A SCALE and Appendices Overview

Fundamentally, SCALE is a causal learning algorithm for discovering compact, diverse skills through interventions in simulation. Figure 3 provides an overview of the approach.

Scaling to higher-dimensional context spaces. The SCALE algorithm scales linearly with the dimensionality of the context space, i.e., $O(|\mathbf{C}|)$, due to the necessity of performing interventions on each context variable. In the experiments examined in this work, the dimensionality of the context space was 36 and 8 for the block stacking and peg insertion domains, respectively. For other applications where the context space is very large, heuristics can be incorporated to first downselect the context space into a smaller candidate space that can be provided to SCALE. Example heuristics could include a distance metric (objects closer to the goal may be more likely to be relevant than those further away) or using other approaches such as meta-level priors [54].

Structure of appendices. These appendices are structured as follows. Appendix B provides greater details into the formalization of the simulator and its role as a causal reasoning engine. Appendix D formalizes the SCALE algorithm using nomenclature introduced in App. C. Next, App. E provides a toy experiment that is designed to convey greater intuition and visualization of the mechanisms that underlie SCALE. Appendix F presents additional experimental details of the block stacking experiment presented in Sec. 6.1. Following this, App. G and H provides two additional experiments in the block stacking domain: a sim-to-real transfer experiment and a downstream task evaluation experiment, respectively. The remaining appendices concern the peg-in-hole insertion domain. Appendix I details additional experimental details first presented in Sec. 6.2, and App. J presents an additional experiment that shows the robustness of SCALE under a task domain shift.

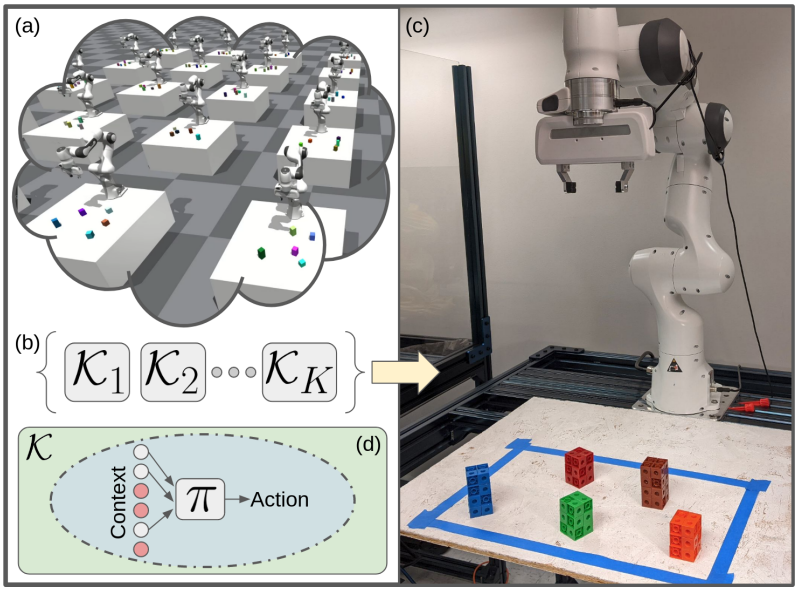


Figure 3: In SCALE, the robot learns skills in simulation using causal reasoning. (a) The simulation is used to solve task instances and conduct interventions to determine causally relevant context variables. (b) Simulation data are used to train a library of skills, (c) which are suitable for sim-to-real transfer learning. (d) Each skill that is learned is parameterized by the relevant variables selected in simulation. Here, red context variables are unnecessary for the skill policy and can be safely ignored. The boundary encircling the policy represents the skill DGR and precondition, which are also learned.

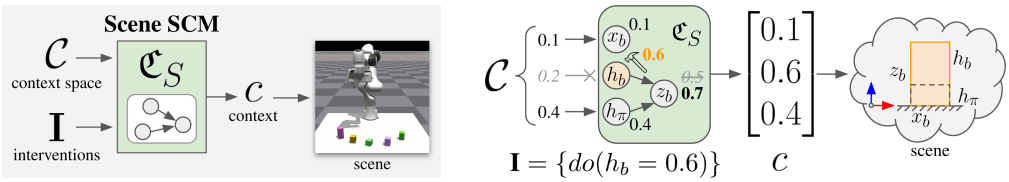


Figure 4: Illustrations of the scene structural causal model used in the simulator $\mathcal{W}$. (a) From context space $\mathcal{C}$ and robot interventions $\mathbf{I}$, the scene SCM $\mathfrak{C}_S$ generates a context vector c that represents a particular *scene* that defines objects and their properties. (b) In this block example, $\mathfrak{C}_S$ is defined using scene variables $\mathbf{\Psi} := \mathbf{C} \cup z_b$ and context variables $\mathbf{C} := \{x_b, h_b, h_\pi\}$, where x_b is block x-position, h_b is block height, h_π is table height upon which the block rests, and $z_b := \frac{1}{2}h_b + h_\pi$ is block z-position. Normally, values of $\mathbf{C}$ are sampled from context space $\mathcal{C}$, but the robot performs an intervention $\mathbf{I} = \{do(h_b = 0.6)\}$ to force the value of h_b to be 0.6. As a result, the dependent variable z_b is determined as 0.7 using this intervened value. Lastly, the scene is constructed and represented as context vector $c = [0.1, 0.6, 0.4]^\mathsf{T}$.

B Simulation as a Causal Reasoning Engine

This appendix provides greater discussion of the simulator formalization used by SCALE. The simulator model, $\mathcal{W} := (\mathfrak{C}_S, T)$, is formalized as follows:

1. a scene structural causal model $\mathfrak{C}_S$ (Fig. 4) that, given context space $\mathcal{C}$ and interventions $\mathbf{I}$, instantiates a scene that can be represented as a context vector, $c \in \mathcal{C}$;
2. the transition model T that captures the domain forward dynamics as the robot interacts with the world through θ starting from the scene initialized from $\mathfrak{C}_S$.

A structural causal model (SCM) [25, 26] can be represented as a directed acyclic graph that is driven by exogenous variables (functional inputs of the graph) that produces the solution for all variables within the graph. These two components of the simulator capture the spatial structure inherent to the scene itself ($\mathfrak{C}_S$), and the spatiotemporal structure of the robot interacting with the world (T). The simulator model $\mathcal{W}$, including the scene SCM and transition function, is provided for the robot to use. In principle, the scene SCM could be learned via causal representation learning [29], e.g., a world models approach that admits causal interventions.

