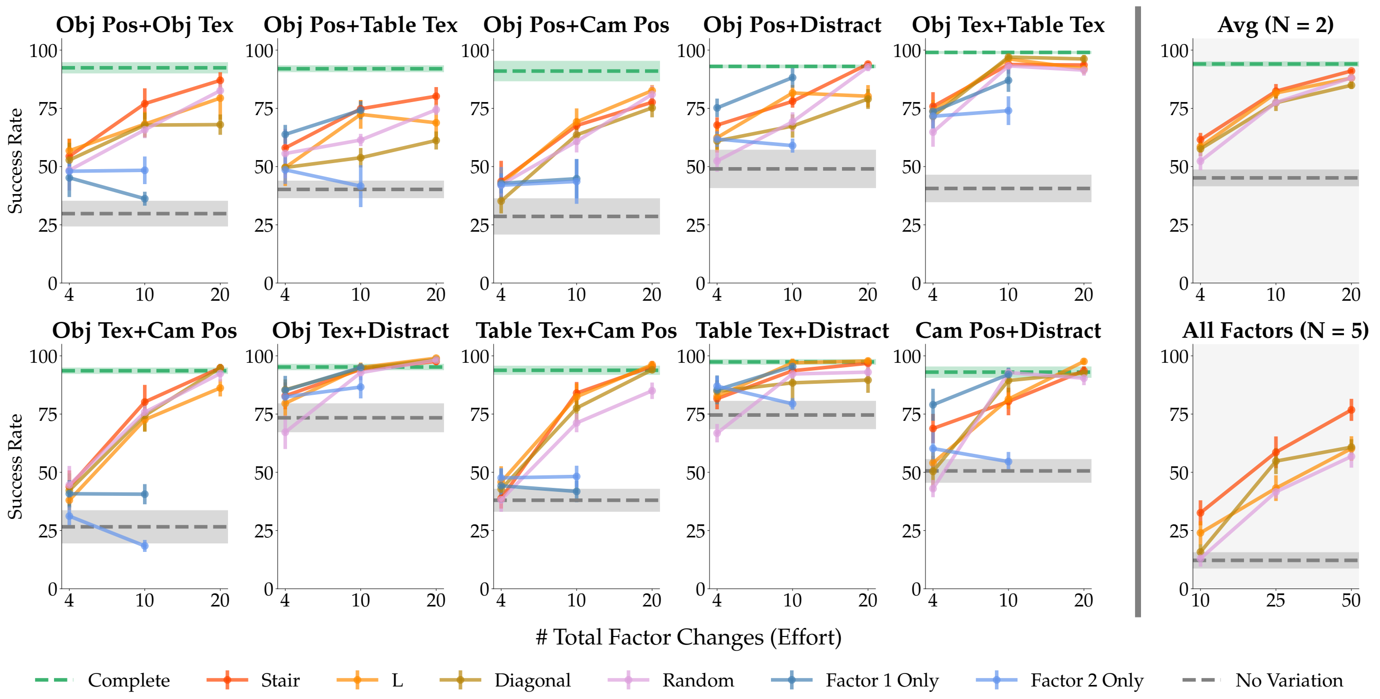


Fig. 11: Simulation results of different data collection strategies for *Door Open*. We report results where $\mathcal{F}^N$ consists of each possible factor pair ($N = 2$), average results across all pairs, and results where $\mathcal{F}^N$ consists of all factors ($N = 5$). Results are similar as with *Pick Place*, with **Stair** generally performing the best, especially in the ($N = 5$) setting. Error bars represent standard error across 5 seeds.

APPENDIX A
DATA COLLECTION STRATEGIES

A. *Pseudocode*

We provide pseudocode for the data collection strategies proposed in Section III-C that are intended to exploit compositional generalization (**Diagonal**, **L**, **Stair**).

