A Task Details

Setup. Our simulation experiments are conducted on Robot Learning Benchmark (RLBench) environment [32].

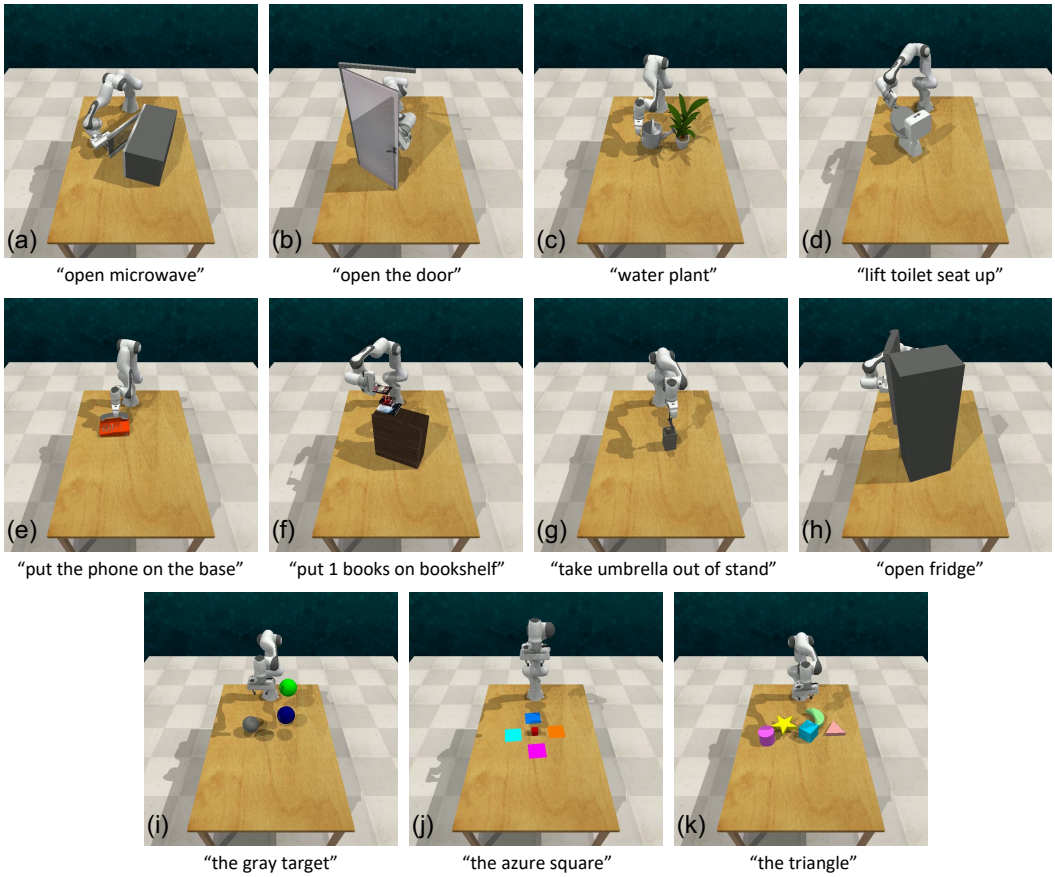

Figure 5: **RLBench Tasks.** Our simulation experiments encompass 11 tasks from RLBench. Tasks (a)-(h) serve as the basis for single-task and multi-task learning, while tasks (i)-(k) are employed to evaluate generalization performance.

For single-task and multi-task learning, we use 8 tasks that closely mirror real-world scenarios, for example, our tasks incorporate objects like microwaves, plants, phones, books, and toilets that align more closely with the reality of domestic robot environments. Our aim is to leverage the powerful semantic understanding and spatial reasoning capabilities of SGR to facilitate the manipulation of a diverse and complex array of objects in real-world situations.

For the generalization part, we conduct experiments on 3 tasks characterized by a substantial amount of visual variations. Among the 3 tasks, two focus on color generalization, while the remaining one is centered around shape generalization. The color variations include 20 instances:`colors` = {`red`, `maroon`, `lime`, `green`, `blue`, `navy`, `yellow`, `cyan`, `magenta`, `silver`, `gray`, `orange`, `olive`, `purple`, `teal`, `azure`, `violet`, `rose`, `black`, `white`}. The shape variations include 5 instances: `shapes` = {`cube`, `cylinder`, `triangle`, `star`, `moon`}.