The scene SCM $\mathfrak{C}_S$ is defined by structural equations with scene variables $\mathbf{\Psi}$, where $\mathbf{C} \subseteq \mathbf{\Psi}$. In the graph induced by $\mathfrak{C}_S$, the scene variables are the nodes, and context variables $\mathbf{C}$ are the root nodes and exogenous variables (functional inputs) of the SCM. The value of the context variables is given by interventions $\mathbf{I} = \{do(C_i = c_i)\}$ if specified, or otherwise sampled from the context space $\mathcal{C}$. The robot only conducts interventions with respect to $\mathbf{C}$ that would yield a steady-state solution and are physically realizable, excluding physically invalid scenes (e.g., object penetration).

The transition model T is the same as typical simulators. The forward dynamics are simulated through the initial state s_0, obtained from the scene created by $\mathfrak{C}_S$, and θ, the inputs to the low-level controller π_l. With these inputs, the system temporally evolves as usual until the end of the episode, where reward R_f is obtained and compared to a threshold R_S to determine if the task was solved.

C Nomenclature

Table 5 summarizes the nomenclature used in this paper and, in particular, the SCALE algorithm (c.f., App. D). Note the use of italics and bold type to disambiguate certain symbols. For example, $\mathbf{X}$ is a set of random variables, but $\boldsymbol{X}$ refers to a dataset matrix. The notation for a variable and its instantiation as a scalar may also be overloaded depending on the context.

Table 5: Table of nomenclature.

Symbol	Meaning
$\mathbf{X}$	set of d random variables, i.e., $\mathbf{X} \coloneqq \{X_1, \ldots, X_d\}$
$\mathcal{X}$	space of $\mathbf{X}$, i.e., $\mathcal{X} \coloneqq [\mathcal{X}_1, \ldots, \mathcal{X}_d]^{\mathrm{T}}$
x	vector instantiation of $\mathbf{X}$ i.e., $x \coloneqq [x_1 \in X_1 \subseteq \mathcal{X}_1, \ldots, x_d \in X_d \subseteq \mathcal{X}_d]^{\mathrm{T}}$
$\boldsymbol{\mathcal{K}}$	set of k robot skills, i.e., $\boldsymbol{\mathcal{K}} \coloneqq \{\mathcal{K}_1, \ldots, \mathcal{K}_k\}$
$\mathcal{D}$	dataset containing ${}^{\mathbf{A}}\boldsymbol{X} \in \mathbb{R}^{m \times n}$ samples from set $\mathbf{A}$ with size n and ${}^{\mathbf{B}}\boldsymbol{Y} \in \mathbb{R}^{m \times p}$ labels from set $\mathbf{B}$ with size p

D SCALE Algorithm

As explained in Sec. 5, the SCALE algorithm (Alg. 1) describes how the skills are learned through batch dataset collection and self-supervision. The procedure for batch dataset collection used by SCALE (SKILLTRAINDATA) is described in Alg. 2.

Note that the number of skills is not a hyperparameter of the SCALE algorithm. Rather, the skill quantity emerges from SPLITINTOSKILLDATASETS from groups of highly-occurring CREST results, where each group becomes the dataset for a particular skill.

Algorithm 1: SKILLS FROM CAUSAL LEARNING

Input: causal reasoning engine $\mathcal{W}$, context space $\mathcal{C}$, controller π_l, reward solved threshold R_S, number of samples n, skill policy function class f_π, number of evaluations m, skill timestep T_f

Initialize: skills $\boldsymbol{\mathcal{K}} \leftarrow \emptyset$

```
// Collect training data
(D_1, ..., D_k) ← SKILLTRAINDATA(W, C, π_l, n)
// Train skills
for j = 1 to k do
    (^C X, ^θ Y, D_A, D_B) ← D_j
    // Train DGR
    ^D X ← REDUCEDIMS(^C X, D_B)
    G ← TRAINDGR(^D X)
    // Train Policy
    ^A X ← REDUCEDIMS(^C X, D_A)
    (^A X+, ^θ Y+) ← DGRINLIERS(G, ^A X, ^D X, ^θ Y)
    π_u ← TRAINPOLICY(f_π, ^A X+, ^θ Y+)
    π ← π_l π_u
    // Train Preconditions
    (^C X_e, ^R Y_e) ← EVALUATEPOLICY(W, C, π, m)
    Pre ← TRAINPRECONDITION(^C X_e, ^R Y_e, R_S)
    // Set Termination Conditions
    β ← T_f
    // Construct Skill
    K ←+ (π, Pre, β, G)
end
```

Result: learned skills $\boldsymbol{\mathcal{K}}$

Algorithm 2: SKILLTRAINDATA

Input: causal reasoning engine $\mathcal{W}$, context space $\mathcal{C}$, controller π_l, reward solved threshold R_S, number of samples n, local region fraction f

Initialize: batch dataset $\mathcal{D}_B \leftarrow \emptyset$

```
// Collect training data
for i = 1 to n do
    c ← SAMPLEVALIDSCENE(W, C)
    (θ, R_f) ← TRYTOSOLVETASK(W, c, π_l)
    TaskSolved ← R_f > R_S
    if TaskSolved then
        A ← CREST(W, c, π_l, θ, R_f, fC)
        D ← CREST(W, c, π_l, θ, R_f, C)
        D_A ← DIMMATRIX(A)
        D_D ← DIMMATRIX(D)
        D_B ←+ (c, θ, D_A, D_D)
    end
end
// Separate into k skill datasets
(D_1, ..., D_k) ← SPLITINTOSKILLDATASETS(D_B)
```

Result: skill training data $(\mathcal{D}_1, \ldots, \mathcal{D}_k)$

E Block Stacking Intuitive Example

To provide greater intuition for SCALE and the causal skill learning problem, we present the *Height-Height* experiment (Fig. 5): a simple example in the block stacking domain that can be easily visualized.

Task and policy description. The *Height-Height* experiment contains 3 blocks: 1) a source block; 2) a target block; and 3) an obstructing block between the source and target block. As in Sec. 6.1, the task is to place the source block on top of the target block. The same controller is used as in Sec. 6.1, which is parameterized by $\theta \in \mathbb{R}^4$. Specifically, each parameter of the controller is defined as follows:

1. $\theta_{\Delta x}$: the distance the source block is moved along the world coordinate frame's $+x$-axis once it is picked up.
2. $\theta_{\Delta y}$: the distance the source block is moved along the world coordinate frame's $+y$-axis once it is picked up.
3. $\theta_{\Delta z_u}$: the distance the source block is lifted (moved along the world coordinate frame's $+z$-axis) during the pick-up motion.
4. $\theta_{\Delta z_d}$: the distance the source block descends (moved along the world coordinate frame's $-z$-axis) during the set-down motion.