Algorithm 1 Diagonal

```
Input: scene S, factor values F (size $N$ factors × $k$ values)
for $j \leftarrow 1$ to $k$ do
    $f \leftarrow 0^N$
    for $i \leftarrow 1$ to $N$ do
        $f_i \leftarrow \mathbf{F}_{ij}$
    end for
    SETFACTORS(S, $f$)
    COLLECTDATA(S)
end for
```

Algorithm 2 L

```
Input: scene S, factor values F (size $N$ factors × $k$ values),
base factor values $f^*$ (size $N$ factors)
for $i \leftarrow 1$ to $N$ do
    $f \leftarrow f^*$
    for $j \leftarrow 1$ to $k$ do
        $f_i \leftarrow \mathbf{F}_{ij}$
        SETFACTORS(S, $f$)
        COLLECTDATA(S)
    end for
end for
```

Algorithm 3 Stair

```
Input: scene S, factor values F (size $N$ factors × $k$ values),
base factor values $f^*$ (size $N$ factors)
$f \leftarrow f^*$
for $j \leftarrow 1$ to $k$ do
    for $i \leftarrow 1$ to $N$ do
        $f_i \leftarrow \mathbf{F}_{ij}$
        SETFACTORS(S, $f$)
        COLLECTDATA(S)
    end for
end for
```

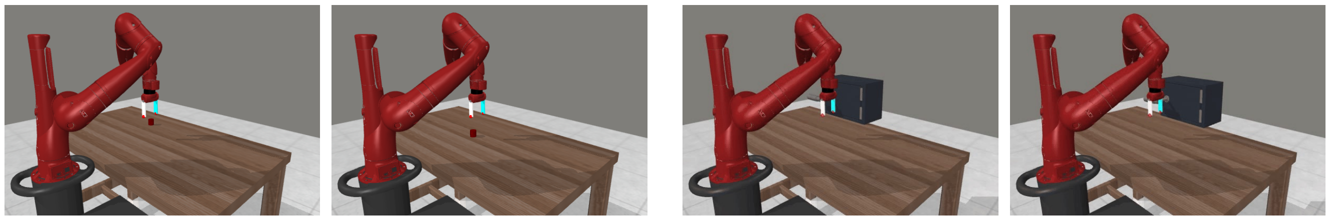
APPENDIX B
SIMULATION EXPERIMENTS

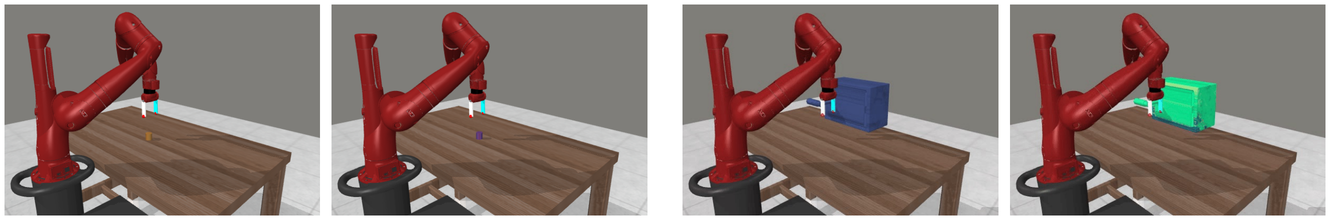
A. *Door Open Results*

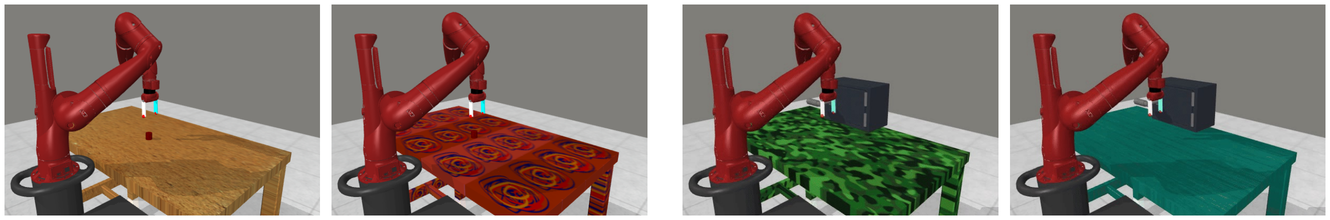
We include additional simulation results for the task *Door Open* in Fig. 11. Results are similar as in *Pick Place*, with generally strong pairwise composition, and **Stair** generally performing the best, especially in the $N = 5$ setting.

