

## 459 A Simulation Task Details

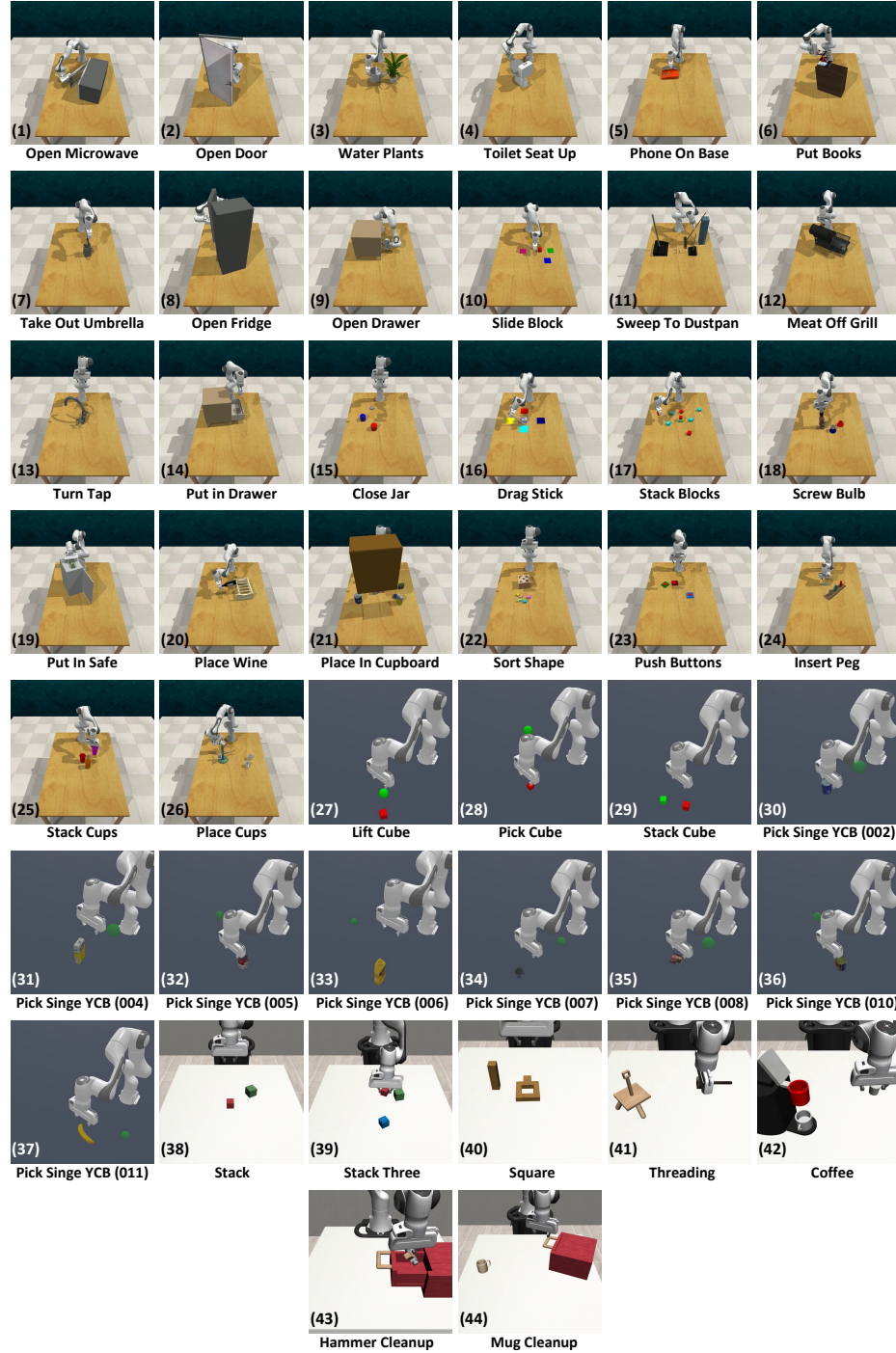


Figure 5: **Simulation Tasks.** Our simulation experiments encompass 26 tasks (1-26) from RL-Bench, 4 tasks (27-37, where 30-37 are 8 different YCB [62] objects of task *PickSingleYCB*) from ManiSkill2, and 7 tasks (38-44) from MimicGen.