The controller behaves as follows:

1. Move robot end-effector to source block and grasp it.
2. Lift up the source block according to $\theta_{\Delta z_u}$.
3. Move the source block in the x-y plane according policy parameters $\theta_{\Delta x}$ and $\theta_{\Delta y}$.
4. Set down the source block according to $\theta_{\Delta z_d}$.
5. Ungrasp the source block.

The context space of this experiment is just 2 variables, h_t and h_o, facilitating 2-dimensional visualizations. For greater clarity, we refer to block properties by whether they belong to the target block

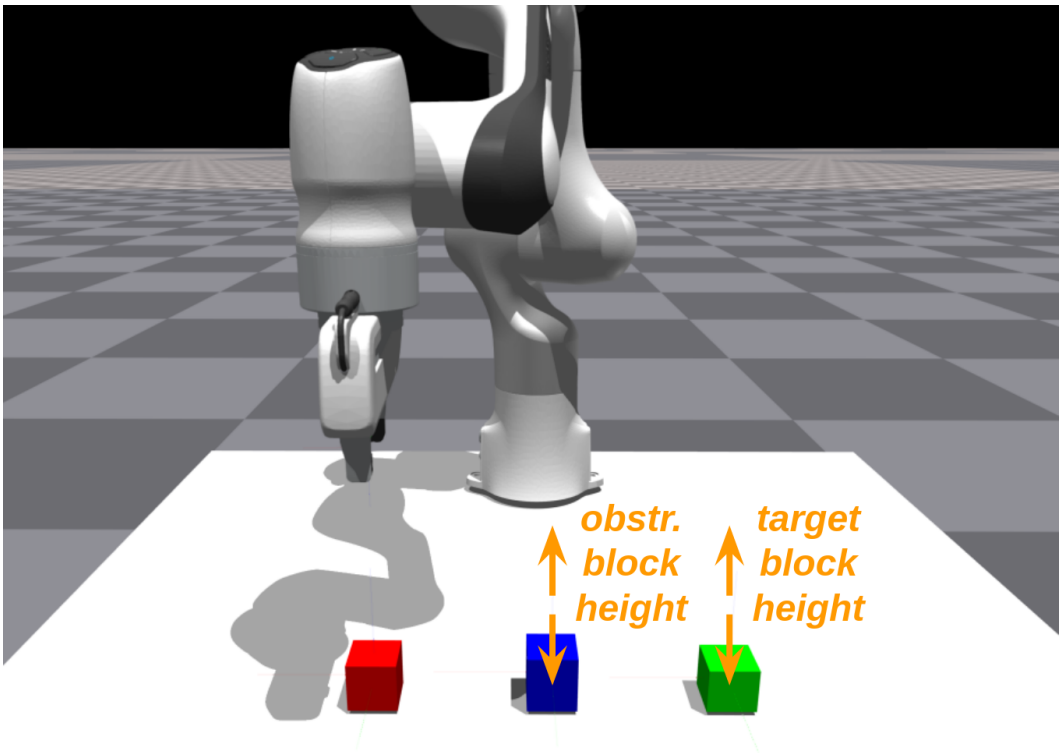


Figure 5: The *Height-Height* experiment is an intuitive example for SCALE in the block stacking domain. In this experiment, only two context variables can vary: the height (z-dimension) of the obstructing and target blocks. All others variables (e.g., features of the source block) do not change throughout this experiment.

(t) or the obstructing block (o), instead of their index (as in Sec. 6.1). For this experiment, only linear approaches are considered.

Skill learning results. The SCALE results for the *Height-Height* experiment are shown in Tab. 6 and Fig. 6. The dataset size for skill learning was 569 samples, from an original size of 581. The remaining 12 samples consisted of CREST results that occurred rarely (2.07%), and thus they were not used for skill learning. Additionally, Fig. 7 visualizes the policy parameters of the dataset. Two primary behaviors were learned: *free motion* ($\mathcal{K}_{free}$), and *obstructed motion* ($\mathcal{K}_{obstr}$). These behaviors emerge *because of* the causal relationships between context variables.

When the obstructing block is shorter than the target block (i.e., $h_t > h_o$), then the obstructing block height can safely be ignored in the robot action (thus, $h_o \notin \mathbf{A}$ for $\mathcal{K}_{free}$). This is reflected by the values of $\theta_{\Delta z_u}$ and $\theta_{\Delta z_d}$ in Fig. 7. In the region corresponding to $\mathcal{K}_{free}$, $\theta_{\Delta z_u}$ varies linearly with respect to the target block height, but not with the obstructing block height. Thus, $\theta_{\Delta z_d}$ is generally 0. The result is that the robot tends to lift the block to a value that depends on the target block height, and no set-down motion ($\theta_{\Delta z_d}$) is needed.

However, when the obstructing block is taller than the target block (i.e., $h_t < h_o$), the obstructing block's geometry interferes with the robot's motion, and the robot must take this into account when taking action. Specifically, the robot must first lift the source block over the obstructing block. After it moves laterally, the robot must descend to set the source block down; dropping the block would typically lead to inadequate reward to solve the task. Because both the heights of these blocks are needed to perform this action, $\{h_t, h_o\} \subseteq \mathbf{A}$ for $\mathcal{K}_{obstr}$. In Fig. 7, the effect of h_o appears in the $\theta_{\Delta z_u}$ parameter values, where the variation in the $\mathcal{K}_{obstr}$ region arises because of needing to lift above the obstructing block height, h_o (and thus, this parameter no longer depends on h_t). However,

for $\theta_{\Delta z_d}$, *both* h_t and h_o are needed, as the distances the robot descends through $\theta_{\Delta z_d}$ arises from the difference between h_t and h_o. Thus, the gradient here shows components for both h_t and h_o.

These two skills encode the two distinct data generating processes within this context space. These processes — the reason why the data are generated a certain way — fundamentally depend on whether the obstructing block is shorter or taller than the target block. Whether a condition holds for a given context requires the value of both of the blocks heights, so both block heights are needed to define each skill's data generating region (i.e., $\{h_t, h_o\} \subseteq \mathbf{D}$).