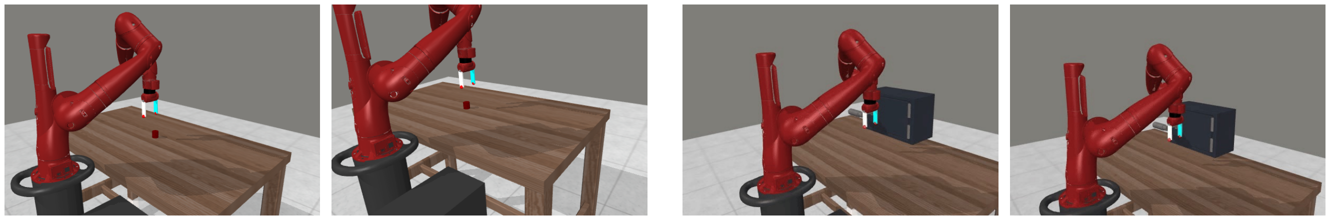
B. *Factors*

For the inherently discrete-valued factors *object texture*, *table texture*, and *distractor objects*, we sample our $k = 10$ values for $\mathcal{F}^N$ from all possible training values specified in *Factor World*. *Distractor objects* also include a size scale (sampled from range $[0.3, 0.8]$), 2D rotation (sampled from range $[0, 2\pi]$), and 2D position (sampled from all possible positions on the table) as part of each value. For *object position*, we sample 2D xy positions from the range $[-0.1, 0.1]$ for both coordinates. We note that the *Pick Place* task includes


(a) *Object Position*


(b) *Object Texture*


(c) *Table Texture*


(d) *Camera Position*

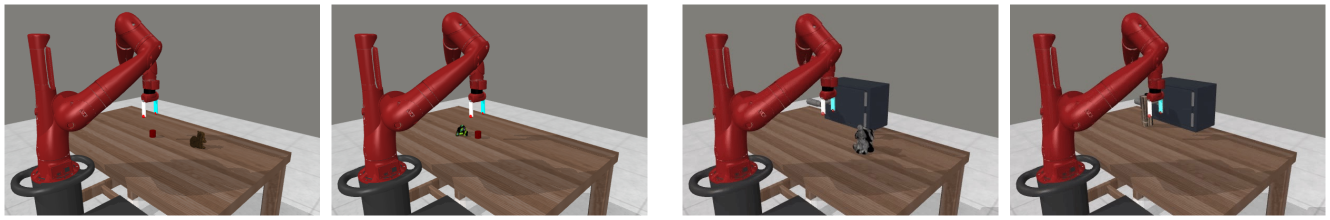

(e) *Distractor Objects*

Fig. 12: Visualization of our factors for the *Pick Place* (left) and *Door Open* (right) tasks from *Factor World*. We show two examples of values for each factor we consider.

a small amount of added noise to *object position* each episode, sampled uniformly from the range $[-0.03, 0.03]$. For *camera position*, we sample 3D xyz positions and 4D rotation quaternions all from the range $[-0.05, 0.05]$. When sampling our k values for each factor, we ensure that the scripted policy is able to solve the task for each value, because some values for *object position* and *distractor objects* can impede the task. In Fig. 12, we visualize two examples of values for each factor, for both the *Pick Place* and *Door Open* tasks. We note that each random seed in our evaluation includes a different set of $k = 10$ values sampled for each factor for $\mathcal{F}^N$.

C. Training

We use the same policy architecture and training hyperparameters from the original *Factor World* experiments [49]. We condition policies on the same observations (2 84x84 RGB images from 2 camera views without history, and proprioception), and use the same action space (3D end-effector position deltas and open/close gripper). Unlike the original *Factor World* experiments, we always use random shift augmentation. Unlike our real robot experiments, we train policies without goal conditioning, as we do not leverage diverse prior data in this setting, so task conditioning is not as essential.