Our simulation experiments are conducted on 3 robot learning benchmarks: RL-Bench [12], ManiSkill2 [13], and MimicGen [14]. See Figure 5 for an overview of the simulation tasks. In these simulations, all cameras have a resolution of  $128 \times 128$ . In the following, we will provide a detailed examination of tasks from the three benchmarks.

## A.1 RL-Bench Tasks

We utilize 26 RL-Bench tasks, including 8 tasks used in SGR [10] and 18 tasks used in PerAct [16] and RVT [17]. For tasks with multiple variations, we use the first variation. In RL-Bench, we use 5 demonstrations per task, unless specified otherwise. Given that SGR and PerAct provide detailed descriptions of these RL-Bench tasks, we omit these details here for simplicity.

## A.2 ManiSkill2 Tasks

We utilize 4 ManiSkill2 tasks, each described in detail as follows. (1) **Lift Cube**: Pick up a red cube and lift it to a specified height. (2) **Pick Cube**: Pick up a red cube and move it to a target position. (3) **Stack Cube**: Pick up a red cube and place it onto a green cube. (4) **Pick Single YCB**: Pick up a YCB [62] object and move it to the target position. In our experiments, we use 8 YCB objects (excluding those that are too difficult to pick up): *002\_master\_chef\_can*, *004\_sugar\_box*, *005\_tomato\_soup\_can*, *006\_mustard\_bottle*, *007\_tuna\_fish\_can*, *008\_pudding\_box*, *010\_potted\_meat\_can*, *011\_banana*. For the first three tasks, we utilize 50 demonstrations per task, while for the last one (*Pick Single YCB*), we employ 50 demonstrations per YCB object.

## A.3 MimicGen Tasks

We utilize 7 MimicGen tasks with 50 demonstrations per task, all employing the initial distribution  $D_1$ , which presents a broader and more challenging range. The details are as follows: (1) **Stack**: Stack a red block on a green one. (2) **Stack Three**: Similar to Stack, but with an additional step of stacking a blue block on the red one. (3) **Square**: Pick up a square nut and place it on a peg. (4) **Threading**: Pick up a needle and thread it through a hole in a tripod. (5) **Coffee**: Pick up a coffee pod, insert it into the coffee machine, and close the machine hinge. (6) **Hammer Cleanup**: Open a drawer, pick up a hammer, place it back into the drawer, and close the drawer. (7) **Mug Cleanup**: Similar to Hammer Cleanup, but with a mug.

## B SGRv2 Details

### B.1 Architecture Details

**Input Data.** The SGRv2 model takes as input RGB images  $\{I_k\}_{k=1}^K$  of size  $H \times W$  and corresponding depth images of the same size from multiple camera views. Point clouds are generated from these depth images using known camera extrinsics and intrinsics. A crucial aspect is the alignment of the RGB images with the point clouds, ensuring a precise one-to-one correspondence between elements in the two data forms. For keyframe control, the point cloud is represented in the robot’s base frame. In contrast, for dense control—inspired by FrameMiner [43]—the point cloud is transformed into the end-effector frame to simplify computation and enhance performance.

For keyframe control, the model additionally receives proprioceptive data  $z$ , which includes four scalar values: gripper open state, left finger joint position, right finger joint position, and action sequence timestep. In dense control, proprioceptive data is not utilized. Additionally, following SGR [10], if a task comes with language instruction  $S$ , this also forms part of the model’s input.

**Other Details.** Following SGR [10], we use CLIP-ResNet-50 as the image encoder for the semantic branch. For the 3D encoder-decoder, we employ PointNeXt-XL. The output from the encoder-decoder is a point-wise feature, denoted as  $f_i^{\text{raw}} \in \mathbb{R}^C$  for the  $i$ -th point, where the feature dimension  $C$  is 64. We apply a linear layer followed by a ReLU activation to produce a processed point-wise feature  $f_i$ , increasing the feature dimension to 256. We then predict the relative position  $\Delta p(f_i)$ ,

magnitude  $m(f_i)$  (for dense control), rotation  $r(f_i)$ , gripper open state  $o(f_i)$ , and collision indicator  $c(f_i)$  (for keyframe control) of the  $i$ -th point, where  $\Delta p, m, r, o, c$  are 3-layer MLPs. Note that when representing the ground-truth actions as one-hot vectors—such as rotation, gripper open state, and collision indicators in keyframe control—the action predictions correspond to the output probabilities following the softmax layer. Finally, for each action component, we assign a learned weight  $w_*(f_i)$  to each point, where  $w_*$  represents separate 3-layer MLPs with softmax normalization across the points dimension.