Note that neither skill can robustly solve the entire task space (55.63% for $\mathcal{K}_{free}$ and 57.50% for $\mathcal{K}_{obstr}$). However, when using the entire library $\boldsymbol{\mathcal{K}}_{HH} = \{\mathcal{K}_{free}, \mathcal{K}_{obstr}\}$ (Tab. 7), the success rate becomes 100.00%, with each skill being selected at approximately 50% chance (49.38% for $\mathcal{K}_{free}$, and 50.62% for $\mathcal{K}_{obstr}$). This is expected because the relationship $h_t > h_o$ holds for half of the context space and $\mathcal{K}_{free}$ should be used, whereas $h_t < h_o$ ($\mathcal{K}_{obstr}$) holds for the other half.

Table 6: Skills $\boldsymbol{\mathcal{K}}_{HH}$ that were discovered for the *Height-Height* experiment. $\mathbf{A}$ and $\mathbf{D}$ are the variables used for the skill's policy and DGR, respectively. Data is the quantity of data used for each skill (from a batch dataset of 581 samples, 569 samples were used to train skills). These samples are used to train a linear policy (Bayesian Ridge regression) using the features from variables in $\mathbf{A}$. Task Solve % is the rate of task solves over the entire context space using only that skill.

Skill	$\mathbf{A}$	$\mathbf{D}$	Data	Task Solve %
$\mathcal{K}_{free}$	$\{h_t\}$	$\{h_t, h_o\}$	253 (43.55%)	55.63% (178)
$\mathcal{K}_{obstr}$	$\{h_t, h_o\}$	$\{h_t, h_o\}$	316 (54.39%)	57.50% (184)

Baseline comparisons. In addition to scale-lin, Tab. 7 shows comparisons against several baselines. The "monopolicy" baselines are monolithic policies; they contain neither a DGR nor a precondition. The "-sk" and "-all" suffixes denote whether the monolithic policy uses the same data as the SCALE library ("-sk", 569 samples) or the entire batch dataset ("-all", 581 samples). Given the similar amount of data, it is unsurprising that monopolicy-lin-sk and monopolicy-lin-all are essentially the same up to the stochasticity of the simulator ($\pm 2\%$). Note that, unlike in Sec. 6.1 and Sec. 6.2, CREST monopolicy baselines are not examined in this experiment; they are functionally equivalent to the monopolicy approaches because the most common CREST result is $\{h_t, h_o\}$, which is the same as the entire context space used for the monopolicy baselines.

As shown in Tab. 7, the skill library obtained by SCALE vastly outperforms the baselines, providing task evaluation performance similar to that of a ground truth policy. This outcome is possible because SCALE learns underlying regions of similar causal structure within the data, whereas monolithic policies ignore such structure. As shown in Fig. 7c–7d, this domain is nonlinear, but can be represented by two smaller linear regions ($h_t > h_o$ and $h_t < h_o$). Learning to regress to both regions with a monolithic linear policy is not possible, but SCALE can solve this domain with separate linear skills, one per region.

Table 7: Task evaluation results for using the skill library $\boldsymbol{\mathcal{K}}_{HH}$ for the block stacking task. Ctrl. is the approach control (skills or one monolithic policy). Fn. Cl. is the approach's function class. Linear approaches use Bayesian ridge regression. Task Solve % is the rate of task solves over the entire context space using the approach. Methods within $\pm 2\%$ (the stochasticity of the simulator) of the best approach are bold. $|\mathbf{A}|$ is the quantity of input variables used for the approach's policy. Data is the amount of training data used for the approach. A ground truth policy is also shown, using all context variables and additional domain knowledge.

Approach	Ctrl.	Fn. Cl.	Task Solve %	$\lvert\mathbf{A}\rvert$	Data
scale-lin (ours)	2 skills	Linear	**100.00%** (320)	1/1	569
monopolicy-lin-sk	1 policy	Linear	64.06% (205)	2	569
monopolicy-lin-all	1 policy	Linear	62.19% (199)	2	581
ground-truth-policy	1 policy	Nonlin.	100.00% (320)	*	–

Summary. Our approach for SCALE — learning skills that encode distinct causal processes — empowers the robot with a diversity of specialized behaviors to use, depending on the context.