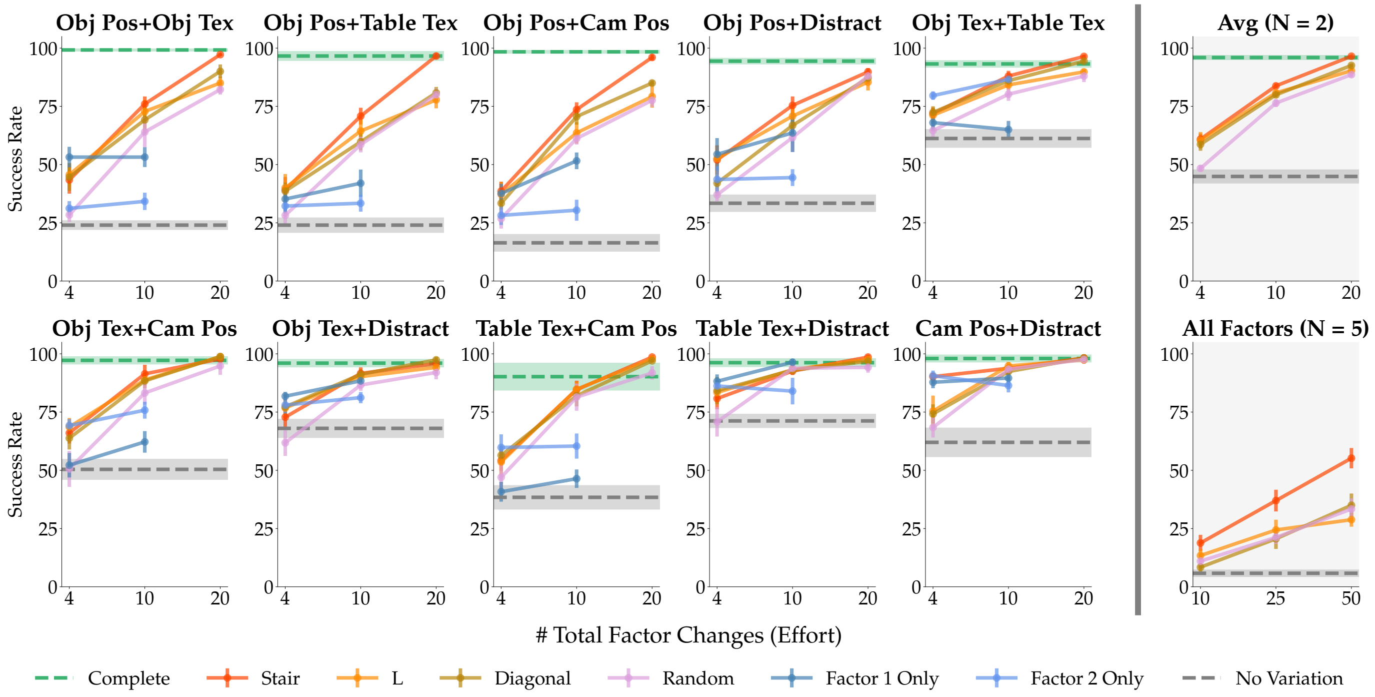


Fig. 13: Simulation results of data collection strategies for *Pick Place*, with additional color jitter augmentation. Overall performance across all strategies slightly improves, but the overall trends are similar to as without color jitter. Error bars represent standard error across 5 seeds.

D. Color Jitter Augmentation

We do not use color jitter augmentation in our main simulation experiments, because the original experiments for *Factor World* found this to reduce overall generalization, except for when training variation was very low [49]. However, here we include additional results with color jitter (in addition to random shift). We find that overall performance across all strategies does slightly improve, but the overall trends across strategies are similar to as without color jitter.

E. Accounting for Factor Value Similarity

In practice, data collection strategies should ideally account for similarity between factor values to improve generalization. For example, when training policies to be robust to the factor *object type*, it would be desirable to prioritize object diversity during data collection to generalize to as many objects as possible, rather than collecting data for overly similar objects. Here, we demonstrate how our proposed data collection framework could incorporate notions of factor similarity when available, through additional *Pick Place* experiments.

We consider the composition of *object position* and *camera position* in the $N = 2$ setting, as these are the only factors we study in simulation where computing similarity/distance is straightforward (we use Euclidean distance for 3D positions, and angular distance for camera rotation quaternions). We modify **Stair** to choose factor values using a similarity-aware strategy, rather than randomly as before. We do this by running a k-medoids algorithm on the set of 10 values for each factor, to determine which k values/medoids in the set minimize the sum of distances from each value to its nearest medoid. We determine k according to what is permissible under different factor change budgets (e.g., for 10 total changes in the $N = 2$ setting, each factor can have $k = \frac{10}{2} = 5$ values).