As depicted in Figure 5, we utilize a diverse set of simulation tasks. In the following sections, we will provide a detailed examination of each task, and for those tasks that have been modified compared with the original RLBench implementation[2], we will discuss the specifics of the modifications in detail.

[2] https://github.com/stepjam/RLBench

A.1 Open Microwave

Filename: `open_microwave.py`

Task: Open the microwave on the table.

Modified: No.

Objects: 1 microwave.

A.2 Open Door

Filename: `open_door.py`

Task: Grip the handle and push the door open.

Modified: No.

Objects: 1 door.

A.3 Water Plants

Filename: `water_plants.py`

Task: Pick up the watering can by its handle and water the plant.

Modified: No.

Objects: 1 watering can and 1 plant.

A.4 Toilet Seat Up

Filename: `toilet_seat_up.py`

Task: Grip the edge of the toilet seat and lift it up to an upright position.

Modified: No.

Objects: 1 toilet with lid closed.

A.5 Phone on Base

Filename: `phone_on_base.py`

Task: Grasp the phone and put it on the base.

Modified: No.

Objects: 1 phone and 1 phone base.

A.6 Put Books

Filename: `put_books_on_bookshelf.py`

Task: Pick up a book and place them on the top shelf.

Modified: No. The original task has 3 variations, here we only use the first variation.

Objects: 1 bookshelf and 3 books.

A.7 Take out Umbrella

Filename: `take_umbrella_out_of_umbrella_stand.py`

Task: Grasp the umbrella by its handle, lift it up and out of the stand.

Modified: No.

Objects: 1 umbrella and 1 umbrella stand.

A.8 Open Fridge

Filename: `open_fridge.py`

Task: Grip the handle and slide the fridge door open.

Modified: No.

Objects: 1 fridge with door closed.

A.9 Reach Target

Filename: `reach_target.py`

Task: Reach the language-instructed colored ball. There are 3 balls, distinguishable solely by their colors, which are randomly distributed in space. The 3 colors are sampled from the full set of 20 color instances, thus offering 20 variations based on the target color.

Modified: Yes. The sizes of the balls are enlarged so that they are distinguishable in the RGB-D input.

Objects: 3 balls.

A.10 Slide Block

Filename: `slide_block_to_color_target.py`

Task: Slide the block to the language-instructed colored square targets. There are 3 distractor target of different colors. The 4 targets are placed symmetrically and the colors are sampled from the full set of 20 color instances, which makes the 20 variations.

Modified: Yes. The original `slide_block_to_target.py` task contains only one target. Three other targets are added.

Objects: 1 block and 4 colored target squares.

A.11 Reach Shape

Filename: `reach_shape.py`

Task: Reach the language-instructed shape of the block. There are always 1 target shape and 4 distractor shapes, which are from the 5 shape instances and randomly placed on the table. The 5 different target shapes make the 5 variations.

Modified: Yes, newly added task.

Objects: 5 shapes.

B SGR Details

B.1 Architecture Details

Input Data. The SGR model takes as input RGB images $\{I_i\}_{i=1}^{K}$ of size $H \times W$ and corresponding depth images of the same size from multiple camera views. Then point cloud, represented in the robot's base frame, is derived from depth images with known camera extrinsics and intrinsics. A key detail here is that the RGB images and the organised point cloud are aligned, ensuring a one-to-one correspondence between the points across the two data forms. In the simulation setting, we have $H = W = 128$ and $K = 4$, corresponding to four camera positions: front, left shoulder, right