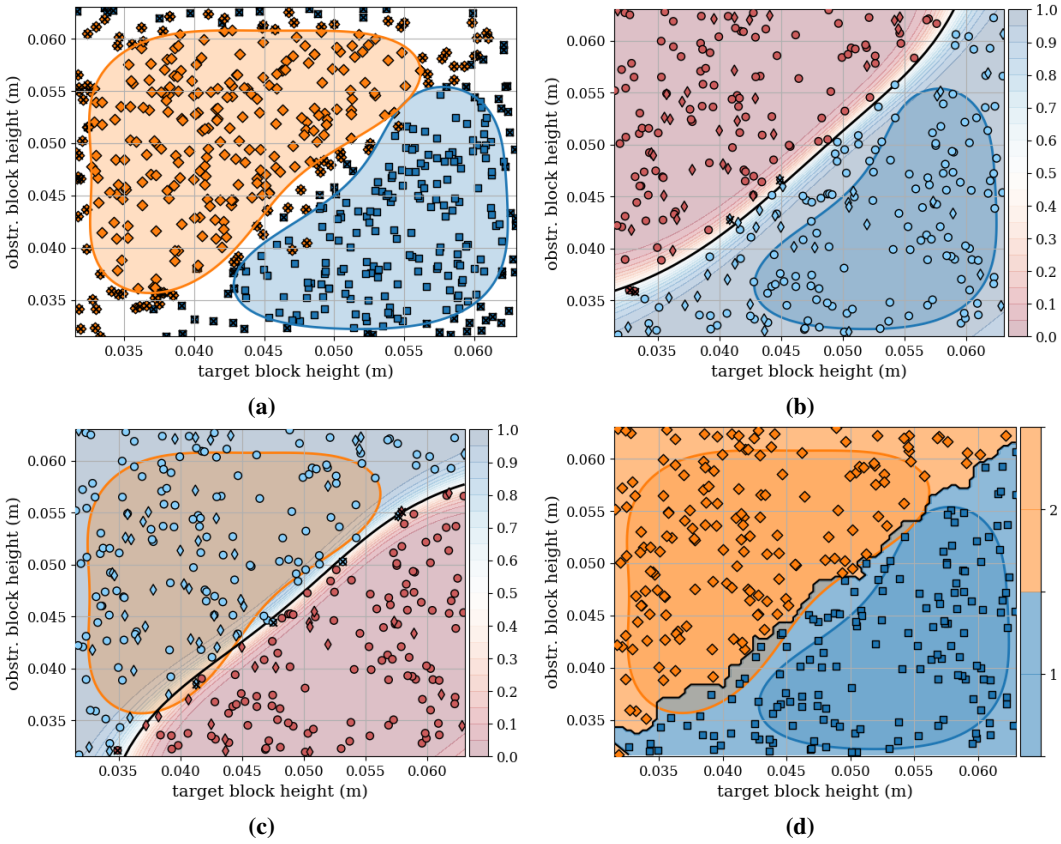


Figure 6: SCALE results for the *Height-Height* experiment. Two skills were found: $\mathcal{K}_{free}$ (free block motion), stylized in blue with rectangular markers, and $\mathcal{K}_{obstr}$ (obstructed block motion), stylized in orange with diamond markers. (a) Learned data generating regions. Each datapoint is a result from CREST. Datapoints that are crossed out are considered outliers and not used for training the policy for that skill. (b–c) Preconditions for $\mathcal{K}_{free}$ and $\mathcal{K}_{obstr}$, respectively. The black line is the decision boundary for the prediction of whether the task would or would not be solved with that skill. Note that each skill's DGR falls within the positive precondition boundary. Training and test data for learning the preconditions are indicated by circle and thin diamond markers, respectively. Datapoints that result in a different prediction than observed are crossed out. (d) Task evaluation when using the skill library $\{\mathcal{K}_{free}, \mathcal{K}_{obstr}\}$ to solve the task. The marker and color of each datapoint indicate which skill was selected for completing the task based on the skill preconditions (i.e., the skill with the highest probability of success). Note that the separation between selecting $\mathcal{K}_{free}$ and $\mathcal{K}_{obstr}$ is consistent with each skills' underlying precondition and DGR. Datapoints that were not solved by the chosen skill are crossed out.

Generalization of the context space can be achieved then through the composition of these behaviors, rather than attempting to learn a monolithic skill or policy that can capture the entire variation. In this example, two skills each with a linear policy is sufficient for generalization with SCALE, whereas a monolithic approach would require a nonlinear policy.

F Additional Details for Block Stacking Experiment

This appendix provides greater information for the block stacking experiment first presented in Sec. 6.1.

Context. Note that the block vertical position $z_b^w \in \mathbf{\Psi}$ is not part of the context, as we only consider cases where the scene can be initialized into a steady state condition. Thus, $z_b^w \coloneqq \frac{1}{2}h_b + h_\pi$.

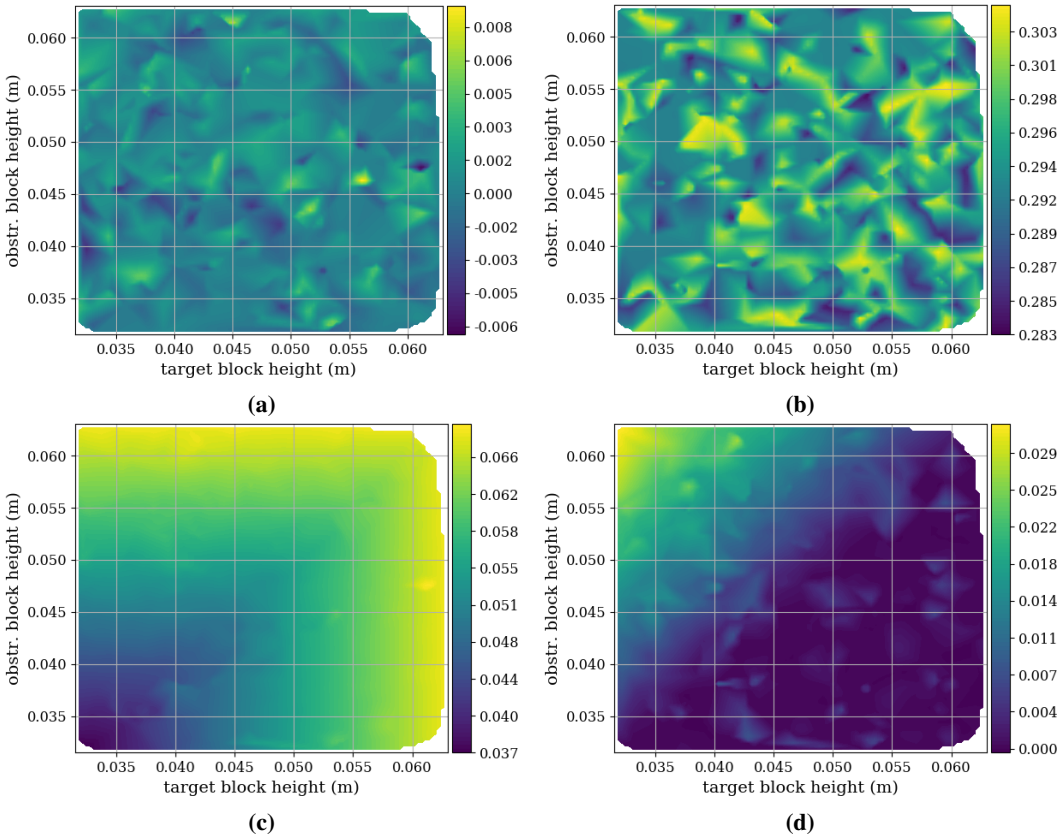


Figure 7: Policy parameters for the *Height-Height* experiment (shown as interpolated across the 569 dataset samples to better visualize the gradients). The units of the parameters are in meters. The parameters $\theta_{\Delta x}$ (a) and $\theta_{\Delta y}$ (b) are generally constant as they are unaffected by the variation in context variables. The notable variations occur in $\theta_{\Delta z_u}$ (c) and $\theta_{\Delta z_d}$ (d). Specifically, the relationship changes whether the obstructing block is taller or shorter than the target block (above or below the $h_t - h_o = 0$ line, respectively).

Reward function. The reward function for the task is $R = R_B - \alpha_L L - \alpha_e e - \alpha_d d$, where $R_B = 10$ is a bonus term obtained when the block is successfully stacked, L is the total end-effector path of the robot ($\alpha_L = 1$), e is the L2 norm error between the source block at the time of release and the goal ($\alpha_e = 1$), and d is the distance the source block travels between the point it was ungrasped to its final position ($\alpha_d = 1$). The task is considered solved if the final reward R_f exceeds solved threshold $R_S = 5$.