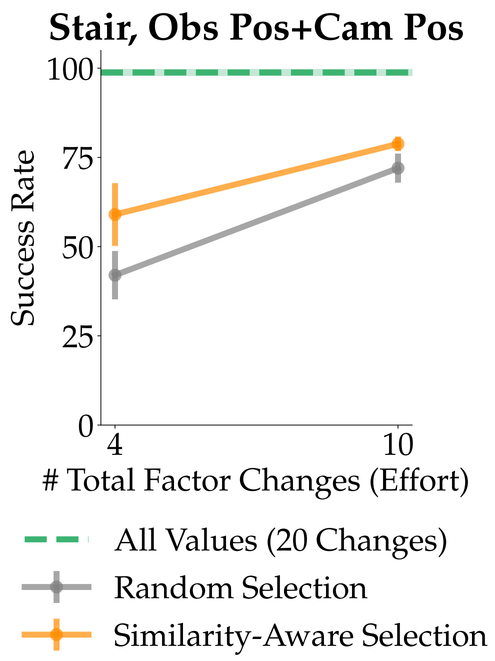


Fig. 14: Performance of the **Stair** strategy for the *Pick Place* task, when composing *object position* and *camera position*. We find that selecting factor values using a similarity-aware strategy outperforms random selection (as done in our main simulation results).

In Fig. 14, we compare using k-medoids for factor value selection (orange), and random selection as done in our main results (gray). We evaluate using the same procedure as in our main simulation results in Section IV-B, where the aim is to generalize to all factor value combinations. We find that

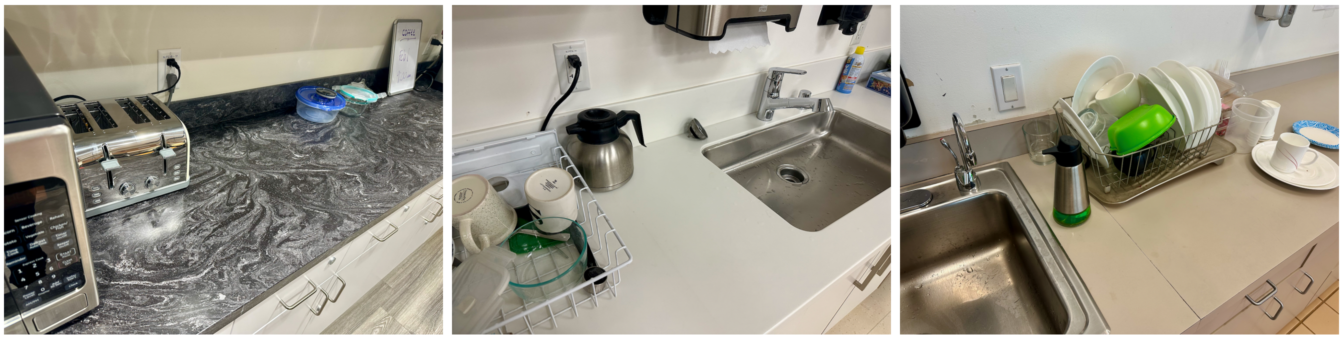

(a) *BaseKitch* (b) *CompKitch* (c) *TileKitch*

Fig. 15: Additional views of each kitchen in our real robot experiments. We name *CompKitch* and *TileKitch* after their countertop materials of composite and tile, respectively. These images were taken after our evaluations, so there may be some slight differences from then.

accounting for similarity does indeed improve upon random selection, although it does not match observing all factor values (green), which requires 20 factor changes.

We note that it often not straightforward to compute similarity metrics for other factors, and that while accounting for similarity can be easily incorporated into our data collection framework, it is mostly orthogonal to our study of composition. We believe it would be interesting for future work to investigate methods of computing similarity metrics for other factors, such as by using text and/or image embeddings, and leveraging this to further improve data collection.

F. *Computing Factor Changes*

For our results in Section IV-B, we compute the total number of factor changes for each strategy as follows. We assume the initial configuration of factor values requires n changes, one for each factor. For **Stair**, **L**, and **Single Factor**, we assume each new configuration of factor values requires 1 additional change. We note this could be a slight underestimate for **L** in practice, as this strategy could require some additional changes when finishing varying one factor and returning to the base factor values f^*, to begin data collection for varying another factor. For **Diagonal** and **Random**, we assume each new configuration of factor values requires n additional changes, as new values for all factors are resampled. We note that this could be a slight overestimate for **Random** in practice, as some of the resampled factor values may not change.

For each budget of factor value changes, we determine the amount of factor value configurations allowed for each strategy. We then divide the budget of total demonstrations by this to determine how many to collect for each configuration, with any remainder going to the last configuration.

