

## A Implementation Details

### A.1 VLM Task Proposals & Success Detection

#### A.1.1 Task Proposals

The task proposal problem can be defined formally as a mapping from the space of images of the robot’s environment  $I$  to the space of language tasks  $T$ . So as to account for the current capabilities of open-source VLMs and the limitation on tasks that can actually be physically completed by the robot, we strict  $T$  to be a discrete set of tasks for each environment the robot is placed in:  $T = \{\tau_1, \tau_2, \dots, \tau_{|T|}\}$ .

We leverage the CogVLM [24] open-source VLM for both task proposals and success detection. Like many open-source VLMs, CogVLM is demonstrated to work particularly well on a set of popular image-understanding benchmarks, many of which are Visual Question Answering (VQA) benchmarks [81, 82, 83, 84, 85]. We leverage the model’s good performance on VQA tasks by converting the problem of task proposals into a VQA problem. Table 2 depicts how a prompt for task proposal can be reformulated into a simpler VQA-style prompt:

Task Proposal Prompt	VQA Prompt
<b>Original:</b> move the orange crayon from the blue plate to the table	<b>VQA-style:</b> Is the orange crayon currently on the blue plate?  <b>Answer that implies task is feasible:</b> True
<b>Original:</b> put the orange crayon on the cloth	<b>VQA-style:</b> Is the orange crayon on top of the cloth?  <b>Answer that implies task is feasible:</b> False
<b>Original:</b> move the red object from the blue plate to the table	<b>VQA-style:</b> Is the red object on the blue plate?  <b>Answer that implies task is feasible:</b> True

Table 2: Task Proposals to VQA

Using this framework, to actually propose a task, first the feasibility of each task is determined by few-shot prompting an LLM to transform task strings from  $T$  into VQA-style prompts. These VQA-style prompts are fed to the VLM and the response is again decoded by an LLM. Task feasibility is then determined by matching the final LLM output with the answer that would imply task feasibility. With this procedure, the set of viable tasks  $T_{\text{viable}} \subseteq T$  is constructed, after which the upper-confidence bound ranking procedure outlined in section 3 is used to pick a task.

#### A.1.2 Success Detection

We leverage a framework very similar to that used for task proposals for success detection. Although not always the case, it is very often the case that the same VQA-style question for task proposal can be used for success detection, with the answer implying success the opposite of the answer that implies task feasibility. This is because a common reason for not proposing a task is that it has already succeeded.

### A.2 Goal-conditioned Policy

#### A.2.1 Model Architecture

Here we outline the neural network architecture and training details of our goal-conditioned policy. The image of the current observation and desired goal observation are frame stacked along the channel dimension and fed through a ResNet-34 [86]. Instead of the usual BatchNorm [87] we

utilize GroupNorm [88] in the ResNet. Following the ResNet is a 3-layer MLP which outputs the mean and standard deviation parameterizing a Gaussian action distribution. Each hidden layer has dimension 256 and uses Swish [89] activations. In practice we output just the mean (and fix the standard deviation to be state-independent).

### 631 A.2.2 Training

To pre-train a base goal-conditioned policy on BridgeData v2 we use the Adam [90] optimizer with cosine learning rate decay from an initial 0.0003 to 0 over the course of 500k gradient steps. We also use linear learning rate warmup for 2000 gradient steps. We use a L2 weight decay of 0.001 and for Adam use  $\beta_1 = 0.9$  and  $\beta_2 = 0.98$ , along with clipping gradients to have maximum norm of 1.0. Before channel-wise concatenation the current and goal images are processed via a standard series of image augmentations, including random resized cropping, brightness, contrast, saturation, and hue augmentations. During pre-training, goal images are sampled uniformly at random from 0 to 24 timesteps into the future, approximately matching the subgoal horizon the image subgoal generator SuSIE was trained with.

To train an improved GC-policy leveraging the autonomous data, we co-train on the autonomous data and pre-training dataset (BridgeData v2). For the improved policies trained on individual scenes, we up sample the autonomous data  $10\times$  with respect to its proportion to the pre-training data. (e.g., When the per-scene autonomous data is the size of 3% of the pre-training dataset, we use a sampling ratio of 30% for autonomous data and 70% for the pretraining data. For the generalist GC-policy trained on all the autonomous data, the dataset sampling ratio is 80% for the pre-training data and 20% for autonomous data. Following the approach used in BC-Zero [59], data from the autonomous data is relabeled with actions being the sum of two consecutive actions ( $a'_t \leftarrow a_t + a_{t+1}$ ). This counteracts the tendency of the robot during autonomous data collection to take lower magnitude actions than those in the pre-training dataset, a behavior that results from the Gaussian MLP head averaging modes in the state-conditioned action distribution. Like during the pre-training stage goals are sampled uniformly at random  $[0, 24]$  timesteps into the future for the pre-training data, and for the autonomous data goals are sampled  $[0, 12]$  timesteps into the future.

## 654 A.3 Language-conditioned Policy

### 655 A.3.1 Model Architecture

The language-conditioned policy architecture is a ResNet-34 with FiLM conditioning. Language instructions are first encoded by a frozen MUSE encoder and then passed through two fully connected layers. The image observation is passed through the ResNet which is conditioned on the language embedding via FiLM layers applied at the end of every ResNet block. The MLP action head is the same for the language-conditioned policy as the goal-conditioned policy.