SCALE skill selection. In all SCALE approaches, the skills were complementary; using the entire skill library afforded greater coverage (greater task solve rate) than any single skill alone. For scale-lin, the skill selection distribution was almost even between $\mathcal{K}_1$ (43.28%) and $\mathcal{K}_2$ (56.72%), with $\mathcal{K}_3$ never being chosen. The skill $\mathcal{K}_3$ is dominated by the other two skills for this task, but $\mathcal{K}_3$ could nonetheless be useful for a different downstream task. Empirically, it was observed that $\mathcal{K}_1$ was chosen for shorter target block heights, whereas $\mathcal{K}_2$ was used elsewhere (see Fig. 8). In the nonlinear case, only $\mathcal{K}_2$ was selected.

G Sim-to-Real Block Stacking Experiment

In this appendix, we demonstrate that the skills learned by SCALE are suitable for sim-to-real transfer. As skills are constructed using only the relevant causal variables, this is a form of *structural* sim-to-real transfer. For this experiment, we evaluate the skill library $\boldsymbol{\mathcal{K}}_{blocks}$ for a real block stack-

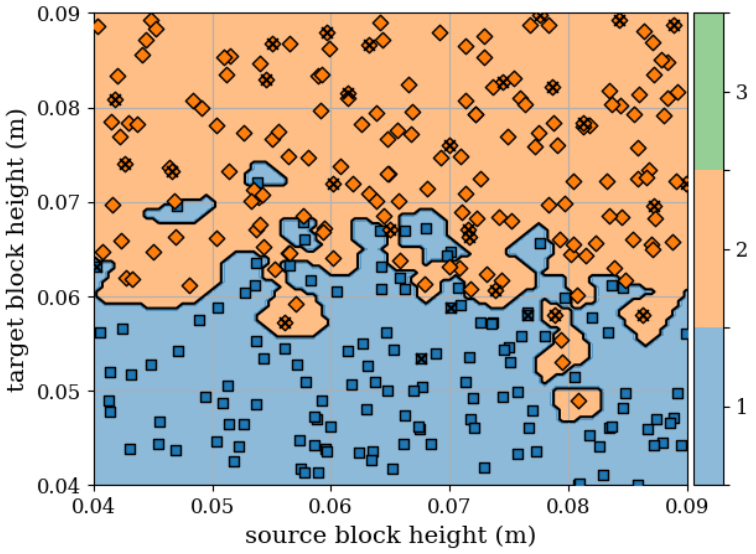


Figure 8: Skill selection for the scale-lin approach for the block stacking task. Skill $\mathcal{K}_1$ is generally selected when h_2 is short, whereas taller h_2 values perform better with $\mathcal{K}_2$ because $h_2 \subseteq \mathbf{A}$. Skill $\mathcal{K}_3$ is dominated by the other two skills and is not selected. Datapoints that were not solved are crossed out.

ing domain with a Franka Emika Panda robot manipulator (Fig. 2c). This experiment is generally similar to task evaluation in simulation, except with a smaller subset of the context space. We assess the SCALE approaches, scale-lin and scale-nonlin, against their monopolicy counterparts. Unlike in Sec. 6.1, we only consider the "-all" monopolicy approaches, as they were generally better performing.

G.1 Experimental Setup

For this experiment, a smaller subset of the context space is varied, as compared to the variation across the entire context space as tested in Tab. 2. From a pool of 20 blocks, 5 were randomly chosen to be used for each experimental trial. The 20 blocks consisted of variations of 10 different colors and 2 different heights (5.7 cm or 7.6 cm). The length and width of the blocks were 4.2 cm. The 5 randomly chosen blocks were placed into the Panda robot workspace and randomly shuffled, producing variation in block x-position, y-position, and orientation. The table height h_π was determined from manual measurement and was not varied for this experiment.

Perception. An Intel RealSense camera mounted to the robot wrist provided RGB-D perception of the x-position, y-position, and orientation of the blocks in the workspace. A depth observation was collected by commanding the robot above the workspace. This point cloud was then processed to yield five clusters via hidden point removal [55], RANSAC-based table plane fitting, and density-based clustering using DBSCAN [56]. Averaging the colors within each cluster yielded the block color. A least-squares optimization procedure fit a cuboid of known length and width to each cluster, yielding the position and orientation of the blocks. Block height was provided by manual input because of inaccuracies with estimation from depth alone. The camera extrinsics were obtained via computer-aided design models of the Panda robot and wrist mount, which were confirmed via manual measurement. The camera intrinsics were used as directly reported by the camera.

Control. The `FrankaPy` library [57] is used to provide impedance-based control of the Panda robot.

G.2 Experimental Results

Table 8 presents the results. For each function class, the skill library learned by SCALE outperforms the full-dimensional monopolicy baseline and is generally comparable to or slightly outcompetes the CREST monopolicy baseline. The ground truth policy matched the linear SCALE approach and is only slightly better than the nonlinear SCALE approach. Compared to the task solve rate in simulation (Tab. 2), scale-lin performed consistently, and scale-nonlin had slightly better performance. All baseline approaches generally matched their evaluation in simulation, except for monopolicy-lin-all, which had a marked degradation. This may arise from domain differences between simulation and reality. Full-dimensional approaches are more susceptible to domain shifts due to their reliance on the entire context space (all 36 variables), whereas SCALE approaches are compressed, using only a minimal subset. Error was only loosely correlated with task solve rate, and likely explains the poor performance of monopolicy-nonlin-all. Even though their errors were similar, it was observed that monopolicy-lin-all tended to underpredict the height needed to clear the target block as compared to scale-lin. This caused the target block to be pushed away from where it should have been for the goal position, leading to block stacking failures.

For both scale-lin and scale-nonlin, skill $\mathcal{K}_2$ was always chosen, as its precondition was on average greater than that of the other skills. Specifically, for scale-lin, the average preconditions were 58.88% for $\mathcal{K}_1$, 75.77% for $\mathcal{K}_2$, and 36.99% for $\mathcal{K}_3$. As the block heights used were only 5.7 cm and 7.6 cm, it is reasonable to expect that skill $\mathcal{K}_1$ would have been chosen more for shorter target block heights (per Fig. 8). For scale-nonlin, the average preconditions were $\mathcal{K}_1$: 20.17%, $\mathcal{K}_2$: 51.84%, $\mathcal{K}_3$: 1.21%.