APPENDIX C
REAL ROBOT EXPERIMENTS

A. *Robot Platform*

We use Logitech C920 webcams for our main and secondary third-person cameras. Our setup also includes the wrist camera used in BridgeData V2, and we experimented with using it in addition to our main third-person camera. While we found that it improved overall robustness when training from scratch, particularly for some factors that wrist cameras provide invariance to (e.g., *object position*, *table height*), policies still benefited from data coverage for these factors. More importantly, we also found that it was incompatible with using BridgeData V2 as prior data, reducing performance when using prior data compared to not using prior data at all. We believe this is because only a small fraction of BridgeData V2 contains wrist camera data, as similar negative results have been observed in other work that use prior robotic datasets with a small proportion of wrist camera data [35]. As we were only able to achieve successful transfer in our experiments in Section V-C by using prior data without a wrist camera, we decided to omit the wrist camera in all our experiments.

B. *Evaluation Protocol*

For our out-of-domain transfer experiments in Section V-C, we name our transfer kitchens *CompKitch* and *TileKitch* after their countertop materials of composite and tile, respectively. We provide additional views of these kitchens, as well as *BaseKitch*, in Fig. 15 to make the difference between these kitchens more apparent. These images were taken after our evaluations, so there may be some slight differences from then.

Our factor *table height* refers to the height of the object table (where objects are manipulated on) relative to the robot. However, we vary this factor in practice by adjusting the height of the mobile table the robot and third-person cameras are mounted on. This still changes the relative height of the robot with respect to the object table, the same as if the object table changed height. For example, in Section V-C, **Higher Table** refers to the object table being higher relative to the robot, which was achieved by lowering the robot's table.

We collect demonstrations using a Meta Quest 2 VR headset for teleoperation. All demonstrations are collected by a single experienced human teleoperator for consistency. We collect our **Stair** dataset by starting at f^*, and then varying factors cyclically in the order *object position*, *object type*, *container type*, *table height*, *table texture*. The order we vary values for each factor is the same as in Fig. 7, from top to bottom.

C. *Training*

We use the same ResNet-34 diffusion goal-conditioned policy architecture from the original BridgeData V2 experiments, except we condition on a history of 2 128x128 RGB image

Data Strategy	L									
Train Method	Bridge					From Scratch				
Factor 1 \ Factor 2	Object Pos	Object Type	Container Type	Table Height	Table Tex	Object Pos	Object Type	Container Type	Table Height	Table Tex
Object/Container Pos	N/A	**7/9**	**6/9**	**2/9**	**6/9**	N/A	5/9	2/9	0/9	3/9
Object Orientation	**2/9**	**5/9**	**5/9**	**1/9**	**3/9**	0/9	2/9	3/9	0/9	1/9
Overall	**36/81**					16/81				

TABLE VI: Additional real robot pairwise composition results for our "*put fork in container*" task, with additional factors *object/container position* and *object orientation*. Similarly as in our main pairwise composition results in Table I, leveraging BridgeData V2 as prior data significantly improves composition for these factors compared to training from scratch.

observations from a third-person view (in addition to a goal image from the same view), and use action chunking to predict the next 4 actions. However, we noticed better performance during inference by only executing the first predicted action, so we do this for all experiments unless otherwise stated. We share the same visual encoder across image observations in the history. We use the same 7D action space as the original experiments (6D end-effector pose deltas, and open/close gripper). We use the same data augmentation from the original experiments, which consists of random crops, random resizing, and color jitter.

For policies using BridgeData V2 as prior data, we first pre-train a model on BridgeData V2 using the original training hyperparameters for 2M gradient steps. As done in the original BridgeData V2 experiments, 10% of trajectories in this data is reserved for validation. We checkpoint every 50K steps, and choose the checkpoint with the lowest validation action prediction mean-squared error as the initialization for later fine-tuning. During fine-tuning, we use the same hyperparameters as during pre-training, except we train for only 300K gradient steps. Also, we do not use a validation set, and instead simply evaluate the final checkpoint. When co-fine-tuning, we train on a mixture of 75% in-domain data and 25% prior data.