## B.2 SGR Details

SGRv2 is built upon SGR [10], which we briefly introduced in Section 3.1. Here, we provide a detailed description of SGR’s three components: semantic branch, geometric branch, and fusion network.

**Semantic Branch.** Using a collection of RGB images  $\{I_k\}_{k=1}^K$  from  $K$  calibrated cameras, they initially apply a frozen pre-trained 2D model  $\mathcal{G}$ , such as CLIP’s visual encoder, to extract multi-view image features  $\{\mathcal{G}(I_k)\}_{k=1}^K$ . When a language instruction  $S$  accompanies a task, they utilize a pre-trained language model  $\mathcal{H}$ , like CLIP’s language encoder, to generate the language features  $\mathcal{H}(S)$ . They align these image features  $\mathcal{G}(I_k)$  with the language features  $\mathcal{H}(S)$  using a visual grounding module, producing  $\{M_k\}_{k=1}^K$ . Subsequently, they rescale the visual or aligned feature maps to the dimensions of the original images through bilinear interpolation and reduce their channels by  $1 \times 1$  convolution, generating a set of features  $\{F_k\}_{k=1}^K$ , where each  $F_k \in \mathbb{R}^{H \times W \times C_1}$ . These high-level semantic features are then back-projected into 3D space to form point-wise features for the point cloud, expressed as  $F_{\text{sem}} \in \mathbb{R}^{N \times C_1}$ , where  $N = K \times H \times W$ .

**Geometric Branch.** They construct the initial point cloud coordinates  $P = \{p_i\}_{i=1}^N \in \mathbb{R}^{N \times 3}$  and RGB features  $F_c \in \mathbb{R}^{N \times 3}$  using multi-view RGB-D images and camera parameters (i.e., camera intrinsics and extrinsics). Optionally, they append a  $D$ -dimensional vector, derived from robot proprioceptive data  $z$  via a linear layer, to each point feature. They then process the point cloud coordinates  $P$  and features  $F_c$  through a hierarchical PointNeXt encoder, extracting compact geometric coordinates  $P' \in \mathbb{R}^{M \times 3}$  and features  $F'_c \in \mathbb{R}^{M \times C_2}$  ( $M < N$ ).

**Fusion Network.** To merge the two complementary branches, they first subsample the point-wise semantic features  $F_{\text{sem}}$  using the same point subsampling procedure as in the geometric branch, resulting in  $F'_{\text{sem}} \in \mathbb{R}^{M \times C_1}$ . They then perform a channel-wise concatenation of the semantic and geometric features to form  $F_{\text{fuse}} = \text{Concat}(F'_{\text{sem}}, F'_c) \in \mathbb{R}^{M \times (C_1 + C_2)}$ . Finally, the fused features are processed through several set abstraction blocks [37, 15], enabling a cohesive modeling of the cross-modal interaction between 2D semantics and 3D geometric information.

## B.3 Training Details

**Losses.** As illustrated in Section 3.3, in keyframe control, our loss objective is as follows:

$$\mathcal{L}_{\text{keyframe}} = \alpha_1 \mathcal{L}_{\text{pos}} + \alpha_2 \mathcal{L}_{\text{rot}} + \alpha_3 \mathcal{L}_{\text{open}} + \alpha_4 \mathcal{L}_{\text{collide}}, \quad (3)$$

where  $\mathcal{L}_{\text{pos}}$  is L1 loss, and  $\mathcal{L}_{\text{rot}}$ ,  $\mathcal{L}_{\text{open}}$ , and  $\mathcal{L}_{\text{collide}}$  are cross-entropy losses. In our experiments, we set  $\alpha_1 = 300$  and  $\alpha_2 = \alpha_3 = \alpha_4 = 1$ .

In dense control, our loss objective is:

$$\mathcal{L}_{\text{dense}} = \beta_1 (\mathcal{L}_{\text{dir}} + \mathcal{L}_{\text{mag}}) + \beta_2 \mathcal{L}_{\text{rot}} + \beta_3 \mathcal{L}_{\text{open}} + \beta_4 \mathcal{L}_{\text{reg}}, \quad (4)$$

where  $\mathcal{L}_{\text{dir}}$ ,  $\mathcal{L}_{\text{mag}}$  and  $\mathcal{L}_{\text{rot}}$  are MSE losses,  $\mathcal{L}_{\text{open}}$  is cross-entropy loss, and  $\mathcal{L}_{\text{reg}}$  is smoothness regularization loss. In our experiments, we set  $\beta_1 = 10$ ,  $\beta_2 = \beta_3 = 1$  and  $\beta_4 = 0.3$ .

