Skill
Pick-and-place (includes reorienting objects in place)
Pushing objects
Wiping (e.g., wiping the table with a cloth)
Sweeping (e.g., sweeping beans into a pile)
Stacking
Folding cloths
Opening and closing drawers
Opening and closing doors
Opening and closing cardboard box flaps
Twisting knobs
Flipping switches
Zipping and unzipping
Turning levers

Table 6 The full set of skills in BridgeData V2. We define a skill to be a group of trajectories that require similar motions from the robot but may include different objects and start/end positions.

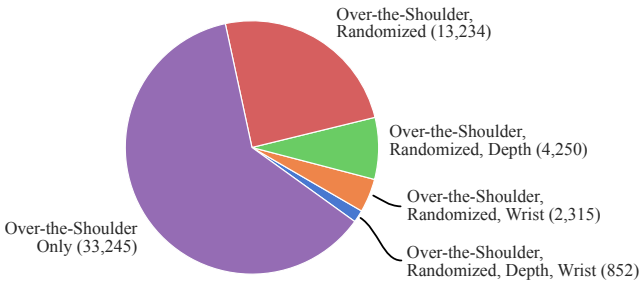

Figure 6 Breakdown of the entire dataset, including the autonomously collected data, by what camera views are included. "Over-the-shoulder" refers to the primary fixed camera, and "randomized" refers to the two alternative camera views that are randomized by the data collectors every 50 trajectories. "Depth", when present, is from the same perspective as the primary fixed camera. "Wrist" refers to the wide-angle wrist-mounted camera. More cameras were added to the hardware setup throughout data collection, so the majority of the data only includes the primary fixed camera view, and very little data currently includes all 4 views. However, now that the hardware is in place, more and more data will include all 4 views as the dataset continues to grow.

A Data statistics

We provide a full list of the skills in our dataset (following our definition of skill in Section 3.3) in Table 6. We provide a breakdown of which portions of the dataset include which sensors in Figure 6.

B Learning method implementation details

Below we list relevant implementation details for each method. The complete set of evaluation tasks is shown in Figure 7.

All the goal-conditioned methods take in both an observation and goal. The observation and goal are stacked channel wise before being passed into a ResNet image encoder. During training, the goal associated with an observation is selected by uniformly sampling an observation from the future timesteps in the trajectory.

Figure 7 The complete set of evaluation tasks. The seen tasks are (clockwise): sweeping beans into a pile, putting a corn cob in a pot, opening a drawer, putting an eggplant in a pot, stacking a block, flipping a pot upright, folding a cloth, and putting a carrot on a plate. The unseen tasks are (left to right): putting a marker in a bowl, sweeping rice into a pile, folding a thick cloth, putting a mushroom in a pot, putting a spoon on a cloth, and wiping the table with a cloth.

.

B.1 Goal-conditioned behavior cloning

Our goal-conditioned policy uses 3 256-unit fully connected layers to transform the image encoding of the observation and goal into a robot action.

B.2 Diffusion goal-conditioned behavior cloning

In our implementation, we adopt the behavior cloning with diffusion strategy from the IDQL [41]. However, we do not learn a value function. IDQL proposes an architecture using a Layer-Norm+ResNet backbone, designed to enhance modeling within continuous spaces. This approach improves behavior cloning by boosting model expressivity and mitigating outliers. We use the DDPM (Denoising Diffusion Probabilistic Models) style objective as introduced by Ho et al. [44].

B.3 Action Chunking with Transformers

We use the same ACT hyperparameters as the original paper [17] except tuning the chunk size. The original ACT has chunk size 100 to accommodate for high-frequency control (50Hz) and long trajectories (1000 steps), while in our case the trajectory is much shorter at around 50-100 steps. We therefore reduce chunk size to 5 and noted better performance. In addition, ACT was originally proposed as a single task method; we modify ACT to make it goal-conditioned. The ACT policy is trained on a consumer grade workstation with 1x Nvidia 2080Ti GPU for 3 days.

B.4 Contrastive RL

Our implementation of contrastive RL mostly follows the design decisions described in [47] except for the following changes. First, given the observation and goal images, we feed them separately through a ResNet-34 encoder instead of a 3-layer CNN image encoder to get output encodings. Those image encodings then pass through two MLPs to get representations of the observation and the goal. Second, we share the ResNet-34 encoder between the value function and the policy. Intuitively, this encourages sharing of the causal information learned by the representations and speeds up learning. This ResNet backbone helps the method to consume large amounts of training data. Third, we increase the GCBC regularization coefficient to 0.2 to avoid sampling out-of-distribution actions and use the same batch size as other methods to make fair comparisons. Our contrastive RL objecive retains the temporal-difference (TD) style used in [46].

B.5 Language-conditioned behavior cloning

Our implementation of LCBC uses a ResNet-34 with FiLM conditioning. The language instruction is first encoded with a frozen MUSE encoder and passed through 2 fully connected layers. The image observation is then passed into the ResNet, which is conditioned on the language embedding using FiLM layers. FiLM layers are applied at the end of every ResNet block to condition on language throughout the network. Finally, the image encoding is passed into a fully connected network to

predict the action distribution. We predict a multivariate Gaussian for the action with fixed standard deviations.

The policy is trained on batch sizes of 128 and using the Adam optimizer with a learning rate of 3e-4. We use a linear warmup schedule with 2000 steps and train for 100k steps.

B.6 RT-1

We use the same hyper-parameters as the original RT-1 paper, except for increasing the sequence length of the transformer from 6 to 15, to accommodate for the longer episode length. We scale each action dimension to range -1 and 1 and use a vocabulary size of 256 to tokenize the actions. All other design choices and hyper-parameters remain the same as in the RT-1 paper.

C Hardware setup

We use an Intel RealSense D435 RGBD camera as a fixed over-the-shoulder camera view and two Logitech C920 webcams to capture alternative camera views that are randomized during data collection. For the wrist camera, we designed a custom 3D printed mount to attach a Raspberry Pi camera module to the gripper. We used a Meta Quest 2 VR headset to teleoperate the robot.