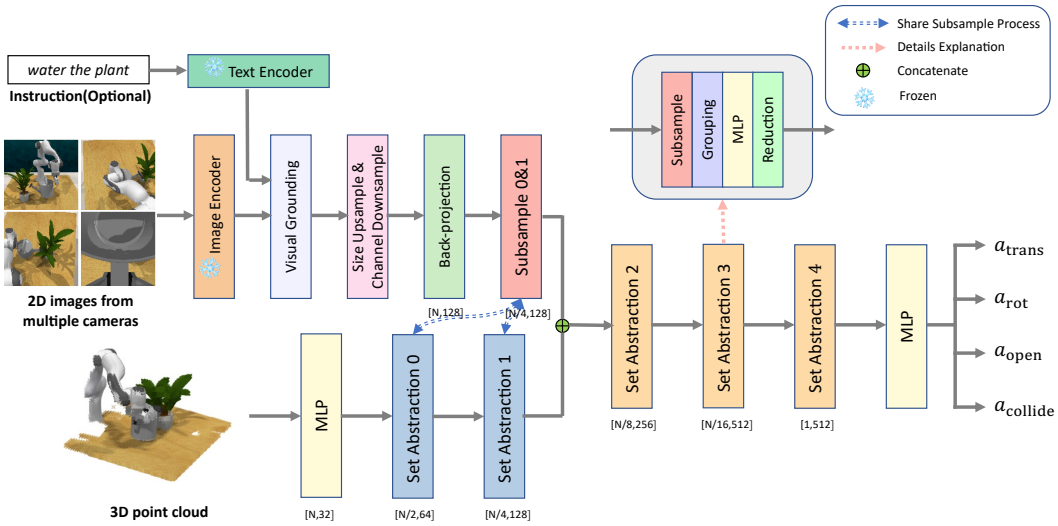

Figure 6: **Architecture Details.**

shoulder, and wrist. For real robot experiments, the setup is $H = W = 256$ and $K = 1$, denoting a single front-facing camera.

The model also receives proprioceptive data z, comprising four scalar values: gripper open state, left finger joint position, right finger joint position, and action sequence timestep. Additionally, if a task comes with language instruction S, this also forms part of the model's input.

Visual Grounding Module. The fundamental role of a visual grounding module is to localize visual areas that correspond to objects possessing a property as dictated by language. To equip our manipulation model with a deep comprehension of a variety of object properties and spatial relationships, including those not previously seen in the manipulation demonstration data, we take inspiration from MaskCLIP [69] and PROGRAMPORT [70].

It's worth noting that vanilla CLIP-ResNet-50 employs attention pooling to merge visual features from various spatial locations into one feature vector, and then uses a linear layer $\mathcal{F}$ as follows:

$$f_{\text{global}} = \mathcal{F}\left(\sum_i \text{softmax}\left(\frac{\text{Emb}_{\text{q}}(\text{AvgPool}\,(X)) \cdot \text{Emb}_{\text{k}}\,(X_i)^T}{C}\right)\text{Emb}_{\text{v}}\,(X_i)\right) \tag{2}$$

where X_i is the input feature at spatial location i, $\text{Emb}_{q,k,v}(\cdot)$ are linear embedding layers and C is some fixed scalar.

Following MaskCLIP [69] and PROGRAMPORT [70], we establish two 1×1 convolution layers, $\mathcal{C}_v$ and $\mathcal{C}_l$, which are initialized with the weights derived from Emb_{v} and $\mathcal{F}$ respectively. Applying these layers in sequence enables the visual feature to be projected into the same space as the language feature. Ultimately, our grounding module formulates a dense mask encompassing image segments corresponding to the language S in the following manner:

$$f_{\text{align}} = \sum (\text{TILE}(\mathcal{H}(S)) \odot \mathcal{C}_l(\mathcal{C}_v(X))) \in \mathbb{R}^{H^{'} \times W^{'} \times 1} \tag{3}$$

where $\mathcal{H}(S) \in \mathbb{R}^D$, $\mathcal{C}_l(\mathcal{C}_v(X)) \in \mathbb{R}^{H^{'} \times W^{'} \times D}$ ($H^{'}$, $W^{'}$, D represent the height, width, and dimension of the pre-aligned feature respectively). $\mathcal{H}$ is the CLIP language encoder and TILE serves as a spatial broadcast operator that expands the language embedding vector to match the spatial dimensions of the visual features. The symbol $\odot$ indicates an element-wise multiplication, while the summation is performed over the feature dimension D. Additionally, we concatenate this feature with the pre-aligned image feature to generate our aligned feature maps:

$$M = \text{Concat}(f_{\text{align}}, \mathcal{C}_l(\mathcal{C}_v(X))) \in \mathbb{R}^{H^{'} \times W^{'} \times (1+D)} \tag{4}$$

Other Details. For the image encoder, we employ CLIP-ResNet-50. To maintain consistency with the resolution predominantly used in the pre-training of vision models, we upsample the input image size from 128×128 to 256×256 in simulation experiments. The CLIP text feature exhibits a dimensionality of 1024, and the CLIP image encoder generates a feature map of 8×8 with 2048 channels. For the 3D encoder, we utilize PointNeXt-S. Detailed information regarding the number of channels and sub-points in each layer of the PointNeXt network can be seen in Figure 6.