**Data Augmentation.** (1) **Translation and rotation perturbations:** in keyframe control, the training samples is augmented with  $\pm 0.125$  m translation perturbations and  $\pm 45^\circ$  yaw rotation perturbations. (2) **Color drop** is to randomly replace colors with zero values. This technique serves as a powerful augmentation for PointNeXt [15], leading to significant enhancements in the performance

of tasks where color information is available. (3) **Feature drop**: Color drop randomly replaces colors with zero values, which results in both the RGB and semantic features becoming constant. However, there are certain tasks where colors play a crucial role, and disregarding color information in these tasks would make them unsolvable. To address this issue, we propose *feature drop*. Specifically, this involves randomly replacing the semantic features with zero values, while keeping the RGB values unchanged. (4) **Point resampling** is a widely used technique in point cloud data processing that adjusts the density of the point cloud. It involves selecting a subset of points from the original dataset to create a new dataset with a modified density. Firstly, we filter out points outside the workspace. Then in keyframe control, we resample 4096 points from the point cloud using farthest point sampling (FPS), while in dense control, we resample 1200 points using the same method. (5) **Demo augmentation** [32] [16], used in keyframe control, captures transitions from intermediate points along a trajectory to keyframe states, rather than from the initial state to the keyframe state. This approach significantly increases the volume of the training data.

**Hyperparameters.** The configuration of hyperparameters applied in our studies are shown in Table 4. For each task, the experiments are conducted on a single NVIDIA GeForce RTX 3090 GPU.

Table 4: Hyper-parameters used in our simulation experiments.

Config	Keyframe Control	Dense Control
Training iterations	20,000	100,000
Lerning rate	0.003	0.0003
Batch size	16	16
Optimizer	AdamW	AdamW
Lr Scheduler	Cosine	Cosine
Warmup step	200	0
Weight decay	$1 \times 10^{-6}$	$1 \times 10^{-6}$
Color drop	0.4	0
Feature drop	0	0.4
Number of input points	4096	1200

## C Real-Robot Details

### C.1 Real-Robot Setup

For our real-robot experiments, we use a Franka Emika Panda manipulator equipped with a parallel gripper. We utilize keyframe control, and the motion planning is executed through MoveIt<sup>2</sup>. Perception is achieved through an Intel RealSense L515 camera, positioned in front of the scene. The camera generates RGB-D images with a resolution of  $1280 \times 720$ . We leverage the `realsense-ros`<sup>3</sup> to align depth images with color images. The extrinsic calibration between the camera frame and robot base frame is carried out using the MoveIt calibration package.

When preprocessing the RGB-D images, we resize the  $1280 \times 720$  images to  $256 \times 256$  using nearest-neighbor interpolation. We choose this interpolation method instead of others, like bilinear interpolation, because the latter can introduce artifacts into the depth map, resulting in a noisy point cloud. Following these steps enables us to process RGB-D images as we do in our simulation experiments. It is essential to adjust the camera’s intrinsic parameters appropriately after resizing the images. We train SGRv2 for 40,000 training steps and use the final checkpoint for evaluation.

### C.2 Real-Robot Tasks

Our real-robot experiments involve three tasks: Tidy Up the Table, Make Coffee, and Move Color Cup to Target. The first two are long-horizon tasks, while the last is a generalization task. We

<sup>2</sup><https://moveit.ros.org/>

<sup>3</sup><https://github.com/IntelRealSense/realsense-ros>



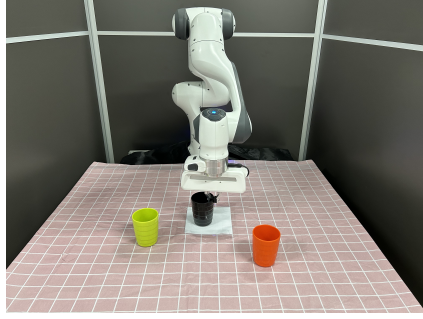


Figure 6: Real-robot generalization task.

provide details of the task design as follows. The videos of experiment rollouts can be found on the anonymous website: <https://robot-sgrv2.github.io/>