### 661 A.3.2 Training

The training procedure of the language-conditioned policy is exactly the same as that for the goal-conditioned policy.

## 664 A.4 Image Subgoal Generation

We leverage SuSIE [25] as our language-conditioned image subgoal generator. During generation we use classifier-free guidance with a weight of 2.0 for the image and 7.5 for the text prompt. Each autonomous trajectory consists of 5 subgoal generations with the low-level policy given 20 timesteps (identical to the horizon SuSIE was trained with) to reach the subgoal.

Generating a sequence of subgoal images one after another offers various practical advantages over creating a single final goal image. Particularly for autonomous robot deployment, a significant benefit arises when the robot’s actions alter the environment in a manner unrelated to the intended goal. In such cases, iterative regeneration can seamlessly integrate these environmental changes into the generated subgoals, eliminating the necessity for the policy to reset the environment to achieve the desired goal image.

## B Robot Setups & Scene Descriptions







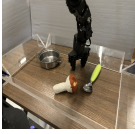


Scene #	Workspace image	Task Descriptions	# Autonomous Successful Trajectories
1		<ol style="list-style-type: none"> <li>1. put the green block in the wooden bowl</li> <li>2. remove the green block from inside the wooden bowl and put it on the table</li> <li>3. put the red fruit in the wooden bowl</li> <li>4. remove the red fruit from inside the wooden bowl and put it on the table</li> </ol>	2056
2		<ol style="list-style-type: none"> <li>1. put the purple eggplant in the brown bowl</li> <li>2. remove the purple eggplant from inside the brown bowl and put it on the table</li> </ol>	282
3		<ol style="list-style-type: none"> <li>1. move the green marker to the left side</li> <li>2. move the green marker to the right side</li> <li>3. put the blue block in the wooden bowl</li> <li>4. remove the blue block from inside the wooden bowl and put it on the table</li> <li>5. put the lemon in the wooden bowl</li> <li>6. remove the lemon from inside the wooden bowl and put it on the table</li> </ol>	364
4		<ol style="list-style-type: none"> <li>1. put the red object on the green plate</li> <li>2. take the red object out of the green plate and put it on the table</li> <li>3. put the carrot on the green plate</li> <li>4. take the carrot out of the green plate and put it on the table</li> </ol>	206
5		<ol style="list-style-type: none"> <li>1. open the drawer</li> <li>2. close the drawer</li> </ol>	221
6		<ol style="list-style-type: none"> <li>1. put the mushroom in the blue bowl</li> <li>2. remove the mushroom from the blue bowl and put it on the table</li> </ol>	46
7		<ol style="list-style-type: none"> <li>1. put the mushroom in the metal pot</li> <li>2. remove the mushroom from the metal pot and put it on the table</li> <li>3. move the green spoon to the left</li> <li>4. move the green spoon to the right</li> </ol>	82
8		<ol style="list-style-type: none"> <li>1. put the carrot on the blue plate</li> <li>2. remove the carrot from the blue plate and put it on the table</li> <li>3. put the purple eggplant on the blue plate</li> <li>4. remove the purple eggplant from the blue plate and put it on the table</li> <li>5. put the lemon on the blue plate</li> <li>6. remove the lemon from the blue plate and put it on the table</li> </ol>	700
9		<ol style="list-style-type: none"> <li>1. put the green veggie on the blue plate</li> <li>2. remove the green veggie from the blue plate and put it on the table</li> <li>3. put the pink spoon on the blue plate</li> <li>4. remove the pink spoon from the blue plate and put it on the table</li> </ol>	200

Table 3: Details on 9 robot scenes

Table B lists the scenes, associated language tasks, and the number of successful autonomous trajectories collected for each of the 9 scenes. In each scene, we decide the objects to be placed in the scene and specify a list of meaningful language tasks, and SOAR autonomously proposes the tasks to self-practice.

## C SOAR-Data Details

Dataset	# Traj.	# Env.	Lang.	Failed Traj.	Public	Collection
RoboNet [65]	162k	10	×	×	✓	scripted
MT-Opt [18]	800k	1	×	✓	✓	scripted, learned
RGB Stacking [68]	400k	5	×	✓	×	learned
BridgeData V2 [12]	60.1k	24	✓	×	✓	human, scripted
RobotSet [91]	98.5k	11	✓	×	✓	human, scripted
<b>SOAR-Data</b>	25k	5	✓	✓	✓	100% autonomous

Table 4: SOAR-Data is a large and publicly available robotic manipulation dataset that is collected fully autonomously and includes both successful and failed trajectories. It has diverse scenes and all trajectories have language annotations. Uniquely among datasets containing autonomous data, the SOAR-Data setup is cheap and replicable, making it an appealing real-world benchmark for learning from suboptimal data.

## D Limitations

Here we highlight two limitations of our work which suggest promising directions for future research. Although SOAR effectively harnesses autonomous data to improve policies significantly, further improvement could be achieved by incorporating unsuccessful autonomous trajectories as training data. Additionally, while our results showcase the capacity of the SOAR framework to robustify existing skills on unseen environments, an interesting area of future work is acquiring skills not present in the pre-training dataset through devising strategies to explore and gather data conducive to learning these new skills.