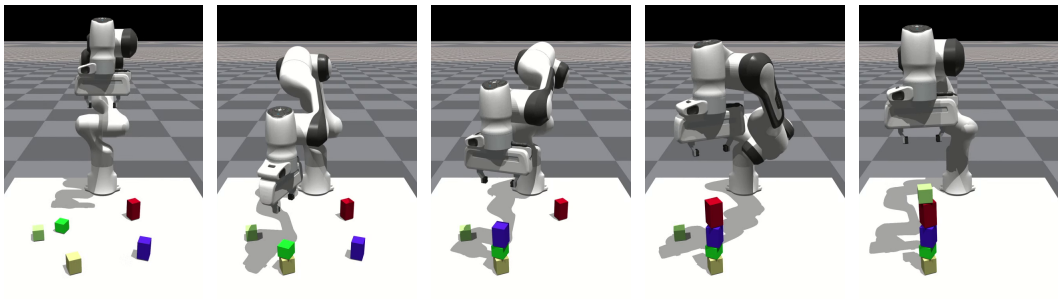
Table 8: Sim-to-real evaluation results for using the skill library $\boldsymbol{\mathcal{K}}_{blocks}$ for a real block stacking domain. Table columns are as described in Tab. 2. Task Solve % is the rate of successful block stacks. Error is the mean error (± 1 standard deviation) in meters between the block position when the block is ungrasped and the goal position determined at the beginning of the trial.

Approach	Ctrl.	Fn. Cl.	Task Solve %	Error	$\lvert\mathbf{A}\rvert$
scale-lin (ours)	3 skills	Linear	**90.00%** (9)	0.010 ± 0.003	4/5/6
monopolicy-lin-all	1 policy	Linear	50.00% (5)	0.008 ± 0.003	36
crest-monopolicy-lin-all	1 policy	Linear	**90.00%** (9)	0.004 ± 0.001	5
scale-nonlin (ours)	3 skills	Nonlinear	**80.00%** (8)	0.007 ± 0.002	4/5/6
monopolicy-nonlin-all	1 policy	Nonlinear	10.00% (1)	0.093 ± 0.040	36
crest-monopolicy-nonlin-all	1 policy	Nonlinear	70.00% (7)	0.013 ± 0.012	5
ground-truth-policy	1 policy	Nonlinear	90.00% (9)	0.002 ± 0.003	*

H Skill Library Use in a Downstream Task: Stacking a Block Tower

To demonstrate the utility of re-using skills learned by SCALE, a follow-up experiment is conducted wherein the skill library $\boldsymbol{\mathcal{K}}_{blocks}$ is used for a task in which it was not specifically trained: stacking a block tower (Fig. 9). This long-horizon task can be decomposed into a number of sequential actions that must be performed correctly, so an approach that can capture the essence of a large problem and re-use smaller, modular components should perform best. Moreover, we do not perform any additional training or fine-tuning; we intentionally use the skills off-training data to test their generalization capability. This is a challenging task: in addition to the long-horizon precision involved, the skills are being evaluated increasingly out-of-distribution at each step, as the effective block heights increase beyond what is seen in training.

Experimental setup. For this experiment, we assume that the robot has access to a planner and additional domain knowledge as a part of this downstream task. We assume that the robot understands that at any step, the target block should be adjusted in the following manner. First, the target block's x- and y-position should be substituted with the bottom-most block's x- and y-position. Then, the target block's height should be substituted with the sum of all heights of the previous blocks, plus a small offset (1.5 cm). Effectively, this can be seen as treating each new step as stacking upon one,



(a) **(b)** **(c)** **(d)** **(e)**

Figure 9: The block tower task. As previously, five blocks are initially available to the robot. However, after each stack attempt, the task does not reset. Instead, the block enumeration changes, so that the previous source block becomes the new target block. This happens four times, after which the task resets. The robot must complete each of the four individual steps successfully, as failure in any step renders the entire block tower task a failure. (a) Initial task scene. (b – d) Successful block stacks for intermediate attempts. (e) A successfully stacked block tower.

increasingly taller block. We leave the development of such a planner that can provide this additional information for future work, but it suffices for this experiment that this information is available.

Block tower results. Table 9 shows the results for stacking the block tower. For this experiment, we use the same linear and nonlinear approaches and baselines from Sec. 6.1. Included is a ground truth policy with access to oracle information.

Overall, we see that the scale-lin approach does best for stacking a tower with five blocks, although a notable gap exists between the ground truth policy. However, a block tower success rate of 48.29% is not unreasonable, given that even the ground truth policy fails almost 30% of the time. The linear approaches are all comparable for the first stacking step, and for the second step with a $N_B = 3$ tall tower, three baseline methods slightly outperform scale-lin. However, for the last two steps, baseline approaches become markedly less performant, leading to scale-lin emerging as the best overall approach despite modest performance in an absolute sense. Each step requires successively greater extrapolation out of the training data, so an approach that can capture the smaller process well should perform best, assuming that this process also holds outside the training data. For the case of the block tower, this is generally true, so the skills learned by scale-lin are best suited for this downstream task despite the challenge of generalization to yet-unseen data.

For the nonlinear function class, performance across all approaches suffers beyond the first stacking step, where the CREST baselines outperform scale-nonlin. The challenge of extrapolation for nonlinear functions is evident here; the best linear approach for each step was better performing than any nonlinear approach (and markedly so for taller towers). Thus, out-of-distribution generalization is not observed for any nonlinear approach, whereas scale-lin exhibits modest performance in this area.

For SCALE approaches, the skill selection rate is intriguing. The skill $\mathcal{K}_1$ does not contain the target block height, which is likely why it was only selected during the first block stack attempts. However, $\mathcal{K}_2$ continues to demonstrate its robustness, as it was used for all remaining block stack attempts in the linear case and for all attempts in the nonlinear case. Its inclusion of target block height in $\mathbf{A}_{\mathcal{K}_2}$ is in fact the reason this skill can extrapolate to taller towers. Like $\mathcal{K}_2$, $\mathcal{K}_3$ also contains the block height, but this skill was generally dominated, and thus it is not surprising it was not selected.

In summary, in addition to the benefits of SCALE described previously for task learning, the capability for SCALE to learn smaller, modular skills is evident in this experiment. Although out-of-distribution generalization was not observed in the nonlinear function class, we see that in principle SCALE does offer these benefits under certain conditions, such as in the linear case. We suggest that this aspect of causal learning is often overlooked for experiments that only concern single-

task learning. However, the benefits of modularity become advantageous for re-using behaviors for downstream tasks at a later time in the robot's operational lifetime.