**Tidy Up the Table** (as shown in Figure 4 Top Left) is to place the clutter on the table in its appropriate locations. The task consists of 6 sub-tasks, each detailed as follows: (1) *Put trash in trash can*: Pick up the trash and place it in the trash can. (2) *Put socks in box*: Pick up the socks and place them in the box. (3) *Put marker in pen holder*: Pick up the marker and place it in the pen holder. (4) *Open drawer*: Grasp the drawer handle and pull it open. (5) *Put lollipop in drawer*: Pick up the lollipop and place it into the drawer. (6) *Close drawer*: Push the drawer closed.

**Make Coffee** (as shown in Figure 4 Bottom Left) is to make pour-over coffee. This task is composed of 4 sub-tasks, each described in detail as follows: (1) *Turn on coffee machine*: Press the button on the coffee machine to activate it. (2) *Put funnel onto carafe*: Pick up the funnel and place it onto the carafe. (3) *Pour powder into funnel*: Pick up the powder holder and pour the powder into the funnel. (4) *Pour water*: Pick up the kettle and pour the water onto the powder in the funnel.

**Move Color Cup to Target** (as shown in Figure 6) is to select the target color cup from three cups and move it to the white area. The target color is indicated through language instructions. We have 6 cups of different colors: *white, red, yellow, orange, black, and green*. Each scenario involves one target cup and two distractor cups of different colors. We collect five demonstrations for each of the first 4 colors and test the model on both 4 seen and 2 unseen scenarios.

## D Additional Results

For our simulation experiments using the SGRv2 on RL Bench with 5 demonstrations (mentioned in Table 1) and on ManiSkill2 and MimicGen with 50 demonstrations (mentioned in Table 2), we employed 3 random seeds to ensure the reliability of our results. In the main body of the paper, we present averaged results for clarity. Here we include both the mean and standard deviation derived from our simulation results. The results for RL Bench are shown in Table 5, and the results for ManiSkill2 and MimicGen are presented in Table 6.

We also report the ablations mentioned in Table 3 for each task in Table 7.