D. Pairwise Composition for Additional Factors

We conduct experiments for two new factors: *object/container position* and *object orientation*. We visualize these factors and their values in Fig. 16. Note that *object/container position* involves new positions of both the fork and container, with more significant changes compared to our previous values for *object position*. We extended the **L** dataset for our original factors with 10 demonstrations for each new factor value, re-trained policies on this extended dataset, and evaluated pairwise composition with these new factors. Unlike our previous results in Section V-B, we do not evaluate the **No Variation** Bridge policy, because we found it was unable to succeed at all with shifts for these factors in isolation, so there was no potential for composition.

While these policies worked for *object/container position*, we found they was unable to perform the task for the *object orientation* shifts we considered in isolation. We hypothesize this could be because different values for this factor have relatively small changes in their visual observations, but require

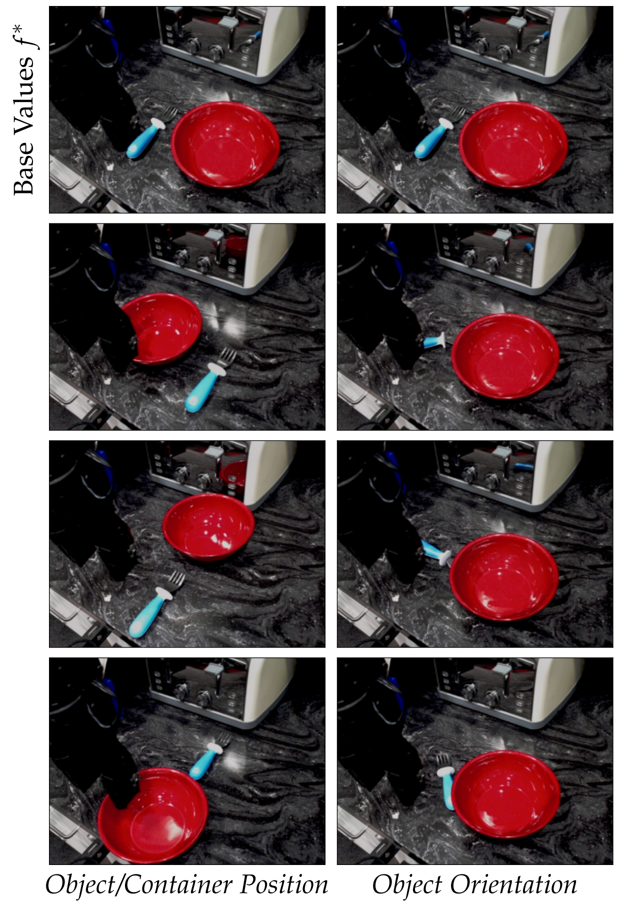


Fig. 16: Visualization of our additional real robot factors in *BaseKitch*. The top row shows our base factor values f^*. The other rows show all deviations from f^* by one factor value.

significantly different behavior, which can make it challenging to learn when to apply the correct behavior. Therefore, for our evaluation on composing *object orientation*, we trained separate policies where we balance training batches such that 50% of in-domain data consists of data for *object orientation*.

We report these results in Table VI. We find that with prior data, *object/container position* achieves similar composition as our original *object position* factor, with a success rate of **20/36** for both. However, when training from scratch, the policy is able to compose this new factor more effectively than the