Table 9: Results for re-using learned behaviors in a representative downstream task: stacking a block tower. The task solve percentage is shown for stacking a tower of at least N_B blocks tall. The sequence is executed in one attempt, so a fully stacked tower ($N_B = 5$) requires 4 successful block stacking attempts. Methods within $\pm 2\%$ (the stochasticity of the simulator) of the best approach at each step are bold. For SCALE approaches, the skill selection rate at each step (not cumulative) is also shown. The abbreviation "mp" stands for monopolicy.

Approach		$N_B = 2$	$N_B = 3$	$N_B = 4$	$N_B = 5$
scale-lin (ours)		**92.20%** (272)	80.73% (222)	**65.23%** (167)	**48.29%** (113)
	$\mathcal{K}_1$	15.59% (46)	0.00% (0)	0.00% (0)	0.00% (0)
	$\mathcal{K}_2$	84.07% (248)	100.00% (275)	100.00% (256)	100.00% (234)
	$\mathcal{K}_3$	0.34% (1)	0.00% (0)	0.00% (0)	0.00% (0)
monopolicy-lin-sk		**93.22%** (275)	**87.23%** (239)	55.08% (141)	1.27% (3)
monopolicy-lin-all		**93.56%** (276)	76.36% (210)	2.33% (6)	0.00% (0)
crest-mp-lin-subs		**93.20%** (274)	**85.40% (234)**	5.84% (15)	0.00% (0)
crest-mp-lin-all		**93.92%** (278)	**85.51% (236)**	5.84% (15)	0.00% (0)
scale-nonlin (ours)		67.46% (199)	2.55% (7)	0.00% (0)	0.00% (0)
	$\mathcal{K}_1$	0.00% (0)	0.00% (0)	0.00% (0)	0.00% (0)
	$\mathcal{K}_2$	100.00% (295)	100.00% (275)	100.00% (256)	100.00% (235)
	$\mathcal{K}_3$	0.00% (0)	0.00% (0)	0.00% (0)	0.00% (0)
monopolicy-nonlin-sk		2.72% (8)	0.00% (0)	0.00% (0)	0.00% (0)
monopolicy-nonlin-all		11.86% (35)	0.00% (0)	0.00% (0)	0.00% (0)
crest-mp-nonlin-subs		**84.75%** (250)	**27.37%** (75)	0.78% (2)	0.00% (0)
crest-mp-nonlin-all		75.59% (223)	11.31% (31)	0.00% (0)	0.00% (0)
ground-truth-policy		96.25% (282)	90.48% (247)	83.14% (212)	69.96% (163)

I Additional Details for Sensorless Peg-in-Hole Insertion Experiment

This appendix serves to provide greater detail for the peg insertion experiment that was described in Sec. 6.2.

Reward function. Our reward function consists of two terms: 1) a penalty based on the Euclidean distance of the peg from the hole, and 2) a bonus of 10 for successful insertion. We also add a regularization term based on the norm of the policy parameters. The task is considered solved if the final reward R_f exceeds solved threshold $R_S = 8$.

SCALE skill $\mathcal{K}_1$. Unlike the other skills in $\boldsymbol{\mathcal{K}}_{peg}$ that were discovered by SCALE, skill $\mathcal{K}_1$ has an empty set of relevant variables. This is surprising as it is difficult to solve this task reliably without taking the help of one of the walls, in which case the wall should show up as a relevant variable. However, we observed that $\mathcal{K}_1$ actually localizes against 2 walls instead of just 1. Hence, when SCALE intervenes on any one of the two walls, the skill is still able to complete the assembly by taking advantage of the other wall. In other words, our assumption that the context space is disentangled does not hold in this case which leads to this erroneous relevant variable set. However, the precondition would limit where this skill would be applied, as skills $\mathcal{K}_{2-5}$ are generally more performant.

SCALE skill selection. For scale-lin, skills $\mathcal{K}_2$ (48.44%) and $\mathcal{K}_5$ (51.56%) were chosen nearly equally. Conversely, the skill selection was more distributed for the nonlinear case: $\mathcal{K}_2$: 46.48%, $\mathcal{K}_3$: 35.16%, $\mathcal{K}_4$: 3.91%, $\mathcal{K}_5$: 14.45%. For both approaches, $\mathcal{K}_1$ was not chosen as it was dominated by the other skills.

J Sensorless Peg-in-Hole Insertion: Domain Shift Experiment

We evaluate the generalization capability of SCALE by evaluating it under a domain shift. All tasks are generated by uniformly sampling the relative position of the center of each wall with respect to

the hole from a given range. The ranges used to generate the training and test tasks are specified in Tab. 10.

We transfer all the policies zero-shot to the test distribution. However, we do re-learn the preconditions of the scale-lin policies for the test distribution. The evaluation results are summarized in Tab. 11. All approaches witness a sharp drop in performance. This is expected as (a) the test tasks are not guaranteed to be feasible and (b) the ranges used to generate the test task are more than double those used in training. However, our multi-skill approach scale-lin performs much better than the baselines. This highlights a key benefit of learning multiple skills. A skill may perform well on the training distribution but it can be rendered invalid due to an unforeseen domain shift. Having a repertoire of different skills allows the robot to still complete the task by switching to a different skill. This makes our multi-skill approach more robust than single-skill approaches.

Table 10: Training and test distributions: relative position of the center of each of the 4 walls is uniformly sampled from the given $(\min, \max)$ range. The ranges used to generate test tasks are more than double the ranges used to generate training tasks in the domain shift experiment. All values are in meters.

	Train				Test			
	x-min	x-max	y-min	y-max	x-min	x-max	y-min	y-max
Wall 1	0.01	0.05	-0.02	0.02	-0.04	0.10	-0.07	0.07
Wall 2	-0.02	0.02	-0.05	-0.01	-0.07	0.07	-0.10	0.07
Wall 3	-0.02	0.02	0.01	0.05	-0.07	0.07	-0.04	0.10
Wall 4	-0.05	-0.01	-0.02	0.02	-0.10	-0.04	-0.07	0.07

Table 11: Task evaluation results under domain shift for sensorless peg-in-hole insertion. We evaluate only linear policies as nonlinear policies perform worse in this domain (see Tab. 4).

Approach	Ctrl.	Fn. Cl.	Task Solve %	$\lvert\mathbf{A}\rvert$
scale-lin (ours)	5 skills	Linear	**64.84%**	0/1/1/1/1
monopolicy-lin-all	1 policy	Linear	44.92%	8
crest-monopolicy-lin-all	1 policy	Linear	39.83%	1