Method	Avg. Success $\uparrow$	Open Microwave	Open Door	Water Plants	Toilet Seat Up	Phone On Base	Put Books	Take Out Umbrella	Open Fridge
R3M	4.7	0.9 $\pm$ 0.6	36.4 $\pm$ 3.7	2.9 $\pm$ 3.7	15.5 $\pm$ 2.1	0.0 $\pm$ 0.0	0.5 $\pm$ 0.9	5.2 $\pm$ 8.7	3.2 $\pm$ 1.4
PointNeXt	25.3	7.1 $\pm$ 6.3	60.9 $\pm$ 5.2	5.6 $\pm$ 4.6	49.9 $\pm$ 14.7	46.4 $\pm$ 4.9	57.5 $\pm$ 8.2	37.5 $\pm$ 2.3	9.2 $\pm$ 4.0
PerAct	22.3	4.3 $\pm$ 7.0	59.6 $\pm$ 16.0	28.5 $\pm$ 3.1	69.3 $\pm$ 11.1	0.0 $\pm$ 0.0	25.1 $\pm$ 4.4	75.9 $\pm$ 7.0	3.1 $\pm$ 1.3
SGR	23.6	6.4 $\pm$ 2.2	55.3 $\pm$ 3.7	24.9 $\pm$ 8.2	30.7 $\pm$ 9.2	47.2 $\pm$ 1.4	29.3 $\pm$ 5.2	36.3 $\pm$ 6.4	7.1 $\pm$ 1.5
RVT	40.4	18.3 $\pm$ 1.8	71.2 $\pm$ 2.8	34.8 $\pm$ 3.3	47.6 $\pm$ 6.7	62.3 $\pm$ 1.4	46.5 $\pm$ 10.9	85.3 $\pm$ 4.5	24.0 $\pm$ 4.2
SGRv2 (ours)	<b>53.2</b>	27.2 $\pm$ 2.0	76.8 $\pm$ 8.0	38.0 $\pm$ 1.7	89.6 $\pm$ 2.8	84.1 $\pm$ 4.5	63.7 $\pm$ 11.8	74.5 $\pm$ 5.5	13.2 $\pm$ 3.4
Method	Open Drawer	Slide Block	Sweep To Dustpan	Meat Off Grill	Turn Tap	Put In Drawer	Close Jar	Drag Stick	Stack Blocks
R3M	0.0 $\pm$ 0.0	24.0 $\pm$ 8.8	0.4 $\pm$ 0.4	0.1 $\pm$ 0.2	26.1 $\pm$ 7.2	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.3 $\pm$ 0.5	0.0 $\pm$ 0.0
PointNeXt	21.7 $\pm$ 20.4	59.5 $\pm$ 22.1	42.0 $\pm$ 34.7	59.9 $\pm$ 17.8	48.7 $\pm$ 13.4	17.1 $\pm$ 27.8	36.0 $\pm$ 4.6	18.5 $\pm$ 32.1	1.9 $\pm$ 1.6
PerAct	56.4 $\pm$ 18.0	47.5 $\pm$ 24.3	2.8 $\pm$ 0.4	85.9 $\pm$ 6.9	8.0 $\pm$ 7.5	0.1 $\pm$ 0.2	0.5 $\pm$ 0.6	10.3 $\pm$ 6.4	1.7 $\pm$ 0.6
SGR	31.9 $\pm$ 6.2	72.0 $\pm$ 27.1	43.6 $\pm$ 8.4	52.7 $\pm$ 5.1	34.4 $\pm$ 7.4	8.3 $\pm$ 9.2	13.3 $\pm$ 5.6	64.4 $\pm$ 11.4	0.0 $\pm$ 0.0
RVT	75.1 $\pm$ 2.6	85.1 $\pm$ 2.2	19.6 $\pm$ 17.4	90.5 $\pm$ 2.2	38.4 $\pm$ 5.4	19.6 $\pm$ 5.5	25.2 $\pm$ 3.6	45.7 $\pm$ 10.9	8.8 $\pm$ 4.2
SGRv2 (ours)	81.3 $\pm$ 3.1	100.0 $\pm$ 0.0	61.5 $\pm$ 7.2	96.5 $\pm$ 3.9	87.9 $\pm$ 6.9	75.9 $\pm$ 3.6	25.6 $\pm$ 2.2	94.9 $\pm$ 0.6	17.5 $\pm$ 3.0
Method	Screw Bulb	Put In Safe	Place Wine	Put In Cupboard	Sort Shape	Push Buttons	Insert Peg	Stack Cups	Place Cups
R3M	0.0 $\pm$ 0.0	0.3 $\pm$ 0.2	0.4 $\pm$ 0.4	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	6.8 $\pm$ 3.7	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
PointNeXt	4.1 $\pm$ 1.5	12.1 $\pm$ 4.2	31.5 $\pm$ 4.5	3.3 $\pm$ 0.6	0.4 $\pm$ 0.4	22.0 $\pm$ 38.1	0.1 $\pm$ 0.2	4.4 $\pm$ 3.8	0.4 $\pm$ 0.4
PerAct	4.4 $\pm$ 5.2	0.9 $\pm$ 0.9	8.7 $\pm$ 1.7	0.4 $\pm$ 0.0	0.4 $\pm$ 0.4	83.1 $\pm$ 5.3	1.9 $\pm$ 1.2	0.1 $\pm$ 0.2	0.7 $\pm$ 0.6
SGR	0.9 $\pm$ 0.8	16.9 $\pm$ 2.2	24.7 $\pm$ 5.8	0.1 $\pm$ 0.2	0.1 $\pm$ 0.2	12.0 $\pm$ 1.4	0.1 $\pm$ 0.2	0.0 $\pm$ 0.0	1.1 $\pm$ 0.9
RVT	24.0 $\pm$ 3.8	30.7 $\pm$ 4.9	92.7 $\pm$ 0.6	5.6 $\pm$ 2.1	1.6 $\pm$ 0.7	90.4 $\pm$ 2.9	4.0 $\pm$ 0.0	3.1 $\pm$ 0.6	1.2 $\pm$ 0.7
SGRv2 (ours)	24.1 $\pm$ 0.6	55.6 $\pm$ 8.0	53.1 $\pm$ 7.4	20.3 $\pm$ 9.2	1.9 $\pm$ 0.6	93.2 $\pm$ 5.3	4.1 $\pm$ 1.4	21.3 $\pm$ 11.8	1.6 $\pm$ 0.7

Table 5: RL Bench results (%) on 5 demonstrations with mean and standard deviation.

