85–98, 2010.

[18] J. Z. Kolter and A. Y. Ng, "Regularization and Feature Selection in Least-Squares Temporal Difference Learning," *Int'l Conf. on Machine Learning*, 2009.

[19] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman, "An Analysis of Linear Models, Linear Value-Function Approximation, and Feature Selection for Reinforcement Learning," *Int'l Conf. on Machine Learning*, 2008.

[20] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-Real: Learning Agile Locomotion for Quadruped Robots," *Robotics: Science and Systems (RSS)*, 2018.

[21] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors," *Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[22] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience," *Int'l Conf. on Robotics and Automation (ICRA)*, 2019.

[23] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement Learning with Parameterized Actions," *AAAI Conf. on Artificial Intelligence*, 2016.

[24] M. Hausknecht and P. Stone, "Deep Reinforcement Learning in Parameterized Action Space," *Int'l Conf. on Learning Representations (ICLR)*, 2016.

[25] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space," *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, 2019.

[26] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[27] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum, "Simulation as an Engine of Physical Scene Understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18 327–18 332, 2013.

[28] D. Ha and J. Schmidhuber, "Recurrent World Models Facilitate Policy Evolution," *Conf. on Neural Information Processing Systems (NeurIPS)*, 2018.

[29] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio, "Toward Causal Representation Learning," *Proceedings of the IEEE*, 2021.

[30] J. Peters, K. Mülling, and Y. Altun, "Relative Entropy Policy Search," *AAAI Conf. on Artificial Intelligence*, 2010.

[31] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Algorithms," *Conf. on Neural Information Processing Systems (NeurIPS)*, 2000.

[32] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The Expressive Power of Neural Networks: A View from the Width," *Conf. on Neural Information Processing Systems (NeurIPS)*, 2017.

[33] A. M. Saxe, J. L. Mcclelland, and S. Ganguli, "Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks," *Int'l Conf. on Learning Representations (ICLR)*, 2014.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[35] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable Baselines," https://github.com/hill-a/stable-baselines, 2018.

[36] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive Neural Networks," *arXiv preprint arXiv:1606.04671*, 2016.