B.2 Training Details

Data Augmentation. To further improve the performance and robustness of our model, we incorporate a variety of data augmentation techniques. (1) **Translation perturbations**: during training, the translation is perturbed with ± 0.125 m. We also experimented with rotation perturbations but found they did not contribute to performance improvement. (2) **Color drop** is to randomly replace colors with zero values. This technique serves as a powerful augmentation for PointNeXt, leading to significant enhancements in the performance of tasks where color information is available. Qian et al. [29] suggest that by implementing color drop, the network is encouraged to pay more attention to the geometric relationships between points, leading to a boost in overall performance. (3) **Point resampling** is a common technique utilized in the processing of point cloud data, often applied to adjust the density of the point cloud. The technique involves selecting a subset of points from the original data, thereby creating a new dataset with altered density. In our simulation experiment, we resample 4096 points from the original point cloud. (4) **Keyframe discovery and demo augmentation** [50] [33] are to convert every data point within a demonstration into a task that involves predicting the subsequent next-best keyframe action. Keyframes are extracted by a heuristic method: an action is a keyframe if the joint velocities are close to zero and the gripper open state is unchanged. This keyframe discovery strategy bypasses the need to predict long sequences of noisy actions, making policy learning more efficient.

Hyperparameters. The configuration of hyperparameters applied in our studies can be found in Table 3. The single-task learning and generalization experiments are trained on a single NVIDIA GeForce RTX 3090 GPU, while the multi-task learning is trained on 8 GPUs. Since color plays a crucial role in some tasks and disregarding color information would render these tasks unsolvable, such as `reach target` and `slide block`, the color drop is not utilized in the generalization experiments.

Table 3: Hyper-parameters used in our simulation experiments.

Config	Single-task	Multi-task	Generalization
Training iterations	$40,000$	$40,000$	$40,000$
Leraning rate	0.0003	0.001	0.0003
Batch size	32	256	32
Optimizer	AdamW	AdamW	AdamW
Weight decay	1×10^{-6}	1×10^{-6}	1×10^{-6}
Color drop	0.2	0.4	0
Instruction provided	No	Yes	Yes
Number of input points	4096	4096	4096

C Evaluation Workflow

Simulation. In pursuit of the reliability and stability of our results, our evaluation follows such subsequent steps. (1) Train the agent on the train set for 40,000 training iterations and save checkpoints every 1,000 iterations. (2) Evaluate the last 20 checkpoints for 25 episodes. (3) Evaluate the best 3 checkpoints for 100 episodes, and use the average performance of the 3 checkpoints as the results. (4) Repeat the above steps for 3 random seeds.

Method	open microwave	open door	water plants	toilet seat up	phone on base	put books	take out umbrella	open fridge	**Average**
R3M	7.0 ± 5.9	24.2 ± 10.9	1.2 ± 0.7	59.7 ± 21.7	0.0 ± 0.0	15.3 ± 16.5	28.3 ± 19.0	4.7 ± 3.8	17.6
CLIP	9.1 ± 10.8	68.4 ± 8.8	4.1 ± 1.8	40.1 ± 20.8	7.3 ± 2.5	38.0 ± 16.3	66.7 ± 8.2	7.4 ± 2.8	30.2
PointNeXt$_{\text{ULIP}}$	9.4 ± 1.6	60.0 ± 5.5	10.4 ± 1.6	63.0 ± 7.9	35.6 ± 4.8	41.8 ± 1.7	49.8 ± 4.2	15.3 ± 1.2	35.7
PointNeXt$_{\text{LfS}}$	17.0 ± 5.4	63.8 ± 6.2	28.2 ± 2.2	52.7 ± 14.5	65.9 ± 19.9	53.1 ± 16.0	$\mathbf{95.1} \pm 3.0$	7.3 ± 4.7	47.9
PERACT	26.9 ± 4.3	47.8 ± 41.1	$\mathbf{41.7} \pm 3.1$	67.1 ± 4.2	0.0 ± 0.0	63.7 ± 3.5	94.4 ± 1.3	10.1 ± 3.9	44.0
SGR (Ours)	$\mathbf{52.6} \pm 5.1$	$\mathbf{80.2} \pm 2.7$	40.2 ± 8.3	$\mathbf{80.1} \pm 1.2$	$\mathbf{81.1} \pm 5.7$	$\mathbf{84.8} \pm 7.4$	94.7 ± 2.3	$\mathbf{34.8} \pm 2.0$	**68.6**

Table 4: Single-task results with mean and standard deviation (%).