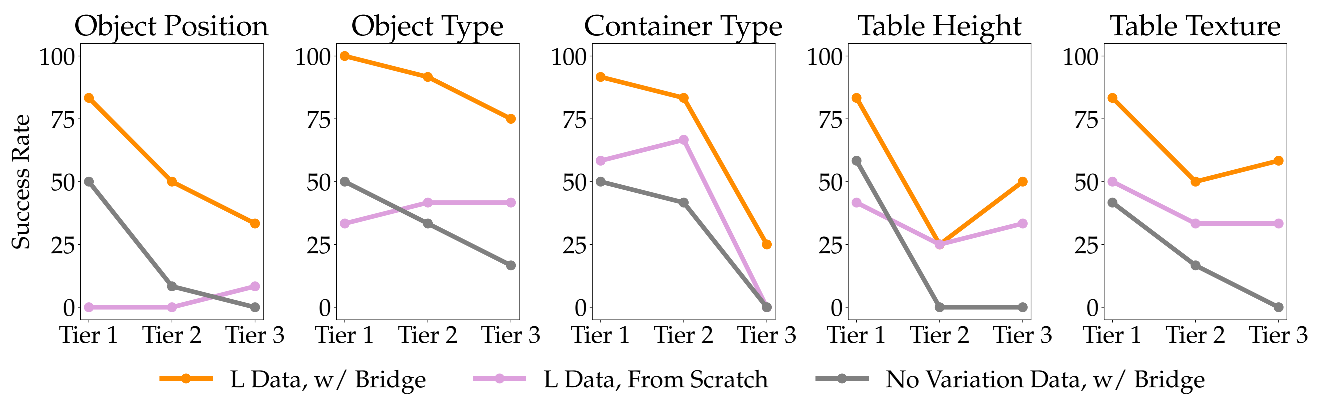


Fig. 17: Per-factor value success rates of policies from our pairwise composition results in Table I. Factor values are placed in tiers, where increasing tiers are more dissimilar from the base factor values f^*. Composition is generally more challenging for factor values that are more dissimilar from the base factor values.

Factor	Tier 1	Tier 2	Tier 3
Object Pos	Down (3)	Up (4)	Left (2)
Object Type	Wooden (3)	Gray (4)	Plastic (2)
Container Type	Blue Plate (4)	Pink Bowl (3)	White Cup (2)
Table Height	Higher 5cm (2)	Lower 5cm (3)	Lower 8cm (4)
Table Tex	Brown Wood (2)	Gift Wrap (4)	White Marble (3)

TABLE VII: Tiers for each factor value used in Fig. 17, determined using our success rate-based similarity metric as described in Appendix C-E. In parentheses next to each factor, we provide the row number where the factor value is visualized in Fig. 7.

original, achieving a success rate of **10/36** compared to **1/36**. This could be because different values for *object/container position* are more visually distinguishable, and their required behavior is also significantly more different, which could make it easier to learn when to apply the correct behavior.

Our policies can sometimes compose *object orientation*, although composition for this is the weakest compared to the other factors we study. This could be due to the aforementioned challenges with learning for this factor, as well as because our data balancing may have insufficiently represented the other factors.

E. Factor Similarity Analysis

Here, we provide additional analysis on when our policies are able to compose factor values from our pairwise evaluation in Section V-B. To do this, for each factor, we consider how similar each of the non-base factor values we consider are to the base factor value, with respect to a policy's ability to generalize across factor values.

To compute a similarity metric, we consider the success rates from our results in Table I, in particular for the policy trained on **No Variation** data and BridgeData V2. We then obtain a success rate for each non-base factor value, by aggregating the results for all factor value pairs that contain that factor value. We use this success rate as our similarity metric, where higher success rates indicate greater similarity, because this captures how well a policy trained on base factor values generalizes to other factor values.

Using this similarity metric, we rank the non-base factor values for each factor as *Tier 1*, *Tier 2*, or *Tier 3*, where a higher tier is more dissimilar from the base factor value. We list the tiers for each factor value in Table VII. We then take the aggregated per-factor value success rates for each policy from our results in Table I (where we aggregate success rates as we do for computing the similarity metric), and plot this against our factor value tiers in Fig. 17.

We find that the policies trained on **L** data (orange and pink lines) generally achieve lower compositional success rates for factor values that are more dissimilar from the base factor values f^*, suggesting that composition is more challenging for these more dissimilar values, although this trend is not strictly monotonic. We note that part of this effect could be because our success rate-based similarity metric may also capture how challenging factor values are in general.

F. R3M

We use the ResNet-50 version of R3M. When training, we pre-compute representations beforehand, and standardize the dataset such that each feature has mean 0 and standard deviation 1 (similar to the batch normalization used in the original R3M experiments). We use the training dataset mean and standard deviation for normalization during inference. We feed normalized representations to the same diffusion policy head architecture used when learning end-to-end, except we do not use goal-conditioning. We increase the amount of training gradient steps from 300K to 500K, to reduce training loss. We also tried 1M gradient steps, which reduces loss even further, but this resulted in worse performance. Instead of executing only the first predicted action as with the end-to-end policies, we execute all 4, which we found to slightly reduce jitteriness. Like when learning end-to-end, we verify our R3M policies are able to succeed with base factor values f^* in *BaseKitch*.

G. VC-1

We train and evaluate VC-1 policies using the same procedure as with R3M, except using the ViT-L version of VC-1 instead of R3M.