Method	Avg. Success $\uparrow$	Avg. Rank $\downarrow$	LiftCube	PickCube	StackCube	PickSingleYCB
PointNeXt	16.8	2.5	50.8 $\pm$ 15.2	4.7 $\pm$ 0.4	10.6 $\pm$ 4.3	1.1 $\pm$ 0.1
SGR	14.9	2.5	26.9 $\pm$ 4.0	12.2 $\pm$ 3.1	3.5 $\pm$ 2.2	17.0 $\pm$ 0.2
SGRv2 (ours)	<b>55.8</b>	<b>1.0</b>	80.5 $\pm$ 7.3	72.9 $\pm$ 4.1	27.7 $\pm$ 4.3	42.2 $\pm$ 2.3

Method	Avg. Success $\uparrow$	Avg. Rank $\downarrow$	Stack	StackThree	Square	Threading	Coffee	HammerCleanup	MugCleanup
PointNeXt	13.6	2.9	56.1 $\pm$ 6.4	3.7 $\pm$ 1.4	0.9 $\pm$ 0.5	3.6 $\pm$ 2.2	12.0 $\pm$ 5.2	11.7 $\pm$ 2.8	7.1 $\pm$ 0.9
SGR	14.2	2.0	50.8 $\pm$ 7.7	5.6 $\pm$ 1.7	1.3 $\pm$ 0.5	4.0 $\pm$ 0.8	14.1 $\pm$ 2.0	14.1 $\pm$ 1.7	9.7 $\pm$ 2.4
SGRv2 (ours)	<b>26.0</b>	<b>1.0</b>	81.2 $\pm$ 4.4	37.9 $\pm$ 1.5	2.8 $\pm$ 0.7	6.7 $\pm$ 2.0	27.9 $\pm$ 7.0	16.1 $\pm$ 3.9	9.7 $\pm$ 2.7

Table 6: ManiSkill2 and MimicGen results (%) on 50 demonstrations with mean and standard deviation.

Method	Avg. Success $\uparrow$	Open Microwave	Open Door	Water Plants	Toilet Seat Up	Phone On Base	Put Books	Take Out Umbrella	Open Fridge
SGRv2	53.2	27.2	76.8	38.0	89.6	84.1	63.7	74.5	13.2
SGRv2 w/o decoder	21.3	4.4	68.4	12.4	38.8	32.8	27.2	35.6	6.8
SGRv2 w/ absolute pos prediction	21.0	8.0	57.6	21.2	17.2	14.8	21.2	44.4	4.4
SGRv2 w/ uniform point weight	44.2	21.6	82.4	28.8	32.0	60.4	68.0	44.0	14.0
SGRv2 w/o dense supervision	40.0	6.4	59.2	6.8	54.4	78.4	60.8	68.0	6.0
Method	Open Drawer	Slide Block	Sweep To Dustpan	Meat Off Grill	Turn Tap	Put In Drawer	Close Jar	Drag Stick	Stack Blocks
SGRv2	81.3	100.0	61.5	96.5	87.9	75.9	25.6	94.9	17.5
SGRv2 w/o decoder	14.0	78.8	15.6	40.8	46.8	4.0	18.8	0.8	0.0
SGRv2 w/ absolute pos prediction	20.4	99.2	50.8	10.8	68.8	4.0	6.4	54.8	1.2
SGRv2 w/ uniform point weight	92.8	100.0	72.8	90.8	67.6	74.0	43.6	97.6	1.6
SGRv2 w/o dense supervision	75.2	92.8	19.2	72.0	84.0	42.0	28.8	59.6	43.2
Method	Screw Bulb	Put In Safe	Place Wine	Put In Cupboard	Sort Shape	Push Buttons	Insert Peg	Stack Cups	Place Cups
SGRv2	24.1	55.6	53.1	20.3	1.9	93.2	4.1	21.3	1.6
SGRv2 w/o decoder	3.2	16.4	27.6	0.0	0.0	56.8	0.4	0.8	1.6
SGRv2 w/ absolute pos prediction	0.4	1.6	3.2	0.0	0.0	35.2	1.6	0.0	0.0
SGRv2 w/ uniform point weight	1.6	32.8	50.4	1.2	0.8	64.4	0.0	5.2	0.8
SGRv2 w/o dense supervision	0.0	30.4	52.0	0.4	0.0	100.0	0.0	0.0	2.4

Table 7: Ablations results (%) for SGRv2 on RL Bench with metrics for each task.