Method	open microwave	open door	water plants	toilet seat up	phone on base	put books	take out umbrella	open fridge	**Average**
CLIP	6.0 ± 6.9	78.0 ± 2.0	6.7 ± 6.1	7.3 ± 8.1	10.0 ± 10.4	38.0 ± 21.2	15.3 ± 13.3	9.3 ± 5.0	21.3
PointNeXt$_{\text{LfS}}$	22.7 ± 1.2	80.0 ± 11.1	19.3 ± 3.1	32.0 ± 20.0	69.3 ± 12.2	65.3 ± 13.3	72.0 ± 10.0	18.0 ± 2.0	47.3
PERACT	22.7 ± 2.3	58.7 ± 11.4	8.0 ± 10.6	64.7 ± 12.9	0.0 ± 0.0	47.3 ± 18.1	$\mathbf{96.0} \pm 2.0$	4.0 ± 3.5	37.7
SGR (Ours)	$\mathbf{38.7} \pm 15.0$	$\mathbf{84.0} \pm 3.5$	$\mathbf{20.7} \pm 1.2$	$\mathbf{67.3} \pm 10.3$	$\mathbf{86.0} \pm 10.6$	$\mathbf{80.0} \pm 15.6$	88.0 ± 6.9	$\mathbf{26.7} \pm 11.7$	**61.4**

Table 5: Multi-task results with mean and standard deviation (%).

	reach target		slide block		reach shape	
Method	Seen	Unseen	Seen	Unseen	Seen	Unseen
R3M	20.2 ± 6.2	5.5 ± 0.9	11.0 ± 5.9	3.0 ± 2.8	78.1 ± 24.5	6.7 ± 8.2
CLIP	42.0 ± 27.0	17.7 ± 4.8	49.0 ± 14.3	16.3 ± 2.8	95.5 ± 2.8	0.2 ± 0.3
PointNeXt$_{\text{ULIP}}$	-	-	-	-	45.9 ± 2.0	6.7 ± 4.7
PointNeXt$_{\text{LfS}}$	99.7 ± 0.3	4.0 ± 2.3	96.5 ± 1.0	6.2 ± 0.8	99.8 ± 0.4	1.8 ± 1.6
DenseFusion	71.8 ± 4.0	9.5 ± 3.1	26.5 ± 1.0	16.5 ± 1.5	99.3 ± 0.6	3.8 ± 3.4
SGR (Ours)	94.0 ± 2.6	$\mathbf{36.8} \pm 1.6$	97.7 ± 1.3	$\mathbf{43.0} \pm 3.0$	99.9 ± 0.2	$\mathbf{27.7} \pm 4.0$

Table 6: Generalization results with mean and standard deviation (%) .

Real-Robot. For the real-robot experiments, we train the agent for 40,000 iterations and simply choose the last checkpoint for evaluation, since it's expensive to evaluate more checkpoints in the real robot. We evaluate 10 episodes per task. During the evaluation of a trained agent, the agent keeps acting until achieving the goal or reaching the maximum episode length.

D Real-Robot Setup

For our real-robot experiments, we use a Franka Emika Panda manipulator equipped with a parallel gripper. Perception is achieved through an Intel RealSense D415 camera, positioned in front of the scene. The camera generates RGB-D images with a resolution of 1280×720. We leverage the `realsense-ros`[3] to align depth images with color images. The extrinsic calibration between the camera frame and robot base frame is carried out using the `easy_handeye` package[4].

When preprocessing the RGB-D images, we initially crop the 1280×720 images to a 720×720 frame, and then resize them to 256×256 using nearest-neighbor interpolation. This interpolation is preferred over others, such as bilinear interpolation, as the latter can cause non-existent points in the depth map, leading to a noisy point cloud. Following these steps, we can process RGB-D images in the same manner as in our simulation experiments. It's important to note that the camera's intrinsics must be adjusted accordingly after the images are cropped and resized. We train SGR for 40,000 training steps with 64 demonstrations in total and use the final checkpoint for inference.

E Additional Results

For all simulation experiments, we use 3 random seeds to ensure the reliability of our results. While we present averaged results in the main body of the paper for clarity, we provide more comprehensive results in this section. Here, we report both the mean and standard deviation derived from our

[3] https://github.com/IntelRealSense/realsense-ros

[4] https://github.com/IFL-CAMP/easy_handeye

simulation results to offer a complete view of our experimental outcomes. Table 4, Table 5, and Table 6 show the results of single-task learning, multi-task learning, and generalization, respectively.